Out-of-core Algorithms for Binary Partition Hierarchies

Josselin Lefèvre[1,2], Jean Cousty[1], Benjamin Perret[1], Harold Phelippeau[2]

[1]LIGM, Univ Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France.
[2]Thermo Fisher Scientific, Bordeaux, France.

Abstract

Binary partition hierarchies (BPH) and minimum spanning trees are essential data structures for hierarchical analysis, such as quasi-flat zones and watershed segmentation. Traditional BPH construction algorithms are limited by their requirement to load the data entirely into memory, making them impractical for processing large images whose processing exceeds the capacity of the computer's main memory. To overcome this limitation, an algebraic framework was introduced, enabling the out-of-core computation of BPH leveraging three key operations: *select*, *join*, and *insert*. In this publication, we present two distinct calculi based on these operations: one designed for general spatial partitions and another optimized for causal partitioning. The second calculus is specifically tailored to meet out-of-core constraints, ensuring efficient processing of large-scale data. We provide detailed algorithms, including pseudo-code and complexity analysis, and conduct experimental comparisons between the two approaches.

Keywords: Image segmentation, Watershed, Hierarchical analysis, Out of core, External memory algorithm

1 Introduction

Hierarchies of partitions are versatile representations that have proved to be useful in a wide variety of computer vision, image analysis, and processing problems. In this context, binary partition hierarchies [1, 2] (BPH) among them those built from altitude ordering and associated minimum spanning trees are key structures for several (hierarchical) segmentation methods: in particular, it has been shown [3, 4] that such hierarchies can be used to efficiently compute quasi-flat zone (also referred to as α-trees) hierarchies [3, 5] and watersheds, including its hierarchical variations [3, 6]. These variations found applications in multiple fields such as biology [7], composite material [8], historical buildings characterization [9], rock instability detection [10], or remote sensing [11, 12]. Efficient algorithms for building BPHs on standard size images are well established, but, with the constant improvement in acquisition systems comes a dramatic increase in image resolutions, resulting in images that can reach several terabytes in size. While a single image may fit into the main memory of a standard workstation, its processing, including notably auxiliary data structures, often does not, causing classical algorithms for BPHs to fail. Consequently, there is a need for scalable algorithms to construct BPHs in an out-of-core manner to handle these images.

This issue has spurred research across various fields [13]. The image processing community has explored methods such as the random walker algorithm [14], while topographic studies focus on calculating watersheds for elevation data [15, 16]. Additionally, there is relevant work in terrain rendering [17] and linear algebra [18], underscoring the broad interest in addressing the need for scalable solutions to handle massive datasets.

As regards hierarchical analysis, previous works [19–21] have explored shared and distributed memory algorithm to compute component trees for terabytes images. The computation of minimum spanning trees of streaming images has been considered [22], and parallel algorithms for the computation of quasi-flat zones hierarchies have been proposed [23]. Recently, massively parallel algorithms for computing max-trees on GPUs were introduced [24]. These works generally rely on processing small pieces of the space independently, "joining" the information from adjacent pieces, and "inserting" this joint information into other pieces. This methodology, which has proved its worth for these different problems, is interesting from a memory point of view and could be exploited for our purposes. Additionally, the Multi-Scale Component-Tree (MSCT) [25] offers a hierarchical representation that adapts to the local informativeness of the image, potentially enhancing the efficiency of handling very large images.

In [26], the authors specifically tackled the challenge of computing a BPH under the out-of-core constraint, *i.e.*, when the objective is to minimize the amount of memory required by the algorithms at each computation step. To achieve this, they introduced an algebraic framework formalizing the distribution of a hierarchy over a partition of the space. This distribution is actually a collection of smaller sub-hierarchies of the BPH, each one being associated with one region (also called a slice or a tile in the following, depending on the context) of the partition. In order to compute this distribution, they introduced a high level calculus encompassing three algebraic operations acting on hierarchies: *select*, *join*, and *insert*. They demonstrated that with a causal partition of the space, it is possible to compute the distribution of a BPH using these operations by browsing different slices of the partition twice (once in a forward pass and once in a backward pass), requiring only information about two adjacent slices in the main memory at any step of the algorithm. In [27], efficient algorithms and complexity analyses were provided for the *select*, *join*, and *insert* operations. This article extends [26] and [27]. In particular, we present two types of BPH computations adapted to large images. The first one is able to consider the distribution of the BPH over any generic partition of the space while the second one, originally presented in [26], deals only with causal partitions. We present the advantages and drawbacks of these two types of computations in the form of (i) properties (namely Properties 9 and 11) asserting to what extent the out-of-core constraint is respected by the second computation, (ii) a counter-example (namely counter-example 1) showing the theoretical limits of the first one, and (iii) an experimental study validating both approaches and experimentally comparing the proposed computation schemes. The implementation of the method in C++ and Python based on the hierarchical graph processing library Higra [28] is available online.[1]

This article is organized as follows. Section 2 provides an overview of existing works. Section 3 defines BPH. Section 4 recalls the notion of the distribution of a hierarchy and the calculus method that can be used to compute such a distribution over any partition and a causal partition of the space. Section 5.1 explains the proposed data structures. Section 5.1 presents the algorithms for the three operations *select*, *join*, and *insert*. Section 6 presents an experimental study of the proposed computation schemes. Finally, Section 7 concludes the work and offers some perspectives.

2 Prior Work

Among all the hierarchical structures used in mathematical morphology [29–31], component trees have undoubtedly received the most attention in terms of parallelism and distributed computing. While existing approaches predominantly focus on reducing computational time, our specific interest lies in methods that focus on optimizing communications.

Early attempts to parallelize the construction of component trees, more precisely of max-trees [32], use shared memory to construct a monolithic global component tree representation [33, 34]. These algorithms rely on a divide-and-conquer strategy: the image is divided into tiles for which a partial component tree is computed. Then, the partial component trees are merged together.

However, due to the continuing technological improvements of sensors, the volume of data has largely increased, which limits the use of shared memory. To address this problem, recent

[1] https://github.com/PerretB/Higra-distributed

work has introduced a novel paradigm involving distributed memory and computing [20]. Among them, Kazemier et al. [21] brought the idea of creating not a single component tree, but a forest of component trees, each tree associated with a tile. This new approach, which was subsequently improved to handle higher dynamic range data [19], allows each tree to be filtered in such a way that the result is the same as filtering the whole tree.

To facilitate communication between processes, these algorithms use a datastructure called as a *boundary tree*. This structure is simply the subtree of a partial component tree comprising every region intersecting the border of a tile. Hence, this structure is transmitted from a computation node to another instead of transmitting the entire partial component tree. Authors claim that these subtrees have significantly smaller memory footprint, often less than 1% of a partial tree in low dynamic range scenario. The method can be roughly summarized as follows:

1. Split the data into tiles and calculate a partial hierarchy on each tile using the usual algorithm;
2. Extract boundary tree from each partial hierarchy and merge them together;
3. Back-propagate the merged information for each partial hierarchy.

This study on component trees is echoed in a domain closely related with our research focus: quasi-flat zones hierarchies (QFZ) [35], also recognized as α-trees. While component trees rely on an order relation between pixel values, our interest lies in BPH, which is built upon a total ordering on edge values. Similarly, the construction of QFZ follows a comparable paradigm, with established connections between with BPH [3].

This latter study was combined with a parallel method for QFZ computation [36] resulting in an efficient algorithm outperforming previous ones [23]. While aligned with the aforementioned component trees algorithms, a unique aspect of their approach involves fusing two hierarchies within the edges-weighted graph framework by inserting edges to its correct rank.

We can get even closer to the structures of interest with the work of Gigli et al. [22], who tackle the problem of computing the MST of the union of two graphs with a non-empty intersection applied to image streaming. This calculation determines the *stable* part of the current MST, the part that will not be affected by future changes. And its *unstable* counterpart. This method even has a direct application to out-of-core computations, as the stable part can be stored on disk and not used in subsequent computations.

Among various strategies explored in the literature, the approaches proposed by Kazemier et al. [21] stand out as particularly noteworthy, given the memory-saving aspect. Therefore, we have drawn inspiration from the latter approach to formalize it within the framework of edge-weighted graphs, with a specific emphasis on memory constraints.

3 Binary Partition Hierarchy by Altitude Ordering

In this section, we first remind the definitions of hierarchy of partitions. Then we define the binary partition hierarchy by altitude ordering using the edge-addition operator [26] and we recall the bijection existing between the regions of this hierarchy and the edges of a minimum spanning tree of the graph.

Let V be a set. A *partition of* V is a set of pairwise disjoint subsets of V. Any element of a partition is called a *region* of this partition. The *ground* of a partition $\mathbf{P}$, denoted by $gr(\mathbf{P})$, is the union of the regions of $\mathbf{P}$. A partition whose ground is V is called a *complete partition of* V. Let $\mathbf{P}$ and $\mathbf{Q}$ be two partitions of V. We say that $\mathbf{Q}$ is a *refinement of* $\mathbf{P}$ if any region of $\mathbf{Q}$ is included in a region of $\mathbf{P}$. A *hierarchy on* V is a sequence $(\mathbf{P}_0, \ldots, \mathbf{P}_\ell)$ of partitions of V such that, for any λ in $\{0, \ldots, \ell - 1\}$, the partition $\mathbf{P}_\lambda$ is a refinement of $\mathbf{P}_{\lambda+1}$. Let $\mathcal{H} = (\mathbf{P}_0, \ldots, \mathbf{P}_\ell)$ be a hierarchy. The integer ℓ is called the *depth of* $\mathcal{H}$ and, for any λ in $\{0, \ldots, \ell\}$, the partition $\mathbf{P}_\lambda$ is called the λ*-scale of* $\mathcal{H}$. In the following, if λ is an integer in $\{0, \ldots, \ell\}$, we denote by $\mathcal{H}[\lambda]$ the λ-scale of $\mathcal{H}$. For any λ in $\{0, \ldots, \ell\}$, any region of the λ-scale of $\mathcal{H}$ is also called a *region of* $\mathcal{H}$. The hierarchy $\mathcal{H}$ is *complete* if $\mathcal{H}[0] = \{\{x\} \mid x \in V\}$ and if $\mathcal{H}[\ell] = \{V\}$. The *ground* of a hierarchy is the support of its 0-scale. The ground of a hierarchy $\mathcal{H}$ is denoted by $gr(\mathcal{H})$. We denote by $\mathcal{H}_\ell(V)$ the set of all hierarchies on V of depth ℓ, by $\mathcal{P}(V)$

the set of all partitions on V, and by $2^{|V|}$ the set of all subsets of V.

The binary partition hierarchy by altitude ordering, simply called binary partition hierarchy in the following, is associated with a graph and to an ordering of the edges of this graph. In general, for image or data segmentation, this ordering is obtained by weighting the edges of the graph (*e.g.*, with a dissimilarity measure between vertices) and by sorting them in increasing order of weight.

Important notation. *In this article, the symbol $G = (V, E)$ stands for a graph where V is a finite set and E is composed of unordered pairs of distinct elements in V. The elements of V are called the* vertices *of G while those of E are called* edges. *We denote by ℓ the number of edges of G, that is, $\ell = |E|$. The symbol $\prec$ denotes a total order on E.*

Let k in $\{1, \ldots, \ell\}$, we denote by $u_k^{\prec}$ the k-th element of E for the order $\prec$. Let u be an edge in E, the *rank of u for $\prec$*, denoted by $r^{\prec}(u)$, is the unique integer k such that $u = u_k^{\prec}$.

Following [26], the binary partition hierarchy can be defined thanks to an elementary exterior operation on hierarchies, called an update. Let $\mathcal{H}$ be any hierarchy in $\mathcal{H}_\ell(V)$, and let $\{x, y\}$ be any edge of G. The *update of $\mathcal{H}$ with respect to $\{x, y\}$*, denoted by $\mathcal{H} \oplus \{x, y\}$, is the hierarchy such that:

- $\mathcal{H} \oplus \{x, y\}[\lambda]$ remains unchanged (*i.e.*, $\mathcal{H} \oplus \{x, y\}[\lambda] = \mathcal{H}[\lambda]$) for any λ in $\{0, k-1\}$, with k being the rank of $\{x, y\}$, while
- for any λ in $\{k, \ldots, \ell\}$, we have $(\mathcal{H} \oplus \{x, y\})[\lambda] = \mathcal{H}[\lambda] \setminus \{R_x, R_y\} \cup \{R_x \cup R_y\}$ where R_x (resp. R_y) denotes the region of $\mathcal{H}[\lambda]$ containing x (resp. y).

Let $E' \subseteq E$ and let $\mathcal{H}$ be a hierarchy. We set $\mathcal{H} \boxplus E' = \mathcal{H} \oplus u_1 \oplus \cdots \oplus u_{|E'|}$ where $E' = \{u_1, \ldots, u_{|E'|}\}$. The binary operation $\boxplus$ is called the *edge-addition*.

Definition 1 (Binary Partition Hierarchy by Altitude Ordering)**.** *Let V be a set, let $\bot_V$ the hierarchy defined by $\bot_V[\lambda] = \{\{v\} \mid v \in V\}$, for any λ in $\{0, \ldots, \ell\}$. The Binary Partition Hierarchy for $\prec$, denoted by $\mathcal{B}^{\prec}$ is the hierarchy $\bot_X \boxplus E$.*

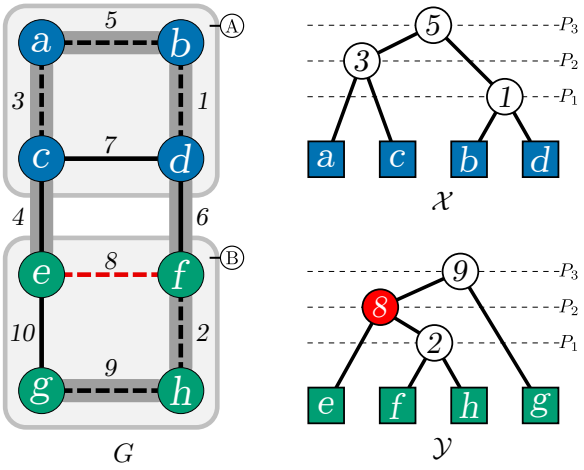


Fig. 1: G a 4-adjacency graph divided into two slices A and B (respectively, blue and green). Each edge of G is associated with its rank. The two slices are separated by their common neighborhood, *i.e.*, edges of ranks 4 and 6. Hierarchy $\mathcal{X}$ (respectively, $\mathcal{Y}$) is the BPH built on A (respectively, B). Ids associated with non-leaf nodes of the BPHs correspond to the weights of their corresponding building edges represented by dashed edges in G. We can note that MSTs built on slices (dashed edges) are not subtrees of the MST built on $A \cup B$ (shown as shadow). In consequence, edge of rank 8 (in red) is part of the hierarchy $\mathcal{Y}$ when it should not be.

Complete hierarchies are often represented with tree structures where each region is represented by a node in the tree. The root corresponds to the whole set V (*i.e.*, the unique region of the ℓ-scale), while the leaves correspond to singleton regions (*i.e.*, the regions of the 0-scale). A node p is the parent of a node n if the region corresponding to n is included in the region represented by p without any intermediary (*i.e.*, there is no proper superset of the former region that is proper subset of the latter). Accordingly, the root is the region without a parent and the leaves are the regions without any child.

Let R be a region of $\mathcal{B}^{\prec}$ and $R^\star$ be the set of non-leaf regions of $\mathcal{B}^{\prec}$. The *rank* of R, denoted by $r(R)$, is the lowest integer λ such that R is a region of $\mathcal{B}^{\prec}[\lambda]$. We consider the map μ from $R^\star$ in E such that, for any non-leaf region R of $\mathcal{B}^{\prec}$, we have $\mu^{\prec}(R) = u_{r(R)}^{\prec}$. We say that $\mu^{\prec}(R)$ is the *building edge* of R. When the order $\prec$ is obtained

from weights defined on the edges of G, the set of building edges of the binary partitions hierarchy defines a minimum spanning tree of this edge-weighted graph [3].

In Figure 1, $\mathcal{Y}$ is the BPH built on the 4-adjacency graph built on the region B. Non-leaf nodes of $\mathcal{Y}$ correspond to the edges of the minimum spanning tree of B (dashed edges).

4 Out-of-core Binary Partition Hierarchy

In this section, we first present the notion of a distribution of a hierarchy over a partition of the space in Section 4.1. Then, we present three operations which play the role of basic building blocks for building distributed hierarchies (Section 4.2). In Section 4.3, we define and study a first calculus for building a distributed BPH, while in Section 4.4 we present a second calculus that is less generic but for which some formal additional guarantees regarding its out-of-core abilities are established.

4.1 Distribution of Hierarchies

In this section, we define the structure we are seeking to build, namely the *distribution a BPH*. Intuitively, distributing a hierarchy consists in splitting it into a set of smaller hierarchies such that:

1. each smaller hierarchy corresponds to a selected subpart of the whole hierarchy that hits a slice of the graph; and
2. the initial hierarchy can be reconstructed by "gluing" those smaller hierarchies.

We first present the operations that allow for selecting a subset of a partition and of a hierarchy that hit a given part of the space, before providing the definition of a distribution of a hierarchy.

The operation *sel* is the map from $2^{|V|} \times \mathcal{P}(V)$ to $\mathcal{P}(V)$ which associates with any subset X of V and to any partition $\mathbf{P}$ of V the subset $sel(X, \mathbf{P})$ of $\mathbf{P}$ which contains every region of $\mathbf{P}$ that contains an element of X.

Definition 2 (Select)**.** *The operation* select *is the map from* $2^{|V|} \times \mathcal{H}_\ell(V)$ *in* $\mathcal{H}_\ell(V)$ *which associates with any subset* X *of* V *and to any hierarchy* $\mathcal{H}$ *on* V *the hierarchy* $select(X, \mathcal{H}) = (sel(X, \mathcal{H}[0]), \ldots, sel(X, \mathcal{H}[\ell]))$*. We say that* $select(X, \mathcal{H})$ *is the* selection of X in $\mathcal{H}$*.*

Intuitively, this operation consists in the extraction of a sub-hierarchy from a given hierarchy, comprising every region that intersects the input subset. Analogously, within the context of a tree representation of a hierarchy, it corresponds to selecting the subtree encompassing all selected leaves corresponding to the vertices of the input subset and their parents. In Figure 2, the hierarchy $\mathcal{X}$ corresponds to the result from applying the operation select $(\{a, b, c, d\}, \mathcal{H})$.

We can use this latter definition to define a representation consisting in the decomposition of a hierarchy in a set of smaller hierarchies.

Definition 3 (Distribution)**.** *Let* $\mathbf{P}$ *be a complete partition on* V *and let* $\mathcal{H}$ *be a hierarchy on* V*.* The distribution of $\mathcal{H}$ over $\mathbf{P}$ *is the set* $\{select(R, \mathcal{H}) \mid R \in \mathbf{P}\}$*.*

The distribution of a hierarchy is depicted on Figure 2 with $\{\mathcal{X}, \mathcal{Y}\}$ being the distribution of $\mathcal{H}$ on the partition $\{A, B\}$.

4.2 Fundamental Operations for Distributed Hierarchy

In this section, we recall the definition of two fundamental operations, *join* and *insert* introduced in [26]. These operations, together with *select* operation, enable the computation of BPH distributions in an out-of-core fashion.

Before introducing these two fundamental operations, let us recall some notions related to the neighborhood of regions in partitions and to the lattice of hierarchies which are necessary in the sequel.

We denote by γ the operator that maps to any two distinct subsets X and Y of V the subset of E made of every edge of E that contains exactly one vertex of X and exactly one vertex of Y, *i.e.*, $\gamma(X, Y) = \{\{x, y\} \in E \mid x \in X \wedge y \in Y\}$. The set $\gamma(X, Y)$ is called the *common neighborhood of* X *and* Y. For instance, in Figure 1, the common neighborhood of the two regions A and B contains the edges $\{c, e\}$ and $\{d, f\}$: $\gamma(A, B) = \{\{c, e\}, \{d, f\}\}$.

Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_\ell(V)$. We say that $\mathcal{Y}$ is *smaller than* $\mathcal{X}$ if, for any λ

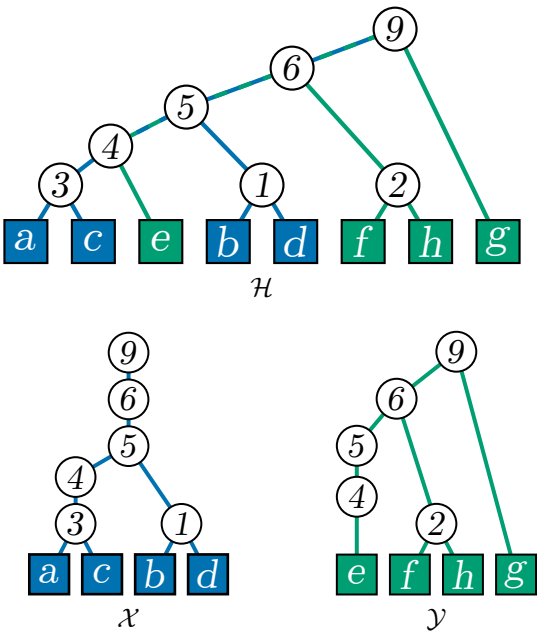


Fig. 2: A BPH $\mathcal{H}$ built on the graph in Figure 1 (on top) and its distribution on the bi-partition of the space $\{A, B\}$ with $A = \{a, b, c, d\}$ (blue) and $B = \{e, f, g, h\}$ (green). Elements of this distribution are $\mathcal{X} = \text{select}\,(A, \mathcal{H})$ and $\mathcal{Y} = \text{select}\,(B, \mathcal{H})$.

in $\{0, \ldots, \ell\}$, the partition $\mathcal{Y}[\lambda]$ is a refinement of $\mathcal{X}[\lambda]$. If $\mathcal{Y}$ is smaller than $\mathcal{X}$, we write $\mathcal{Y} \sqsubseteq \mathcal{X}$. The set $\mathcal{H}_\ell(V)$ equipped with the relation $\sqsubseteq$ is a lattice whose supremum is denoted by $\sqcup$ (see [37] for properties of this lattice and supremum and see Figure 3 for an illustration).

Definition 4 (Join)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies in $\mathcal{H}_\ell(V)$. The* join of $\mathcal{X}$ and $\mathcal{Y}$, *denoted by $join(\mathcal{X}, \mathcal{Y})$, is the hierarchy defined by $join(\mathcal{X}, \mathcal{Y}) = (\mathcal{X} \sqcup \mathcal{Y}) \boxplus F$, where F is the common neighborhood of the grounds of $\mathcal{X}$ and of $\mathcal{Y}$,* i.e., *$F = \gamma(gr(\mathcal{X}), gr(\mathcal{Y}))$.*

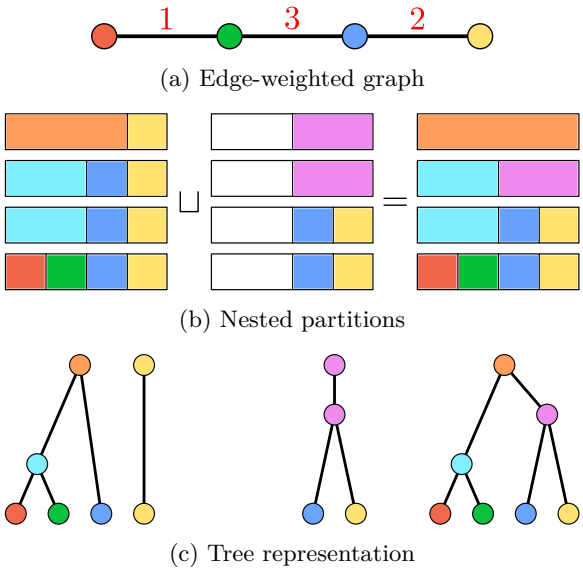
The operation join is illustrated in Figure 4. We can see that edges 4 and 6 are inserted at their proper rank in $\mathcal{J}$ discarding the region 7 which does not belong to the MST.

Definition 5 (Insert)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be two hierarchies.*

We say that $\mathcal{X}$ is insertable in $\mathcal{Y}$ *if, for any λ in $\{0, \ldots, \ell\}$, for any region Y of $\mathcal{Y}[\lambda]$, Y is either included in a region of $\mathcal{X}[\lambda]$ or is included in $V \backslash gr(\mathcal{X}[\lambda])$.*

Let $\mathcal{X}$ be insertable in $\mathcal{Y}$. The insertion of $\mathcal{X}$ into $\mathcal{Y}$ *is the hierarchy $\mathcal{Z}$, such that, for any λ in $\{0, \ldots, \ell\}$, $\mathcal{Z}[\lambda] = \mathcal{X}[\lambda] \cup \{R \in \mathcal{Y}[\lambda] \mid R \cap gr(\mathcal{X}[\lambda]) = \emptyset\}$. The insertion of $\mathcal{X}$ into $\mathcal{Y}$ is denoted by $insert(\mathcal{X}, \mathcal{Y})$.*

The insert operation is illustrated on Figure 5. We can see that $\mathcal{X}$ is *insertable* into $\mathcal{Y}$ because

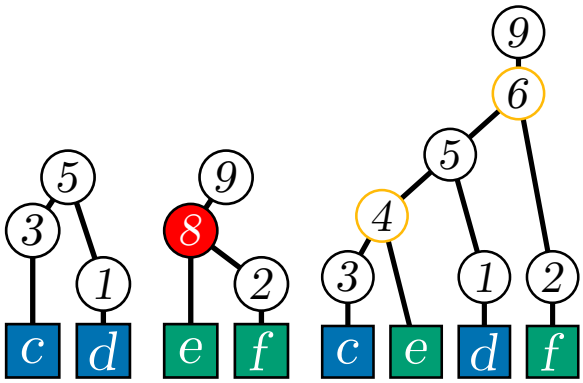


(a) Edge-weighted graph

(b) Nested partitions

(c) Tree representation

Fig. 3: Illustration of the supremum of two series of nested partitions.



(a) $\mathcal{X}$ (b) $\mathcal{Y}$ (c) $\mathcal{J} = join\,(\mathcal{X},\ \mathcal{Y})$

Fig. 4: *Join* of $\mathcal{X}$ and $\mathcal{Y}$ given their common neighborhood $\{\{d, f\}, \{c, e\}\}$ of ranks 6 and 4.

i) the two rightmost leaves of $\mathcal{Y}$ are included in $V \backslash gr(\mathcal{X}[\lambda])$ and ii) every other region of $\mathcal{Y}$ is included into a regions of $\mathcal{X}$ at the same scale. As a result, the region V of $\mathcal{Y}$ is discarded in $\mathcal{Z}$ because $V \cap gr(\mathcal{X}[5]) \neq \emptyset$ ($gr(\mathcal{X}[6])$ highlighted in red).

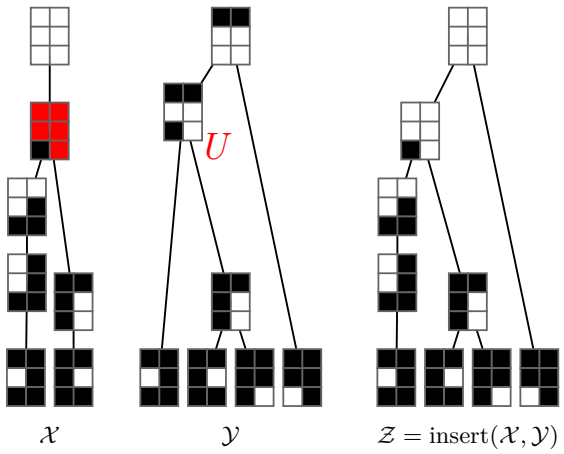


Fig. 5: Insertion of $\mathcal{X}$ into $\mathcal{Y}$ to obtain $\mathcal{Z}$. Each region is represented by the vertices of the input space included in this region in white. The region U of $\mathcal{Y}$ at altitude 6 is discarded from $\mathcal{Z}$ because of its non-null intersection with $gr(\mathcal{X}[6])$ (highlighted in red).

4.3 Computing Distribution for Generic Partitioning

In this section, we present a calculus to build the distribution of the BPH over any complete partition of the space, using the three select, insert, and join operations. This calculus relies on the construction of an intermediary structure that we call a *Global Border Tree* (GBT). After presenting, this calculus, we finish the section with a counter example showing the limitation of this approach for certain out-of-core computation contexts.

We denote by $\delta^\bullet$ the operator that maps any subset F of E to the subset of V made of every vertex of V that belongs to an edge in F, *i.e.*, $\delta^\bullet(F) = \cup F$.

Let A and B be any two subsets of V. We set $\gamma^\bullet_B(A) = \delta^\bullet(\gamma(A, B)) \cap A$. In other words, $\gamma^\bullet_B(A)$ contains every vertex in A which is adjacent to a vertex in B. If A is a subset of B, the set $\gamma^\bullet_{V\backslash A}(A)$ is called the *(internal) border of* A.

Definition 6 (Global Border Tree)**.** *Let* $\mathbf{P}$ *be a complete partition on* V*, and let* $\mathcal{H}$ *be a hierarchy on* V*. Let* F *be the set made of every vertex which belongs to the border of a region of* $\mathbf{P}$*:* $F = \cup\{\gamma^\bullet_{V\backslash R}(R) \mid R \in \mathbf{P}\}$*. The* Global Border Tree of $\mathcal{H}$ over $\mathbf{P}$ *is the hierarchy select*$(F, \mathcal{H})$*.*

A detailed algorithm to build the global border tree associated with a partition $\mathbf{P}$ of V into tiles and to an ordering $\prec$ on the edges of G is presented in Algorithm 1. Intuitively, this algorithm aims at building the (local) BPH associated with each tile in $\mathbf{P}$ (line 4), selecting the border of each tile in these local BPH (line 5), and incrementally joining them together to form the global border tree (line 6). In order to build the local BPH, we call the algorithm presented in [4], hereafter denoted by PWK. The detailed algorithms to compute the results of join and selection operations are described in Section 5.

Algorithm 1: Global Border Tree

Data: A graph (V, E), a total order $\prec$ on E, and a partition $\mathbf{P} = \{S_0, \ldots, S_k\}$ of V
Result: The global border tree $\mathcal{M}_k$ of $\mathcal{B}^\prec$ over $\mathbf{P}$, and the set $\{\mathcal{B}^\prec_i, \ldots, \mathcal{B}^\prec_k\}$ of partial BPH associated to the tiles of $\mathbf{P}$.

1 Call PWK algorithm to compute $\mathcal{B}^\prec_{S_0}$
2 $\mathcal{M}_0 :=$ select $\left(\gamma^\bullet_{V\backslash S_0}(S_0), \mathcal{B}^\prec_{S_0}\right)$
3 **foreach** i ***from*** 1 ***to*** k **do**
4 Call PWK algorithm to compute $\mathcal{B}^\prec_{S_i}$
5 $\mathcal{S} :=$ select $\left(\gamma^\bullet_{V\backslash S_i}(S_i), \mathcal{B}^\prec_{S_i}\right)$
6 $\mathcal{M}_i :=$ join $(\mathcal{S},\ \mathcal{M}_{i-1})$

At each step i, we compute $\mathcal{S}$ being the border tree of the partial hierarchy of index i (line 5). Consequently, $\mathcal{M}_i$ is the *partial global border tree* computed on $\cup_{j=0}^{i}\{S_j\}$ so

$$\mathcal{M}_i = \text{select}\left(F, \mathcal{B}^\prec_{\cup\{S_j \mid j\in\{0,\ldots i\}\}}\right)$$

$$\text{with} \quad F = \bigcup_{j=0}^{i} \left\{ \gamma^{\bullet}_{V \setminus S_j}(S_j) \right\}$$

This latter is obtained by calling *join* on $\mathcal{M}_{i\text{-}1}$ and $\mathcal{S}$ on line 6.

Property 7. *Let $\mathbf{P} = (S_0, \ldots, S_k)$ be a complete partition of V, let $\mathcal{H}$ be a hierarchy on V, and let $\mathcal{M}$ be the GBT of $\mathcal{H}$ over $\mathbf{P}$. In Algorithm 2, $\mathcal{M}_k = \mathcal{M}$.*

Let us now present Algorithm 2 that allows one to obtain the distribution of the BPH, using the global border tree as an intermediary datastructure. Algorithm 2 starts by computing the global border tree and the partial BPHs associated with the tiles of the space by calling Algorithm 1. Then, the tiles are browsed (line 2) and, for each of them, we select its border in the global border tree (line 3) before inserting this selection in the partial BPH associated with the given tile (line 4).

Algorithm 2: GBT-based distributed BPH

Data: A graph (V, E), a total order $\prec$ on E, and a partition $\mathbf{P} = \{S_0, \ldots, S_k\}$ of V.
Result: The distribution $\{\mathcal{B}_0, \ldots, \mathcal{B}_k\}$ of the BPH $\mathcal{B}^{\prec}_V$ over $\mathbf{P}$.

1 Call Algorithm 1 to compute the GBT $\mathcal{M}$ and the set $\{\mathcal{B}^{\prec}_{S_0}, \ldots, \mathcal{B}^{\prec}_{S_k}\}$ of partial BPH associated to the tiles of $\mathbf{P}$
2 **foreach** i ***from*** 0 ***to*** k **do**
3 $\quad \mathcal{S} := \text{select}\left(\gamma^{\bullet}_{V \setminus S_i}(S_i), \mathcal{M}\right)$
4 $\quad \mathcal{B}_i := \text{insert}(\mathcal{S}, \mathcal{B}^{\prec}_i)$

It is noteworthy that the number of calls to the three operations remains linear with respect to the number of tiles in the partition of the space.

Another notable property is that this calculation scheme remains valid regardless of the partition of the space and the order in which tiles are considered as long as the borders and common neighborhoods of two tiles are known. This computation is also valid as long as, for each tile, we can store in the principal memory of the computer the triplet formed by the global border tree $\mathcal{M}$, the partial BPH $\mathcal{B}^{\prec}_i$, and the element $\mathcal{B}_i$ of the distribution associated with the given tile.

Despite these advantageous characteristics, this computation has the drawback of requiring to load in memory the global border tree which, in some particular cases, can grow arbitrarily. In the context of out-of-core computation, a desirable characteristic would be that the amount of memory necessary to a given step of the algorithm is bounded. In other words, one may wish that the number of regions of $S, \mathcal{B}^{\prec}_i$, and $\mathcal{M}$ does not exceed the size of $\mathcal{B}_i$, the element of the distribution associated with the i-th tile of the partition. By the very construction presented in Algorithm 2, it can be seen that S and $\mathcal{B}^{\prec}_{S_i}$ are always smaller than $\mathcal{B}_i$. However, unfortunately, this is not the case of $\mathcal{M}$, which can be larger than $\mathcal{B}_i$ and larger than the union of multiple elements of the distribution as shown by the following counterexample. We will see in the next section that if we consider a particular partition of the space, called a causal partition, then a second algorithm to compute the distribution of a BPH can be proposed for which this situation cannot occur, *i.e.*, in this case, the size of the datastructure needed at each computation step never exceeds the size of two consecutive elements of the resulting distribution.

Counterexample 1. Let us consider the set $V = \{x_0, \ldots, x_7\}$ of vertices and the edges $E = \{\{x_i, x_{i+1}\} \mid i \in \{0, \ldots 7\}\}$ ordered such that $\{x_0, x_1\} \prec \{x_1, x_2\} \prec \ldots \prec \{x_6, x_7\}$. The BPH for this ordering $\prec$ is the hierarchy $\mathcal{H} = (\mathcal{H}[0], \ldots, \mathcal{H}[6])$ such that $\mathcal{H}[0] = \{\{x_i\} \mid i \in \{0, \ldots, 7\}\}$ and $\mathcal{H}[7] = V$, and such that, for any $\lambda \in \{1, \ldots, 6\}$, we have:

$$\mathcal{H}[\lambda] = \{e_{\lambda\text{-}1}\} \cup \{\{x_i\} \mid i \in \{\lambda + 1, \ldots, 7\}\} \tag{1}$$

where

$$e_\lambda = \{x_i \mid i \in \{0, \ldots, \lambda + 1\}\} \tag{2}$$

A representation of $\mathcal{H}$ is shown in Figure 6. Let $\mathbf{P}$ be the partition of V such that

$$\mathbf{P} = \{P_0 = \{x_0, x_1\}, P_1 = \{x_2, x_3\}, \\ P_2 = \{x_4, x_5\}, P_3 = \{x_6, x_7\}\} \quad (3)$$

The distribution of $\mathcal{H}$ over $\mathbf{P}$ (represented on Figure 6) is the set of hierarchies

$$\{\mathcal{L}_i = \text{select}\,(P_i, \mathcal{H}) \mid i \in \{0, \ldots, 3\}\} \quad (4)$$

whose region sets are

$$R(\mathcal{L}_0) = \{\{x_0\}, \{x_1\}, e_0, \ldots, e_6\} \quad (5)$$
$$R(\mathcal{L}_1) = \{\{x_2\}, \{x_3\}, e_1, \ldots, e_6\} \quad (6)$$
$$R(\mathcal{L}_2) = \{\{x_4\}, \{x_5\}, e_3, \ldots, e_6\} \quad (7)$$
$$R(\mathcal{L}_3) = \{\{x_6\}, \{x_7\}, e_5, \ldots, e_6\} \quad (8)$$

The GBT $\mathcal{M}$ (highlighted in bold blue in Figure 6) contains the following regions.

$$R(\mathcal{M}) = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}, \\ \{x_6\}, e_0, e_1, e_2, e_3, e_4, e_5, e_6\} \quad (9)$$

We can indeed see that $R(\mathcal{M})$ contains 13 regions which are greater than the number of regions in $\mathcal{L}_3$ (since $|R(\mathcal{L}_3)| = 4$). We can further observe that the number of regions in $\mathcal{M}$ (in blue on the figure below) is greater than the sum of the number of regions in the two consecutive elements $\mathcal{L}_2$ and $\mathcal{L}_3$ of the distribution. Indeed, $|R(\mathcal{L}_2)| + |R(\mathcal{L}_3)| = 6 + 4 = 10$.

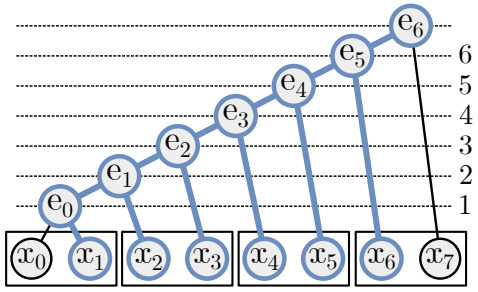


Fig. 6: Illustration of Counterexample 1. The partition of the space $\{x_0, \ldots, x_7\}$ into 4 tiles is represented by the black rectangles, the BPH is represented by the black and blue tree while the GBT is highlighted in bold blue.

4.4 Computing Distribution on Causal Partitions

In this section, we recall the principle of the out-of-core calculus of the distribution of the BPH over a causal partition introduced in [26]. The presented workflow is designed to limit the size of the data structures simultaneously present in memory. We ensure that at each stage the size of the computed structure is bounded by the size of neighboring elements of the distribution.

In the following, we consider the special case of a 4-adjacency graph representing a 2d image that can be divided into slices. It should be noted that this is not a limiting factor, and the method can easily be adapted to any regular grid graph such as 6 adjacency for 3d images considered in Section 4.

Let h and w be two integers representing the height and the width of an image. In the following, the set V is the Cartesian product of $\{0, \ldots, h$ - $1\} \times \{0, \ldots, w$ - $1\}$. Thus, any element x of V is a pair $x = (x_i, x_j)$ such that x_i and x_j are the coordinates of x. With the 4-adjacency relation, the set E of edges is equal to $\{\{x, y\} \in V \mid |x_i - y_i| + |x_j - y_j| \leq 1\}$.

Let k be a positive integer, the *causal partition of V (into k slices)* is the sequence $(S_0, \ldots, S_k)$ such that for any t in $\{0, \ldots, k\}$, $S_t = \{(i, j) \in V \mid t \times \frac{h}{k} \leq i < (t+1) \times \frac{h}{k},\ 0 \leq j < w\}$. Any element of a causal partition $mathbf{P}$ is called *a slice of* $\mathbf{P}$.

The major advantage of considering this specific partition is that due to its linear nature, any

slice of index k has at most two neighboring slices, namely the slices of index k+1 and k - 1.

Given any causal partition $\mathbf{P}$ of V, Algorithm 3 allows computing the distribution of the BPH $\mathcal{B}^{\prec}$ over $\mathbf{P}$.

This algorithm can be divided in two parts: a causal and an anti-causal traversal of the slices. Each of these parts relies on the same idea. First, start with the causal traversal. Given a causal partition of V into $k+1$ slices, for any i in $\{0,\ldots,k\}$ compute the BPH on S_i with a call to the algorithm PWK (line 3). Then, for any i in $\{1,\ldots,k\}$ *select* the part of this hierarchy containing the vertices adjacent to the previous slice and *join* it with the part of the hierarchy associated with the previous slice containing the vertices adjacent to the current slice, leading to the "*merged tree*" denoted by $\mathcal{M}_i^{\uparrow}$ (line 4). The merged tree is then *inserted* in the current BPH which gives $\mathcal{B}_i^{\uparrow}$ (line 5). The hierarchies $\mathcal{B}_i^{\uparrow}$ associated with slice i misses the information located in slices of higher indices, and consequently only the last local hierarchy $\mathcal{B}_k^{\uparrow}$ belongs to the distribution, *i.e.*, $\mathcal{B}_k^{\uparrow} = \text{select}\,(S_k, \mathcal{H})$.

In order to compute the whole distribution, and after having spread information in the causal direction, information must be back propagated in the reverse anti-causal direction so that each local hierarchy is enriched with the global context (lines 7 to 9). At each step i, we construct a new merged tree $\mathcal{M}_i^{\downarrow}$ between each pair of neighboring hierarchies, which this time takes into account all the information present in the input data. This merged tree is then inserted into $\mathcal{B}_i^{\uparrow}$ to form $\mathcal{B}_i^{\downarrow}$, which as shown in [26] is the element of the distribution associated with the slice S_i: $\mathcal{B}_i^{\downarrow} = \text{select}\,(S_i, \mathcal{B}^{\prec})$.

The next lemma, which can be deduced from Theorem 17 of [26] and Definition 5, shows that the intermediary datastructures $\mathcal{M}_i^{\uparrow}$ and $\mathcal{M}_i^{\downarrow}$ of Algorithm 3 can be expressed in a way similar to the intermediary datastructures $\mathcal{M}_i$ and $\mathcal{M}$ of Algorithms 1 and 2. Then, this allows us to compare these intermediary datastructures and to deduce notably that the intermediary datastructures of Algorithm 3 are always smaller than those of Algorithms 1 and 2 (see Property 10).

Lemma 8. *Let $(S_0,\ldots,S_k)$ be a causal partition of V. Let i be any element in $\{1,\ldots,k\}$. The following statements hold true:*

1. $\mathcal{M}_i^{\uparrow} = \textit{select}\left(\delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)), \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\ldots i\}\}}\right)$;
2. $\mathcal{M}_i^{\downarrow} = \textit{select}\,(\delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)), \mathcal{B}^{\prec}_V)$.

From Lemma 8, we deduce the following property which states that the intermediary datastructures of *Algorithm 3* are subparts of those of Algorithms 1 and 2.

Property 9. *Let $\mathbf{P} = (S_0,\ldots,S_k)$ be a causal partition of V. Let i be any element in $\{0,\ldots,k\}$. Let $\mathcal{M}$ be the Global Border Tree built on V over $\mathbf{P}$ and $\mathcal{M}_i$ be the Global Border Tree on $\cup_{j=0}^{i}\{S_j\}$. Then, we have $\mathcal{M}_i^{\uparrow} \sqsubseteq \mathcal{M}_i$ and $\mathcal{M}_i^{\downarrow} \sqsubseteq \mathcal{M}$.*

By considering Property 9, we can derive some important implications regarding the sizes of certain components within the GBT structures.

Corollary 10. *The sizes of the merged trees computed in Algorithm 3 are bounded as follows:*

$$\max_{1\le i\le k} |\mathcal{M}_i^{\uparrow}| \le \max_{1\le i\le k} |\mathcal{M}_i|$$

$$\max_{1\le i\le k} |\mathcal{M}_i^{\downarrow}| \le |\mathcal{M}|$$

The proofs of Property 9 and Corollary 10 are given in Appendix B, and Appendix C respectively.

The next property, which is the main result of this section, establishes that the size of the intermediary datastructures of Algorithm 3 is, at each step of the algorithm, smaller than the size of the datastructures that we seek to build at this step: in the step involving the construction of the element $\mathcal{B}_i^{\prec}$ of the distribution of the BPH, the size of the auxiliary datastructures never exceeds the size of the two consecutive elements $\mathcal{B}_{i-1}^{\prec}$ and $\mathcal{B}_i^{\prec}$. We recall that, as shown by Counterexample 1.

Property 11. *Let $(S_0,\ldots,S_k)$ be a causal partition of V and let $\mathcal{H}$ be the BPH built on V. Let i be any element in $\{1,\ldots,k\}$. Given $n_i = |\textit{select}\,(S_{i\text{-}1}, \mathcal{H})| + |\textit{select}\,(S_i, \mathcal{H})|$, we have $|\mathcal{M}_i^{\uparrow}| \le n_i$ and $|\mathcal{M}_i^{\downarrow}| \le n_i$.*

Algorithm 3: Out-of-core binary partition hierarchy for causal partitioning [26].

Data: A graph (V, E), a total order $\prec$ on E, and a causal partition $\mathbf{P} = (S_0, \ldots, S_k)$ of V
Result: The distribution $(\mathcal{B}_0^{\downarrow}, \ldots, \mathcal{B}_k^{\downarrow})$ of the BPH $\mathcal{B}_V^{\prec}$ over $\mathbf{P}$.

$\mathcal{B}_0^{\uparrow} := \mathcal{B}_{S_0}^{\prec}$ // call PWK algorithm
foreach i ***from*** 1 ***to*** k **do** // Causal traversal of the slices
  Call PWK algorithm to compute $\mathcal{B}_{S_i}^{\prec}$
  $\mathcal{M}_i^{\uparrow} := \text{join}\left(\text{select}\left(\gamma^{\bullet}_{S_i}(S_{i\text{-}1}), \mathcal{B}_{i\text{-}1}^{\uparrow}\right), \text{select}\left(\gamma^{\bullet}_{S_{i\text{-}1}}(S_i), \mathcal{B}_{S_i}^{\prec}\right)\right)$
  $\mathcal{B}_i^{\uparrow} := \text{insert}(\text{select}\left(\gamma^{\bullet}_{S_{i\text{-}1}}(S_i), \mathcal{M}_i^{\uparrow}\right), \mathcal{B}_{S_i}^{\prec})$
$\mathcal{B}_k^{\downarrow} := \mathcal{B}_k^{\uparrow}$; $\mathcal{M}_k^{\downarrow} := \mathcal{M}_k^{\uparrow}$
foreach i ***from*** $k-1$ ***to*** 0 **do** // Anticausal traversal of the slices
  $\mathcal{B}_i^{\downarrow} := \text{insert}(\text{select}\left(\gamma^{\bullet}_{S_{i+1}}(S_i), \mathcal{M}_{i+1}^{\downarrow}\right), \mathcal{B}_i^{\uparrow})$
  if $i > 0$ **then** $\mathcal{M}_i^{\downarrow} := \text{insert}(\text{select}\left(\gamma^{\bullet}_{S_{i\text{-}1}}(S_i), \mathcal{B}_i^{\downarrow}\right), \mathcal{M}_i^{\uparrow})$

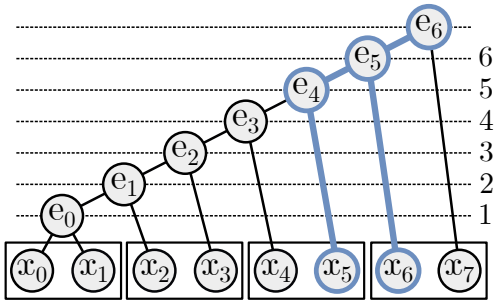


Fig. 7: Given a partition of the space in 4 slices, the BPH in black and the merged tree of the last two slices in bold blue.

Property 11 does not hold true in general when using Algorithms 1 and 2. A proof is given in Appendix D.

An insight into the proof of this property is observed in Figure 7. We can see the merged tree for the last two slices in bold blue which contains 5 nodes: 2 leaves and 3 non-leaf nodes. As a recall, the GBT associated with the very same BPH and partition presented in Figure 6 contains 13 nodes: 6 leaves and 7 non-leaf nodes. Then, the number of regions in the GBT is comparable to that of the entire BPH (15 nodes) while the merged tree is closer, in size, to a subtree of the BPH associated with one of its neighboring tiles.

Hence, unlike the calculation described in the previous section, only the information associated with two neighboring tiles is required at each stage. The merged tree computed at each iteration of both causal an anti-causal traversal is systematically smaller than equal to the GBT (strictly smaller when $|\mathbf{P}| > 2$). This makes this algorithm a more scalable method than the previous approach, at the cost of a higher number of steps. These practical considerations are addressed in Section 6.

5 Data Structures and Algorithms

In this section, we present algorithms for of *select*, *join* and *insert* operations. We first introduce the main data structures before presenting the algorithms.

5.1 Data Structures

In this section, we present the data structures used in the algorithms defined in the following sections. These data structures are designed to contain only the necessary and sufficient information so that we never need to have all the data in the main memory at once. The data structure representing a local hierarchy $\mathcal{H}$ assumes that the nodes of the hierarchy are indexed in a particular order and relies on three "attributes": 1) a mapping of the indices from the local context (a given slice) to the global one (the whole graph) noted $\mathcal{H}$`.map`, 2) the parent array denoted by $\mathcal{H}$`.par` encoding the parent relation between the tree nodes, and 3) an array $\mathcal{H}$`.weights` giving, for each non-leaf-node of

the tree, the weight of its corresponding building edge.

More precisely, given a binary partition hierarchy $\mathcal{H}$ with n regions, every integer between 0 and n-1 is associated with a unique region of $\mathcal{H}$. Moreover, this indexing of the regions of $\mathcal{H}$ follows a topological order such that:

1. Any leaf region is indexed before any non-leaf region;
2. Two leaf regions $\{x\}$ and $\{y\}$ are sorted with respect to an arbitrary order on the element V. In practice, we use the *raster scan order* of the pixels. Thus, $\{x\}$ has an index lower than $\{y\}$ if x is before y with respect to the raster scan order;
3. Two non-leaf regions are sorted according to their rank, *i.e.*, the order of their building edges for $\prec$.

This order can be seen as an extension of the order $\prec$ on E to the set $V \cup E$ that enables 1) to efficiently browse the nodes of a hierarchy according to their scale of appearance in the hierarchy and 2) to efficiently match regions of V with the leaves of the hierarchy. By abuse of notation, this extended order is also denoted by $\prec$ in the following.

To keep track of the global context, a link between the indices in the local tree and the global indices in the whole graph is stored in the form of an array `map` which associates: 1) to the index i of any leaf region R, the vertex x of the graph G such that $R = \{x\}$, *i.e.*, `map[i]=x`, and 2) to the index i of any non-leaf region R, its building edge, *i.e.*, `map[i]=`$\mu^{\prec}(R)$.

The parent relation of the hierarchy is stored thanks to an array `par` such that `par[i]=j` if the region of index j is the parent of the region of index i.

The binary partition hierarchy is built for a particular ordering $\prec$ of the edges of G. In practice, this ordering is induced by weights computed over the edges of G. To this end, we store an array `weights` of size $|\mathcal{R}^\star(\mathcal{H})|$, *i.e.*, the number of non-leaf regions, elements such that, for every region R in $\mathcal{R}^\star(\mathcal{H})$ of index i, `weights[i]` is the weight of the building edge $\mu^{\prec}(R)$ of region R. The edges can then be compared according to the following total order induced by the weights: we set $u \prec v$ if the weight of u is less than the one of v or if u and v have equal weights but u comes before v with respect to the raster scan order.

5.2 Select

In this part, we give Algorithm 4 to compute the result of the *select* operation. This operation consists in "selecting" all regions of a given hierarchy $\mathcal{H}$ intersecting a subset X of the space.

Select algorithm proceeds in 3 steps:

1. *Lines 5-9.* Mark any leaf-node of $\mathcal{H}$ that corresponds to an element of X, *i.e.*, any leaf-region $\{x\}$ with $x \in X$;
2. *Lines 11-15.* Traverse the hierarchy from leaves to root and mark any node that is a parent of a marked node; computing its new index in the output hierarchy with `sNodeIdx`;
3. *Lines 18-24.* Build the hierarchy $\mathcal{S}$ whose nodes are the marked nodes of $\mathcal{H}$.

Algorithm 4: SELECT

```
   Data: A hierarchy H, a subset X of V
 1 . Result: The hierarchy S = select (X, H)
 2 . Initialize an array sNodeIdx to -1 for
     every region of H
 3 i := 0                    // i iterates over X
 4 j := 0                    // j over the leaves of H
 5 while i < |X| and j < |H.leaves| do
 6     if X[i] = H.map[j] then
 7         sNodeIdx[j] := 0
 8         i := i + 1
 9     j := j + 1
10 count := 0
11 foreach n from 0 to |H|-2 do
12     if sNodeIdx[n] ≠ -1 then
13         sNodeIdx[n] := count
14         sNodeIdx[H.par[n]] := 0
15         count := count + 1
16 sNodeIdx[H.root] := count
17 n_S := 0
18 foreach n from 0 to |H|-1 do
19     if sNodeIdx[n] ≠ - then
20         S.par[n_S] := sNodeIdx[H.par[n]]
21         S.map[n_S] := H.map[n]
22         if n ∈ R*(H) then
23             S.weight[n_S − |X|] :=
               H.weight[n_S − |H.leaves|]
24     n_S := n_S + 1
25 return S
```

In Algorithm 4, we assume that X is sorted and that $X \subseteq gr(\mathcal{H})$, which is always the case in Algorithm 3. For each element $X[i]$ of X, we search for the index j of a leaf of $\mathcal{H}$ mapped to $X[i]$, *i.e.*, such that $\mathcal{H}.\texttt{map[j]} = X[i]$. To this end, it is necessary to make a traversal of the leaves of $\mathcal{H}$. As mentioned before, the leaves correspond to the first indices by construction. The first step can then be performed in linear time with respect to the number of leaf regions of $\mathcal{H}$. The second step consist in traversing the whole hierarchy from leaves to root in order to mark every region of $\mathcal{H}$ which belongs to select $(X, \mathcal{H})$ *i.e.*, regions parent of a marked one. The complexity of this step is therefore linear with respect to the number of regions of $\mathcal{H}$. Finally, the last step boils down to extracting the hierarchy select $(X, \mathcal{H})$ from the marked nodes. For this, a new hierarchy is created by traversing $\mathcal{H}$ again. As the traversal is done by increasing order of index, the properties relating to the weights of the building edges and order of appearance of regions are preserved. The complexity of this last step is linear with respect to the number of regions of $\mathcal{H}$. Thus, Algorithm 4 has a linear $O(n)$ complexity, where n is the number of regions of $\mathcal{H}$.

5.3 Join

Intuitively, this operation merges two hierarchies according to their common neighborhood, that is the set of edges linking their grounds. In [23], the authors proposed an algorithm that can be used to successively add edges of the common neighborhood at their correct rank. Essentially, this involves updating the hierarchy by traversing the branches associated with the endpoints of each added edge. The worst-case complexity is then linear with respect to the size of the hierarchy for adding a single edge. Thus, the overall complexity of such join procedure would be $O(kn)$ where n is the number of region of the considered hierarchies and k is the number of edges in the common neighborhood. In this section, we drop the multiplicative dependency in the size of the neighborhood F at the cost of introducing a sorting of F and we present an algorithm whose complexity is quasi-linear with respect to the size n of the hierarchies and linearithmic with respect to the number k of edges in F.

Algorithm 5: JOIN

Data: Two hierarchies $\mathcal{X}$ and $\mathcal{Y}$, the common neighborhood F of $gr(\mathcal{X})$ and $gr(\mathcal{Y})$.
Result: A collection $Q_D = \text{join}(\mathcal{X}, \mathcal{Y})$

```
1  foreach node n_i of X do Q_D.MakeSet(i)
2  foreach node n_i of Y do
3      Q_D.MakeSet(i + |X.leaves|)
4  aDescendent(X, 0)
5  aDescendent(Y, |X.leaves|)
6  F :=sort(F)
7  i_x := |X.leaves|; i_y := |Y.leaves|; i_f := 0
8  while i_x < |X| or i_y < |Y| or i_f < |F| do
9      if F[i_f] ≺ X.map[i_x]
         and F[i_f] ≺ Y.map[i_y] then
10         (x, y) := F[i_f]
11         m := F[i_f]
12         w := weight(F[i_f]); i_f += 1
13     else if X.map[i_x] ≺ Y.map[i_y] then
14         (x, y) := X.desc[i_x]
15         m := X.map[i_x]
16         w := X.weight[i_x]; i_x += 1
17     else
18         (x, y) := Y.desc[i_y]
19         m := Y.map[i_y]
20         w := Y.weight[i_y]; i_y += 1
21     c_x := Q_D.FindCanonical(x)
22     c_y := Q_D.FindCanonical(y)
23     if c_x ≠ c_y then
24         n := Q_D.Union(c_x, c_y);
25         Q_D.map[n] := m
26         Q_D.weight[n − (|X.leaves| +
             |Y.leaves|)] := w
```

A detailed presentation of the proposed algorithm is given in Algorithm 5 which calls auxiliary functions presented in Algorithm 6. Intuitively, in order to compute the join of two hierarchies $\mathcal{X}$ and $\mathcal{Y}$, Algorithm 5 consists in "emulating" PWK algorithm on the graph obtained from (i) the edges associated with the non-leaf nodes of $\mathcal{X}$ and of $\mathcal{Y}$ and (ii) the edges in the common neighborhood F of $\mathcal{X}$ and $\mathcal{Y}$. Therefore, all these edges are considered in increasing order with respect to $\prec$ and, for each edge, it is decided if this edge must be considered or not in the creation process of the join hierarchy. The decision is made based on

the potential creation of a cycle if this edge was added during the minimum spanning tree creation process.

A main observation can be made to highlight the difference between the situation encountered in the contexts of join algorithm and PWK algorithm: in the context of join, some edges, which are associated to the nodes of the hierarchies $\mathcal{X}$ and $\mathcal{Y}$, are made of vertices that do not belong to the underlying space (*i.e.*, the common neighborhood of the slices supporting the grounds of $\mathcal{X}$ and $\mathcal{Y}$).

To process them, we compute a pair attribute `desc` for each non-leaf nodes of $\mathcal{X}$ and $\mathcal{Y}$. Roughly speaking, this attribute assigns a source and target vertex to each of these edges. For a given node, the source is determined as a leaf of the subtree rooted in its first child (arbitrarily chosen), whose `map` necessarily refers to a node of the underlying space. On the other hand, the target is either another a leaf of the subtree rooted in its second child or, alternatively, assigned a value of -1 in cases where the node has only one child (*e.g.*, region of $\mathcal{X}$ of rank 4 in Figure 2)

When such edge is found, the standard algorithm can be shortcut leading to a modified version of the PWK auxiliary functions presented in Algorithm 6 at line 12. This test detects the edges for which a shortcut must occur based on the value of `desc`. Indeed, if the target of an edge is equal to -1, no cycle can be created. As a result, the node is added as the parent of the canonical element associated with the source of this edge. This attribute is pre-computed for every node of $\mathcal{X}$ and of $\mathcal{Y}$ by the auxiliary function `aDescendent` Overall, the following steps are performed in Algorithm 5:

- *Lines 1-3.* Initialize the Union-find data structures;
- *Lines 4-5.* Compute the attribute `desc` for both $\mathcal{X}$ and $\mathcal{Y}$;
- *Lines 6 to 20.* Browse the edges in increasing order. Observe that it implies sorting the edges in the common neighborhood F of $\mathcal{X}$ and $\mathcal{Y}$ in increasing order for $\prec$ (non-leaf nodes of $\mathcal{X}$ and $\mathcal{Y}$ are already sorted by construction);
- *Lines 23-26* Apply PWK steps, calling the modified version of the auxiliary functions.

The first step complexity is linear with respect to the number of elements of $gr(\mathcal{X}) \cup gr(\mathcal{Y})$. The second step uses the auxiliary function

Algorithm 6: AUXILIARY FUNCTIONS FOR *JOIN* ALGORITHM

```
   // The functions called hereafter on Q_T and Q_BT
      are those described in [4]
 1 Procedure Q_D.MakeSet(q)
 2     Q_D.Root[q] := q
 3     Q_BT.MakeSet(q)
 4     Q_T.MakeSet(q)
 5
 6 Function Q_D.FindCanonical(q)
 7     return Q_T.FindCanonical(q)
 8
 9 Function Q_D.Union(c_x, c_y)
10     t_u := Q_D.Root[c_x]
11     Q_BT.par[t_u] := Q_BT.size
12     if c_y = −1 then
         Q_D.Root[c_x] := Q_BT.size
13     else
14         t_v := Q_D.Root[c_y]
15         Q_BT.par[t_v] := Q_D.size
16         c := Q_T.Union(c_x, c_y)
17         Q_D.Root[c] := Q_BT.size
18     end
19     Q_BT.MakeSet(Q_BT.size)
20     return Q_D.size − 1
21
22 Function aDescendent(H, s)
23     foreach node n of H do
         H.desc[n]:= (−1, −1)
24     foreach leaf node n of H do
         H.desc[n].first:= n + s
25     foreach non-root non-leaf node n of
         H in increasing order for ≺ do
26         p := H.par[n]
27         if H.desc[p].first= −1 then
             H.desc[p].first:= H.desc[n].first
28         else
             H.desc[p].second:= H.desc[n].first
29     end
```

`aDescendent` to compute attributes `desc` for both $\mathcal{X}$ and $\mathcal{Y}$ consisting in a leaves to root traversal which gives a linear complexity with respect to the number of regions of each hierarchy. It should be noted that this last function takes a parameter *shift* which allows to index the leaves of the second hierarchy after those of the first. Third step requires to sort the edges of F with respect to $\prec$

Algorithm 7: INSERT

Data: Two hierarchies $\mathcal{X}$ and $\mathcal{Y}$ such that $\mathcal{X}$ insertable into $\mathcal{Y}$.
Result: $\mathcal{Z}$: the hierarchy insert$(\mathcal{X}, \mathcal{Y})$

$x := 0;\ y := 0;\ z := 0;$ // indices for the nodes/regions of $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$
Initialize an array InZ of $|\mathcal{Y}|$ Booleans to true (resp. to false) for every leaf (resp. non-leaf) region of $\mathcal{Y}$
while $x < |\mathcal{X}|$ ***or*** $y < |\mathcal{Y}|$ **do**
  if $x < |\mathcal{X}|$ ***and*** $y < |\mathcal{Y}|$ ***and*** $\mathcal{X}.map[x] = \mathcal{Y}.map[y]$ **then**
    // Duplicate region (x, y) found in $\mathcal{X}$ and $\mathcal{Y}$, keep (and renumber) it in $\mathcal{Z}$
    $\mathrm{C}_{\mathcal{X}\to\mathcal{Z}}[x] := z;\ \mathrm{C}_{\mathcal{Y}\to\mathcal{Z}}[y] := z;\ \mathrm{C}_{\mathcal{Z}\to\mathcal{X},\mathcal{Y}}[z] := (x, y)$
    $x \mathrel{+}= 1;\ y \mathrel{+}= 1;\ z \mathrel{+}= 1$
  else if $\mathcal{Y}.map[y] \prec \mathcal{X}.map[x]$ **then**
    if $InZ[y] = true$ **then** // Keep (and renumber) region y in $\mathcal{Z}$
      $InZ[\mathcal{Y}.\mathrm{par}[y]] := true$
      $\mathrm{C}_{\mathcal{Y}\to\mathcal{Z}}[y] := z;\ \mathrm{C}_{\mathcal{Z}\to\mathcal{X},\mathcal{Y}}[z] := (-1, y)$
      $y \mathrel{+}= 1;\ z \mathrel{+}= 1$
    else $y \mathrel{+}= 1$ // Discard region y from $\mathcal{Z}$
  else // Keep (and renumber) region x in $\mathcal{Z}$
    $\mathrm{C}_{\mathcal{X}\to\mathcal{Z}}[x] := z;\ \mathrm{C}_{\mathcal{Z}\to\mathcal{X},\mathcal{Y}}[z] := (x, -1)$
    $x \mathrel{+}= 1;\ z \mathrel{+}= 1$
$\mathcal{Z} :=$ initialize a tree structure with $n_{\mathcal{Z}} = z$ nodes
foreach z ***from*** 0 ***to*** $n_{\mathcal{Z}}$ **do**
  $(x, y) := \mathrm{C}_{\mathcal{Z}\to\mathcal{X},\mathcal{Y}}[z]$
  if $x \neq -1$ **then**
    $\mathcal{Z}.\mathrm{map}[z] := \mathcal{X}.\mathrm{map}[x]$
    if $[x] = \mathcal{X}.root$ **then** $\mathcal{Z}.\mathrm{par}[z] := z$
    else $\mathcal{Z}.\mathrm{par}[z] := \mathrm{C}_{\mathcal{X}\to\mathcal{Z}}[\mathcal{X}.\mathrm{par}[x]]$
    if $z \geq |\mathcal{X}.leaves|$ **then** $\mathcal{Z}.\mathrm{weight}[z - |\mathcal{X}.leaves|] := \mathcal{X}.\mathrm{weight}[x - |\mathcal{X}.leaves|]$
  else
    $\mathcal{Z}.\mathrm{map}[z] := \mathcal{Y}.\mathrm{map}[y]$
    if $y = \mathcal{Y}.root$ **then** $\mathcal{Z}.\mathrm{par}[z] := z$
    else $\mathcal{Z}.\mathrm{par}[z] := \mathrm{C}_{\mathcal{Y}\to\mathcal{Z}}[\mathcal{Y}.\mathrm{par}[y]]$
    if $z \geq |\mathcal{Y}.leaves|$ **then** $\mathcal{Z}.\mathrm{weight}[z - |\mathcal{Y}.leaves|] := \mathcal{Y}.\mathrm{weight}[x - |\mathcal{Y}.leaves|]$

before browsing the edges which implies a complexity of $O(k \times log(k) + |\mathcal{X}| + |\mathcal{Y}|)$ with k the number of edges in F. The fourth step is equivalent to PWK algorithm in terms of complexity. Then, its complexity is $O(m \times \alpha(n))$ where m is sum of the number of edges in F and the number of non-leaf nodes of $\mathcal{X}$ and $\mathcal{Y}$, where n is the number of leaf nodes in $\mathcal{X}$ and $\mathcal{Y}$ and where $\alpha()$ is the inverse Ackermann function which grows sub-logarithmically.

5.4 Insert

In this part, we present Algorithm 7. The operation *insert* aims to enrich a hierarchy $\mathcal{Y}$ by inserting the regions of another hierarchy $\mathcal{X}$ by computing a new hierarchy $\mathcal{Z} = \text{insert}(\mathcal{X}, \mathcal{Y})$ (called the insertion of $\mathcal{X}$ into $\mathcal{Y}$). Regions within $\mathcal{X}$ are inserted at the correct rank in the output hierarchy, and some regions of $\mathcal{Y}$ are discriminated and discarded from the output hierarchy.

As stated in Definition 5, $\mathcal{X}$ must be *insertable* into $\mathcal{Y}$; this assumption holds true at each call to *insert* in Algorithm 2 and Algorithm 3.

From a high-level point of view, insert algorithm proceeds in two main steps:

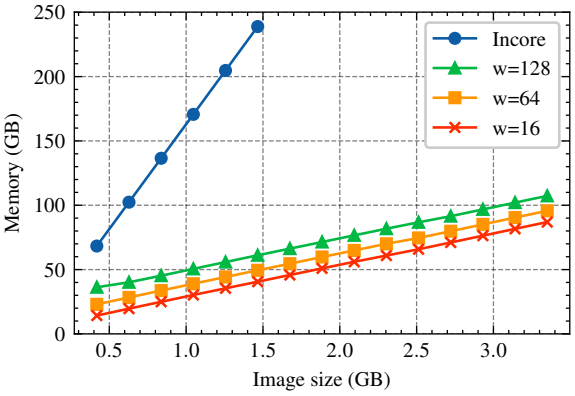
1. *Lines 2-14.* Identify and renumber the regions of $\mathcal{X}$ and $\mathcal{Y}$ that belong to $\mathcal{Z}$ and store the correspondences between the new number of the regions in $\mathcal{Z}$ and the indices of the initial regions in $\mathcal{X}$ and $\mathcal{Y}$. It can be observed that this step is necessary since a region of $\mathcal{Z}$ can be duplicated in both $\mathcal{X}$ and $\mathcal{Y}$ and that some regions of $\mathcal{Y}$ are discarded from $\mathcal{Z}$. In order to perform this step, the regions of $\mathcal{X}$ and $\mathcal{Y}$ are simultaneously browsed in increasing order for $\prec$. The correspondences between the regions of the hierarchies are stored in three arrays: $\mathrm{C}_{\mathcal{X}\to\mathcal{Z}}$, $\mathrm{C}_{\mathcal{Y}\to\mathcal{Z}}$, and $\mathrm{C}_{\mathcal{Z}\to\mathcal{X},\mathcal{Y}}$;
2. *Lines 16-27.* Build the parenthood relation (`par`) of the hierarchy $\mathcal{Z}$ using the parenthood relation of the hierarchies $\mathcal{X}$ and $\mathcal{Y}$ and the correspondences between the regions of the hierarchies. At the same time, we also build the attributes `map` and `weight` associated with $\mathcal{Z}$.

During the first step, each region of the two hierarchies $\mathcal{X}$ and $\mathcal{Y}$ is considered once and processed with a limited number of constant-time instructions. Thus, the overall time complexity of Lines 2-14 is linear with respect to the number of nodes of $\mathcal{X}$ and $\mathcal{Y}$. The worst-case complexity of the second step is also linear with respect to the number of nodes of $\mathcal{X}$ and $\mathcal{Y}$ since $\mathcal{Z}$ contains at most all regions of each of hierarchy. Thus, the overall complexity of Algorithm 7 is $O(|\mathcal{X}| + |\mathcal{Y}|)$.

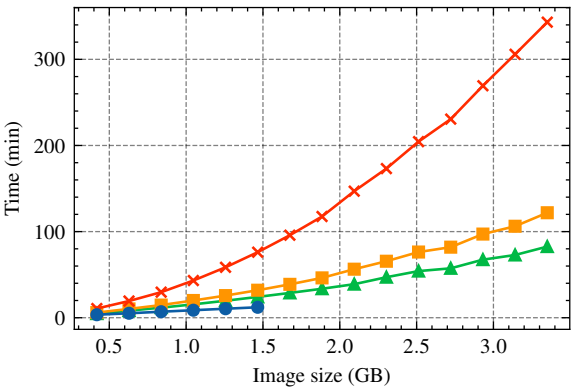
6 Experiments

In this section, we assess the performances of our framework over 3D 8-bit images. We conduct a comparative analysis of various computation patterns and partitioning strategies. The experiments are executed on a workstation equipped with an Intel Xeon processor running at 3.90GHz and 256GB RAM. Processed images are stored in HDF5 format [38] to facilitate tiled reading while intermediate and output data structures are managed through Zarr [39] using the Blosc compressor.

The experiments are conducted on real images, specifically a FIB-SEM volume of the CA1 hippocampus region of the brain with dimensions $2048 \times 1536 \times 1065$ (approximately 3.3 gigavoxels) [40]. It should be noted that it is not possible to run the classic algorithm for this size of volume on the workstation on which we carried out the experiments. At least 8 slices are needed to be able to compute the distribution.



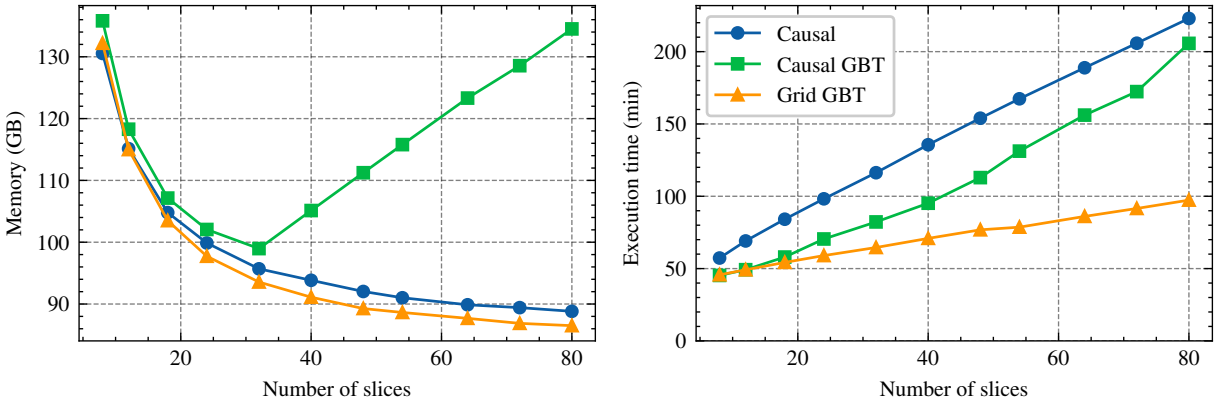
(a) Peak Memory Consumption



(b) Execution Time

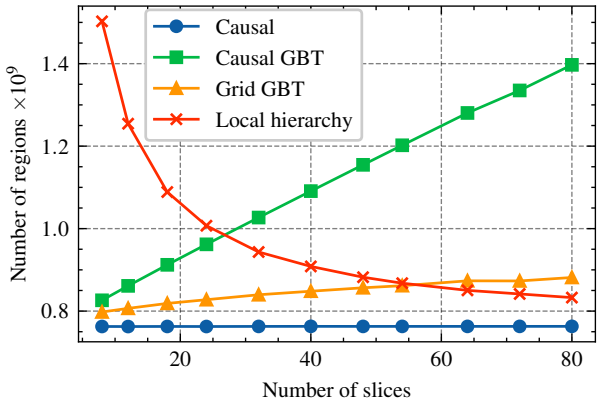
Fig. 8: Comparative analysis of peak memory consumption and execution time between the In-Core algorithm and the out-of-core causal given as a function of the image size. The variable w correspond to the slice width.

The first experiment consists of selecting several subsets of the test image of increasing size in order to observe memory consumption trends relative to the size of the input image. In this way, we can compare the in-core algorithm and the calculation of the distribution with the causal algorithm presented in Algorithm 3 varying size and number of tiles. Indeed, in Figure 8 the first

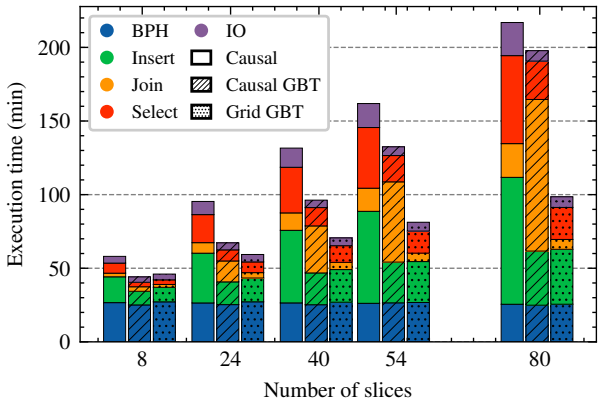


(a) Peak memory consumption

(b) Execution time



(c) Region count



(d) Core operations timings

Fig. 9: Comparative analysis across different computation patterns on FIB-SEM volume as a function of the number of slices.

data point on the green curve corresponds to a causal partition of 2 tiles of $128 \times 1536 \times 1065$ while for the orange curve this corresponds to 4 tiles with a width of 64 voxels, and for the red curve 16 tiles with a width of 16 voxels. Then, for each subsequent data point, the number of slices is incremented by 1, 2, and 8, respectively. Tendencies depicts the evident disparity in peak memory utilization between the out-of-core and in-core algorithms. Note that beyond 1.5GB, the in-core algorithm cannot produce results.

While both algorithms demonstrate a linear increase in memory consumption proportionate to image size, the out-of-core algorithm's slope is 6.7 times lower as that of the conventional in-core approach. Consequently, considering the current workstation, the upper limit of image size amenable to processing ascends from 1.5 GB to 10 GB with w=16. Moreover, we observe that the peak memory consumption decreases as we increase the number of slices.

In terms of execution times, it appears that, as expected, the OOC calculation time is longer than the in-core approach. However, the order of magnitude remains similar. We can also see that processing time increases with the number of tiles, highlighting a trade-off between execution time and memory consumption.

In the preceding experiment, we exclusively utilized only one of the presented calculation patterns. To compare them, we computed the distribution associated with the complete FIB-SEM volume while incrementally increasing the number of tiles. In the Figure 9, `Causal` refers to the high-level computation detailed Section 4.4, `Causal GBT` represents the global border tree

approach detailed in Section 4.3 applied to a causal image partitioning, and `Grid GBT` denotes the same approach but within the context of grid partitioning.

At first glance, the peak memory consumption in Figure 9a decreases with the increasing number of slices for all three methodologies. However, `Causal GBT` approach suffers from a sudden increase in memory consumption becoming no longer competitive. This abrupt escalation in memory consumption is due to the fast increase in the GBT size, correlated with the number of voxel at the interface between two tiles. In Figure 9c, we can see that the number of regions of `Causal GBT` quickly exceeds the average number of local hierarchy regions. While the number of regions in a local hierarchy decreases as a function of the number of slices, the number of regions in the global border tree increases linearly.

In contrast, the grid partitioning method does not exhibit such phenomenon in the first graph because the slope of the `Grid GBT` size is much less pronounced compared to `Causal`. As expected, the size of the largest merged tree in for `Causal` remains nearly constant regardless the number of tiles. While an explosion in memory consumption may occur for an excessive number of tiles, in the `Grid GBT`, it is not the case for the Causal one, which boasts superior scalability.

However, in Figure 9b, we can see that the execution time of GBT-based approaches remains systematically lower than the causal with an average speedup of 2 for `Grid GBT`. Therefore, this algorithm is a good method for reducing computation time as long as the number of tiles remains reasonable, which seems to be beneficial from an application point of view in order to strike a happy medium between memory consumption and computation time.

This disparity in computation time between the two GBT-based and `Causal` approaches can be easily explained by considering the size of the data structures and the number of calls to each operation. In Figure 9d we present a comparison of the the execution time of each core operation involved in the distribution computation. The term "BPH" refers to Playing with Kruskal algorithm [4], and IO encompasses both reading and writing operations on disk.

Firstly, when contrasting `Causal GBT` with `Grid GBT`, the only notable difference lies in *join*. The steep slope observed in the `Causal GBT` approach is a consequence of the rapid growth in the size of the GBT, making the merging process with a local border tree increasingly resource intensive.

Secondly, `Grid GBT` outperforms `Causal` in terms of execution time for each operations, except for BPH. It is easy to see that, given the methodology, the number of calls to *insert*, *select* and IO is halved. Regarding *join* and *select*, the performance improvement is also attributed to the fact that in causal partitioning, the number of voxels at the interface grows linearly with the number of tiles, while in grid partitioning, it is proportional to the cubic root of the number of tiles as illustrated in Figure 10.

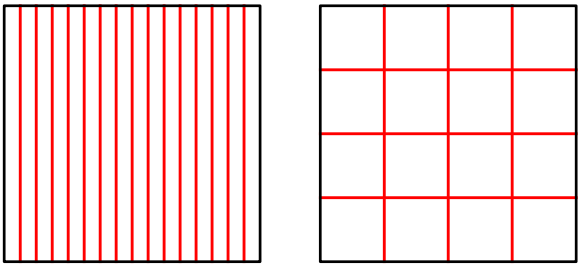


Fig. 10: Causal and grid partition into 16 tiles. It is necessary to draw 16 frontiers (in red) for the causal partition and only 9 for the grid partition to reach the same number of slices.

7 Conclusion

In this article, we introduced two high-level calculi for computing the distribution of the binary partition hierarchy (BPH), which consists of collections of subtrees of the BPH. The first calculus is applicable to any partition of the space, while the second is tailored specifically for causal partitions. Both calculi rely on three fundamental algebraic operations on hierarchies—*select*, *join*, and *insert*—which exhibit linear and linearithmic complexities.

We also outlined the properties necessary to ensure that the second calculus satisfies the out-of-core constraints. Our experiments compared these workflows, revealing that although the first

calculus lacks certain guarantees, it may still be suitable for applications with a moderate number of slices. Looking ahead, we plan to develop efficient algorithms for processing the distribution of a BPH, with the goal of enhancing the out-of-core toolkit by integrating hierarchical analysis tools for (marked) watershed and hierarchical watershed segmentation.

Acknowledgements

This work was supported by the French ANR grant ANR-20-CE23-0019.

Appendix A Proof of Property 7

Before proving Property 7, let us first introduce some lemmas.

Lemma 12. *Let A and B be two disjoint subsets of V and let A' and B' be two subsets of A and B, respectively, then we have:*

$$\begin{aligned}&\mathit{select}\left(A' \cup B', \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B\right)\\ &= \mathit{select}\left(A', \mathcal{B}^{\prec}_A\right) \sqcup \mathit{select}\left(B', \mathcal{B}^{\prec}_B\right)\end{aligned}$$

Lemma 13. *Let A and B be two disjoint subsets of V and Z be a set such as $Z \cap A \neq \emptyset$ and $Z \cap B \neq \emptyset$. Then we have:*
$\mathit{select}\,(Z, \mathit{join}\,(\mathcal{B}^{\prec}_A,\ \mathcal{B}^{\prec}_B)) = \mathit{join}\,(\mathit{select}\,(Z \cap A, \mathcal{B}^{\prec}_A),\ \mathit{select}\,(Z \cap B, \mathcal{B}^{\prec}_B))$

Proof. (of Lemma 13) Let $\mathcal{H} = \text{select}\,(Z, \text{join}\,(\mathcal{B}^{\prec}_A,\ \mathcal{B}^{\prec}_B))$. We are going to prove that $\mathcal{H} = \text{join}\,(\text{select}\,(Z \cap A, \mathcal{B}^{\prec}_A),\ \text{select}\,(Z \cap B, \mathcal{B}^{\prec}_B))$. We have

$$\begin{aligned}\mathcal{H} &= \text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B \boxplus \gamma(A, B))\\ &\quad \text{by definition of join}\\ &= \text{select}(Z, \text{insert}(\text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B \boxplus \gamma(A, B)),\\ &\quad \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B))\\ &\quad \text{by Lemma 13 of [26]}\\ &= \text{insert}(\text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B \boxplus \gamma(A, B)),\\ &\quad \text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B))\\ &\quad \text{by Lemma 12 of [26]}\\ &= \text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B) \boxplus \gamma(A, B)\\ &\quad \sqcup \text{select}(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B)\end{aligned}$$

From Property 7.3 of [26], we have

$$\begin{aligned}\mathcal{H} = \text{select}\left(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B\right) \sqcup \text{select}\left(Z, \mathcal{B}^{\prec}_A \sqcup \mathcal{B}^{\prec}_B\right)\\ \boxplus \gamma(A, B)\end{aligned}$$

Since $Z \cap A \subseteq A$ and $Z \cap B \subseteq B$, by Lemma 12 we deduce that:

$$\begin{aligned}\mathcal{H} &= \text{select}\left(Z \cap A, \mathcal{B}^{\prec}_A\right) \sqcup \text{select}\left(Z \cap B, \mathcal{B}^{\prec}_B\right)\\ &\quad \boxplus \gamma(A, B)\\ &= \text{join}\left(\text{select}\left(Z \cap A, \mathcal{B}^{\prec}_A\right),\ \text{select}\left(Z \cap B, \mathcal{B}^{\prec}_B\right)\right)\\ &\quad \text{by definition of join}\end{aligned}$$

□

At initialization step of Algorithm 1, we have $\mathcal{M}_0 = \text{select}\left(\gamma^{\bullet}_{V \setminus S_0}(S_0), \mathcal{B}^{\prec}_{S_0}\right)$. Then at the first iteration we have

$$\begin{aligned}\mathcal{M}_1 &= \text{join}\left(\text{select}\left(\gamma^{\bullet}_{V \setminus S_1}(S_1), \mathcal{B}^{\prec}_{S_1}\right)\right)\\ &\quad \left(\text{select}\left(\gamma^{\bullet}_{V \setminus S_0}(S_0), \mathcal{B}^{\prec}_{S_0}\right)\right)\\ &= \text{select}(\gamma^{\bullet}_{V \setminus S_0}(S_0) \cup \gamma^{\bullet}_{V \setminus S_1}(S_1),\\ &\quad \text{join}\,(\mathcal{B}^{\prec}_{S_0},\ \mathcal{B}^{\prec}_{S_1}))\\ &\quad \text{by Lemma 13}\\ &= \text{select}\left(\gamma^{\bullet}_{V \setminus S_0}(S_0) \cup \gamma^{\bullet}_{V \setminus S_1}(S_1), \mathcal{B}^{\prec}_{S_0 \cup S_1}\right)\\ &\quad \text{by Property 9 of [26]}\end{aligned}$$

By recurrence, we have

$$\begin{aligned}\mathcal{M}_i = \text{select}\left(F, \mathcal{B}^{\prec}_{\cup\{S_j \mid j \in \{0,\ldots i\}\}}\right)\\ \text{with} \quad F = \bigcup_{j=0}^{i} \left\{\gamma^{\bullet}_{V \setminus S_j}(S_j)\right\}\end{aligned}$$

Then, if we consider a partition into $k+1$ regions, when $i = k$ we have

$$\mathcal{M}_k = \text{select}\left(F, \mathcal{B}_V^{\prec}\right)$$
$$\text{with} \quad F = \bigcup_{j=0}^{k} \left\{\gamma^{\bullet}_{V\setminus S_j}(S_j)\right\}$$

This latter rewriting fits Definition 6. □

Appendix B Proof of Lemma 8

Proof. (of statement 1) At each step of the first loop of Algorithm 3, we have $\mathcal{M}^{\uparrow} = \text{join}\left(\text{select}\left(\gamma_i^{\bullet}(S_{i-1}), B_{i-1}^{\uparrow}\right), \text{select}\left(\gamma_{i-1}^{\bullet}(S_i), B_{S_i}^{\prec}\right)\right)$. Theorem 17 of [26] states that $B_{i-1}^{\uparrow} = \text{select}\left(S_i, \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\dots i\}\}}\right)$. Then, since $\delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)) \cap \gamma_i^{\bullet}(S_{i-1}) \subseteq \gamma_i^{\bullet}(S_{i-1})$ and $\delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)) \cap \gamma_{i-1}^{\bullet}(S_i) \subseteq \gamma_{i-1}^{\bullet}(S_i)$, by Lemma 13 we can deduce that $\mathcal{M}^{\uparrow} = \text{select}\left(\delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)), \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\dots i\}\}}\right)$ □

Proof. (of statement 2) Let A, B and C be three pairwise disjoint subsets of V such that $\gamma(A, C) = \emptyset$. Let $\mathcal{M}^{\uparrow} = \text{join}\left(\text{select}\left(\gamma_B^{\bullet}(A), \mathcal{B}_A^{\prec}\right), \text{select}\left(\gamma_B^{\bullet}(A), \mathcal{B}_A^{\prec}\right)\right)$ and $\mathcal{M}^{\downarrow} = \text{insert}(\text{select}\left(\gamma_B^{\bullet}(A), \mathcal{B}^{\prec}_{A\cup B\cup C}\right), \mathcal{M}^{\uparrow})$. Then we have

$$\begin{aligned}
\mathcal{M}^{\downarrow} &= \text{join}(\text{select}\left(\gamma_B^{\bullet}(A), \mathcal{B}_A^{\prec}\right), \\
&\qquad \text{select}\left(\gamma_A^{\bullet}(B), \mathcal{B}^{\prec}_{B\cup C}\right)) \\
&\qquad \text{by Lemma 15 of [26]} \\
&= \text{select}\left(\gamma_B^{\bullet}(A) \cup \gamma_A^{\bullet}(B), \text{join}\left(\mathcal{B}_A^{\prec}, \ \mathcal{B}^{\prec}_{B\cup C}\right)\right) \\
&\qquad \text{by Lemma 13} \\
&= \text{select}\left(\gamma_B^{\bullet}(A) \cup \gamma_A^{\bullet}(B), \mathcal{B}^{\prec}_{A\cup B\cup C}\right) \\
&\qquad \text{by definition of join}
\end{aligned}$$

Since $\gamma_B^{\bullet}(A) \cup \gamma_A^{\bullet}(B) = \delta^{\bullet}(\gamma(A, B))$, we have $\mathcal{M}^{\downarrow} = \text{select}\left(\delta^{\bullet}(\gamma(A, B)), \mathcal{B}^{\prec}_{A\cup B\cup C}\right)$. □

Appendix C Proof of Property 9

Proof. In order to establish Property 9, we first introduce the following lemma:

Lemma 14. *Let $\mathcal{H}$ be a BPH on V. Let $A \subseteq V$ and $B \subseteq V$ such that $A \subseteq B$. The we have $\textit{select}(A, \mathcal{H}) \sqsubseteq \textit{select}(B, \mathcal{H})$.*

Then, we proceed in two steps:

(1) $\mathcal{M}_i^{\uparrow} \sqsubseteq \mathcal{M}_i$; and
(2) $\mathcal{M}_i^{\downarrow} \sqsubseteq \mathcal{M}$.

(1) As stated in Section 4.3, $\mathcal{M}_i$ built on $\mathbf{P}$ is defined as follows

$$\mathcal{M}_i = \text{select}\left(F, \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\dots i\}\}}\right) \tag{C1}$$

where

$$F = \bigcup_{j=0}^{i} \left\{\gamma^{\bullet}_{V\setminus S_j}(S_j)\right\} \tag{C2}$$

From the definition of $\delta^{\bullet}$, γ and $\gamma^{\bullet}$, it can be seen that

$$F = \bigcup_{j=0}^{i} \left\{\gamma^{\bullet}_{V\setminus S_j}(S_j)\right\} \tag{C3}$$
$$= \bigcup_{j=1}^{i} \{\delta^{\bullet}(\gamma(S_{j\text{-}1}, S_j))\} \tag{C4}$$

As stated in Property 8, we have

$$\mathcal{M}_i^{\uparrow} = \text{select}\left(C, \mathcal{B}^{\prec}_{\cup\{S_j \mid j\in\{0,\dots i\}\}}\right) \tag{C5}$$

where

$$C = \delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i)) \tag{C6}$$

We can deduce that for any $i \in \{1, \dots, k\}$ we have $C \subseteq F$. Therefore, given Lemma 14 we have $\mathcal{M}_i^{\uparrow} \sqsubseteq \mathcal{M}_i$.

(2) For this second statement, we can express $\mathcal{M}$ and $\mathcal{M}_i$ with C from Equation (C6) and F from Equation (C4).

$$\mathcal{M} = \text{select}\left(F, \mathcal{B}_V^{\prec}\right) \tag{C7}$$
$$\mathcal{M}_i^{\downarrow} = \text{select}\left(C, \mathcal{B}_V^{\prec}\right) \tag{C8}$$

We have $C \subseteq F$, and with Lemma 14 we have $\mathcal{M}^{\downarrow} \sqsubseteq \mathcal{M}$.

□

Appendix D Proof of Property 11

Proof. Given $n_i = |\text{select}\,(S_{i\text{-}1}, \mathcal{H})| + |\text{select}\,(S_i, \mathcal{H})|$, we will prove statements of Property 11 in the following order:

(1) $|\mathcal{M}_i^{\downarrow}| \leq n_i$; and
(2) $|\mathcal{M}_i^{\uparrow}| \leq n_i$.

(1) From Property 8, we have $\mathcal{M}_i^{\downarrow} = \text{select}\,(F, \mathcal{H})$ where $F = \delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i))$. We can see that $F \subseteq (S_{i\text{-}1} \cup S_i)$ which gives us Equation (D9). Each region R of $\text{select}\,(S_{i\text{-}1} \cup S_i, \mathcal{H})$ either intersects an element of $S_{i\text{-}1}$ or S_i or both. As a result, R belongs either to $\text{select}\,(S_{i\text{-}1}, \mathcal{H})$ or $\text{select}\,(S_i, \mathcal{H})$ or both. As a result, we have Equation (D10).

$$|\mathcal{M}_i^{\downarrow}| \leq |\text{select}\,(S_{i\text{-}1} \cup S_i, \mathcal{H})| \tag{D9}$$
$$\leq |\text{select}\,(S_{i\text{-}1}, \mathcal{H})| + |\text{select}\,(S_i, \mathcal{H})| \tag{D10}$$

(2) Given $C = \delta^{\bullet}(\gamma(S_{i\text{-}1}, S_i))$, we can write the two structures as follows:

$$\mathcal{M}_i^{\downarrow} = \text{select}\,\left(C, \mathcal{B}_V^{\prec}\right) \tag{D11}$$
$$\mathcal{M}^{\uparrow} = \text{select}\,\left(C, \mathcal{B}^{\prec}_{\cup\{S_j \mid j \in \{0,\dots k\}\}}\right) \tag{D12}$$

Considering Lemma 15, we have $|\mathcal{M}_i^{\uparrow}| \leq |\mathcal{M}_i^{\downarrow}|$. Thus we have $|\mathcal{M}_i^{\uparrow}| \leq |\mathcal{M}_i^{\downarrow}| \leq n_i$

Lemma 15. *Let $(S_0, \dots, S_k)$ be a causal partition of V, $A \subseteq V$ and u be a pair of distinct elements of V. For any $i \in \{0, \dots, k\}$, the following inequalities holds true:*

$$|\mathit{select}\,(A, \mathcal{H})| \leq |\mathit{select}\,(A, \mathcal{H} \boxplus \{u\})| \tag{D13}$$
$$|\mathit{select}\left(X, \mathcal{B}^{\prec}_{\cup\{S_j \mid j \in \{0,\dots i\}\}}\right)| \leq |\mathit{select}\,(X, \mathcal{B}_V^{\prec})| \tag{D14}$$

□

References

[1] Salembier, P., Garrido, L.: Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval. TIP **9**(4), 561–576 (2000) https://doi.org/10.1109/83.841934

[2] Randrianasoa, J.F., Kurtz, C., Desjardin, Passat, N.: Binary partition tree construction from multiple features for image segmentation. Pattern Recognition **84**, 237–250 (2018) https://doi.org/10.1016/j.patcog.2018.07.003

[3] Cousty, J., Najman, L., Perret, B.: Constructive links between some morphological hierarchies on edge-weighted graphs. In: ISMM, pp. 86–97 (2013). https://doi.org/10.1007/978-3-642-38294-9_8

[4] Najman, L., Cousty, J., Perret, B.: Playing with Kruskal: algorithms for morphological trees in edge-weighted graphs. In: ISMM, pp. 135–146 (2013). https://doi.org/10.1007/978-3-642-38294-9_12

[5] Meyer, F., Maragos, P.: Morphological scale-space representation with levelings. In: International Conference on Scale-Space Theories in Computer Vision, pp. 187–198 (1999). https://doi.org/10.1007/3-540-48236-9_17

[6] Meyer, F.: The dynamics of minima and contours. In: ISMM, pp. 329–336 (1996). https://doi.org/10.1007/978-1-4613-0469-2_38

[7] Baum, D., Weaver, J.C., Zlotnikov, I., Knötel, D., Tomholt, L., Dean, M.N.: High-Throughput Segmentation of Tiled Biological Structures using Random-Walk Distance Transforms. Integrative and Comparative Biology **59**(6), 1700–1712 (2019) https://doi.org/10.1093/icb/icz117

[8] Soares, A.P., Baum, D., Hesse, B., Kupsch, A., Müller, B.R., Zaslansky, P.: Scattering and phase-contrast x-ray methods reveal damage to glass fibers in endodontic posts following dental bur trimming. Dental Materials **37**(2), 201–211 (2021) https://doi.org/10.1016/j.dental.2020.10.018

[9] Paiva, P.V.V., Cogima, C.K., Dezen-Kempter, E., Carvalho, M.A.G.: Historical building point cloud segmentation combining hierarchical watershed transform and curvature analysis. Pattern Recognition Letters **135**, 114–121 (2020) https://doi.org/10.1016/j.patrec.2020.04.010

[10] Le Moigne, B., Rault, C., Guiotte, F., Thomas, D.J., Dewez, T.: RIDIM: Unveiling Rock Instabilities through Hierarchical Segmentation of 3D Point Clouds. 14th International Symposium on Landslides. Poster (2024)

[11] K C, S., Aryal, J., Ryu, D.: Automated delineation of the agricultural fields using multi-task deep learning and optical satellite imagery. In: IGARSS 2023 - 2023 IEEE International Geoscience and Remote Sensing Symposium, pp. 2795–2798 (2023). https://doi.org/10.1109/IGARSS52108.2023.10282141

[12] Maia, D.S., Pham, M.-T., Lefèvre, S.: Watershed-based attribute profiles with semantic prior knowledge for remote sensing image analysis. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing **15**, 2574–2591 (2022) https://doi.org/10.1109/JSTARS.2022.3153110

[13] Vitter, J.S.: External memory algorithms and data structures: dealing with massive data. ACM Comput. Surv. **33**(2), 209–271 (2001) https://doi.org/10.1145/384192.384193

[14] Wang, S., Zhang, M., Yang, K., Chen, K., Ma, S., Jiang, J., Wu, Y.: Noswalker: A decoupled architecture for out-of-core random walk processing. In: ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 466–482. ACM, New York, NY, USA (2023). https://doi.org/10.1145/3582016.3582025

[15] Arge, L., Danner, A., Haverkort, H., Zeh, N.: I/O-Efficient Hierarchical Watershed Decomposition of Grid Terrain Models, pp. 825–844. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/3-540-35589-8_51

[16] Danner, A., Mølhave, T., Yi, K., Agarwal, P.K., Arge, L., Mitasova, H.: Terrastream: From elevation data to watershed hierarchies. In: ACM Symposium on Advances in Geographic Information Systems, pp. 212–219 (2007). https://doi.org/10.1145/1341012.1341049

[17] Lindstrom, P., Pascucci, V.: Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. IEEE Transactions on Visualization and Computer Graphics **8**(3), 239–254 (2002) https://doi.org/10.1109/TVCG.2002.1021577

[18] Toledo, S.: A survey of out-of-core algorithms in numerical linear algebra, pp. 161–179. American Mathematical Society, USA (1999). https://doi.org/10.1090/dimacs/050

[19] Gazagnes, S., Wilkinson, M.H.F.: Distributed connected component filtering and analysis in 2d and 3d tera-scale data sets. IEEE Transactions on Image Processing **30**, 3664–3675 (2021) https://doi.org/10.1109/TIP.2021.3064223

[20] Götz, M., Cavallaro, G., Geraud, T., Book, M., Riedel, M.: Parallel computation of component trees on distributed memory machines. TPDS **29**(11), 2582–2598 (2018) https://doi.org/10.1109/TPDS.2018.2829724

[21] Kazemier, J.J., Ouzounis, G.K., Wilkinson, M.H.: Connected morphological attribute filters on distributed memory parallel machines. In: ISMM, pp. 357–368 (2017). https://doi.org/10.1007/978-3-319-57240-6_29

[22] Gigli, L., Velasco-Forero, S., Marcotegui, B.: On minimum spanning tree streaming for hierarchical segmentation. PRL **138**, 155–162 (2020) https://doi.org/10.1016/j.patrec.2020.07.006

[23] Havel, J., Merciol, F., Lefèvre, S.: Efficient tree construction for multiscale image representation and processing. JRTIP **16**(4), 1129–1146 (2019) https://doi.org/10.1007/

s11554-016-0604-0

[24] Carlinet, E., Blin, N., Lemaitre, F., Lacassagne, L., Geraud, T.: Max-tree computation on GPUs. IEEE TPDS (2022) https://doi.org/10.1109/TPDS.2022.3158488

[25] Perrin, R., Leborgne, A., Passat, N., Naegel, B., Wemmert, C.: Multi-scale component-tree: A hierarchical representation for sparse objects. In: DGMM, pp. 312–324. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-57793-2_24

[26] Cousty, J., Perret, B., Phelippeau, H., Carneiro, S., Kamlay, P., Buzer, L.: An algebraic framework for out-of-core hierarchical segmentation algorithms. In: DGMM, pp. 378–390 (2021). https://doi.org/10.1007/978-3-030-76657-3_27

[27] Lefèvre, J., Cousty, J., Perret, B., Phelippeau, H.: Join, select, and insert: Efficient out-of-core algorithms for hierarchical segmentation trees. In: Discrete Geometry and Mathematical Morphology, pp. 274–286. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-19897-7_22

[28] Perret, B., Chierchia, G., Cousty, J., Guimarães, S.J.F., Kenmochi, Y., Najman, L.: Higra: Hierarchical graph analysis. SoftwareX **10**, 100335 (2019) https://doi.org/10.1016/j.softx.2019.100335

[29] Najman, L., Cousty, J.: A graph-based mathematical morphology reader. Pattern Recognition Letters **47**, 3–17 (2014) https://doi.org/10.1016/j.patrec.2014.05.007 . Advances in Mathematical Morphology

[30] Passat, N., Kurtz, C., Vacavant, A.: Editorial — virtual special issue: "hierarchical representations: New results and challenges for image analysis". Pattern Recognition Letters **138**, 201–203 (2020) https://doi.org/10.1016/j.patrec.2020.07.019

[31] Bosilj, P., Kijak, E., Lefèvre, S.: Partition and inclusion hierarchies of images: A comprehensive survey. Journal of Imaging **4**(2) (2018) https://doi.org/10.3390/jimaging4020033

[32] Salembier, P., Oliveras, A., Garrido, L.: Antiextensive connected operators for image and sequence processing. IEEE Transactions on Image Processing **7**(4), 555–570 (1998) https://doi.org/10.1109/83.663500

[33] Matas, P., Dokládalová, E., Akil, M., Grandpierre, T., Najman, L., Poupa, M., Georgiev, V.: Parallel algorithm for concurrent computation of connected component tree. In: Advanced Concepts for Intelligent Vision Systems, pp. 230–241. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88458-3_21

[34] Wilkinson, M.H.F., Hui Gao, Hesselink, W.H., Jonker, J.-E., Meijster, A.: Concurrent computation of attribute filters on shared memory parallel machines **30**(10), 1800–1813 https://doi.org/10.1109/TPAMI.2007.70836 . Accessed 2023-03-17

[35] Ouzounis, G.K., Soille, P.: Pattern spectra from partition pyramids and hierarchies. In: Soille, P., Pesaresi, M., Ouzounis, G.K. (eds.) Mathematical Morphology and Its Applications to Image and Signal Processing, pp. 108–119. Springer, ??? (2011). https://doi.org/10.1007/978-3-642-21569-8_10

[36] Havel, J., Merciol, F., Lefèvre, S.: Efficient schemes for computing α-tree representations. In: Mathematical Morphology and Its Applications to Signal and Image Processing, pp. 111–122. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38294-9_10

[37] Ronse, C.: Partial partitions, partial connections and connective segmentation. JMIV **32**(2), 97–125 (2008) https://doi.org/10.1007/s10851-008-0090-5

[38] Group, T.H.: Hierarchical Data Format, Version 5. https://github.com/HDFGroup/hdf5

[39] Miles, A., Kirkham, J., Durant, M., Bourbeau, J., Onalan, T., Hamman, J., Patel, Z., shikharsg, Rocklin, M., dussin, Schut, V., Andrade, E.S., Abernathey, R., Noyes, C., sbalmer, bot, Tran, T., Saalfeld, S., Swaney, J., Moore,

J., Jevnik, J., Kelleher, J., Funke, J., Sakkis, G., Barnes, C., Banihirwe, A.: Zarr-developers/zarr-python: V2.4.0. https://doi.org/10.5281/zenodo.3773450

[40] Lucchi, A., Smith, K., Achanta, R., Knott, G., Fua, P.: Supervoxel-based segmentation of mitochondria in EM image stacks with learned shape features. IEEE Transactions on Medical Imaging **31**(2), 474–486 (2012) https://doi.org/10.1109/TMI.2011.2171705