Abstract—This paper introduces an efficient algorithm for persistence diagram computation, given an input piecewise linear scalar field \( f \) defined on a \( d \)-dimensional simplicial complex \( K \), with \( d \leq 3 \). Our work revisits the seminal algorithm “PairSimplices” (Edelsbrunner et al. 2002), (Zomorodian, 2010) with discrete Morse theory (DMT) (Forman, 1998), (Robins et al. 2011), which greatly reduces the number of input simplices to consider. Further, we also extend to DMT and accelerate the stratification strategy described in “PairSimplices” (Edelsbrunner et al. 2002), (Zomorodian, 2010) for the fast computation of the \( 0^{th} \) and \( (d-1)^{th} \) diagrams, noted \( D_0(f) \) and \( D_{d-1}(f) \). Minima-saddle persistence pairs \( (D_0(f)) \) and saddle-maximum persistence pairs \( (D_{d-1}(f)) \) are efficiently computed by processing, with a Union-Find, the unstable sets of 1-saddles and the stable sets of \( (d-1) \)-saddles. This fast pre-computation for the dimensions 0 and \( (d-1) \) enables an aggressive specialization of (Bauer et al. 2014) to the 3D case, which results in a drastic reduction of the number of input simplices for the computation of \( D_1(f) \), the intermediate layer of the sandwich. Finally, we document several performance improvements via shared-memory parallelism. We provide an open-source implementation of our algorithm for reproducibility purposes. Extensive experiments indicate that our algorithm improves by two orders of magnitude the time performance of the seminal “PairSimplices” algorithm it extends. Moreover, it also improves memory footprint and time performance over a selection of 14 competing approaches, with a substantial gain over the fastest available approaches, while producing a strictly identical output.

Index Terms—discrete Morse theory, persistence diagrams, scalar data, Topological data analysis.

I. INTRODUCTION

Scalar data is central to many fields of science and engineering. It can be the result of an (i) acquisition process (examples include CT-scans produced in medical imaging or one-dimensional time-series produced by punctual sensors) or it can be the result of a (ii) numerical computation (examples include simulations in computational fluid dynamics, material sciences, etc.). In both cases, the data is typically provided as a low-dimensional scalar field (1D, 2D, or 3D) defined on the vertices of either (i) a regular grid (e.g., pixel or voxel images) or (ii) a mesh (e.g., polyhedral surfaces and volumes, AMR grids, etc.). An established strategy to generically process either cases of data provenance is to subdivide each cell of the input domain into simplices [49], [54], hence converting the input data into a generic representation that facilitates subsequent processing, namely a piecewise linear scalar field defined over a simplicial complex (i.e., poly-lines in 1D, triangulated surfaces in 2D and tetrahedral meshes in 3D). However, such scalar fields are provided in the applications with an ever-increasing size and geometrical complexity, which significantly challenges their interpretation by human users. This motivates the design of advanced data analysis tools, to support the interactive exploration and analysis of the features of interest present in large datasets. This is precisely the purpose of Topological Data Analysis (TDA) [30], which provides a toolbox of techniques for the generic, robust, and efficient extraction of structural features in data.

Topological methods have been investigated by the visualization community for more than twenty years [51], with applications to a variety of domains, including combustion [15], [42], [61] fluid dynamics [20], [57] material sciences [46], [62], chemistry [7], [37], [75], or astrophysics [84], [89] to name a few. Several topological data representations studied in TDA (such as the persistence diagram [30], the contour tree [16], [39], [91], the Reeb graph [8], [40], [76], [77], [78] or the Morse-Smale complex [14], [23], [43], [45], [47], [83]) have been specialized and used successfully in visualization, in particular for the explicit extraction and visual representation of structural patterns hidden in the data. An important aspect of TDA is its ability to provide multi-scale hierarchies of the above topological data representations, which consequently enables multi-scale visualization, exploration and analysis. In that setting, Topological Persistence [30] is an established importance measure which enables to distinguish the most salient topological structures present in the data from those corresponding to noise. In typical analysis pipelines (as shown in Fig. 1), this importance measure drives the simplification of the above topological representations, resulting in interactive, multi-scale data explorations. In practice, topological persistence can be obtained by computing Persistence
Diagrams [30]. Several algorithms have been proposed for their computation (see Section I-A) and many software packages are publicly available. However, most of them generically target data defined in arbitrary dimension (with a specific focus towards high dimensional point clouds) and provide only limited specialization for low-dimensional scalar data.

In this work, we introduce a novel algorithm for the fast computation of persistence diagrams for scalar data defined on 1, 2 or 3-dimensional domains. In contrast to previous work, our approach specifically takes advantage of the low dimensionality of typical scalar data by revisiting a stratification strategy [4, 30, 31], that we call *sandwiching*, from the perspective of discrete Morse theory [34] (see Section III for an overview). Our algorithm has several advantages over existing approaches. Our extensive experiments (Section IX) demonstrate a substantial gain over existing algorithms, both in memory footprint and time performance, while delivering a strictly identical output. Moreover, it is output sensitive and most of its internal procedures can be accelerated with shared-memory parallelism (Section VII).

For reproducibility purposes, we provide a C++ implementation of our approach. We also contribute a benchmark package, which exploits three-dimensional data from a public repository [58] and compares our approach to a variety of publicly available implementations. We believe such a benchmark has the potential to become a reference experiment for future work on the topic. Finally, we present an application (Section VIII) to the fast and robust extraction of generators for surfaces, volume data or high-dimensional point clouds.

A. Related Work

This section describes the literature related to our work. First, we provide a quick overview of the usage of persistent homology in data visualization. Second, we briefly review the related computational methods.

**Persistent Homology in Visualization.** Persistent Homology has originally been introduced independently by several research groups [2, 31, 35, 79]. In many applications involving data analysis, topological persistence quickly established itself as an appealing importance measure that helps distinguish salient topological structures in the data.

In data visualization, except a few approaches dealing with graph layouts [90] and dimensionality reduction [29, 74], Persistent Homology has been mostly used in previous work in scientific visualization, typically dealing with the interactive visual analysis of scalar data (coming from acquisitions or simulations). In that context, topological persistence is typically used as a measure of importance driving the simplification of the input data itself [63, 95], or the multi-scale hierarchical representation of topological abstractions [51], such as contour trees [16, 39, 91], Reeb graphs [8, 40, 76, 77, 78] or Morse-Smale complexes [14, 23, 43, 45, 47, 83]. For instance, in the “Topology ToolKit” (TTK) [9, 94] (an open-source library for topological data analysis and visualization), data is typically pre-simplified interactively, by removing low persistence features [63, 95], yielding a multi-scale hierarchy for the subsequent topological data representations (Fig. 1). Similar analysis pipelines have been documented in a number of applications, including combustion [15, 42, 61] fluid dynamics [20, 57] material sciences [46, 62], chemistry [7, 37, 75], or astrophysics [84, 89]. Topological persistence has also been used as an importance measure in several other scalar data analysis tasks, such as data segmentation [10, 17], isosurface extraction [97], data compression [88] or transfer function design for volume rendering [102]. The persistence diagram (Section II-D) is a popular topological data representation, which concisely and robustly captures the number and salience of the features of interest present in the data. As such, it is an effective visual descriptor of the population of features in data, for ensemble summarization [33, 59, 98] or feature tracking [64, 86, 87].

**Algorithms for Computing Persistent Homology.** In general, the standard approach to the computation of persistent homology involves the reduction of the boundary matrix [30] (which describes the facet/co-facet relations between the simplices of the input domain). This approach is now the core procedure of many software packages. This includes for instance *PHAT* [6] and *Dipheo* [5] (which feature additional accelerations [19, 24], along with specific data structures for cubical cell complexes [101]), *Gudhi* [66] (which also features specific accelerations [11, 13, 25, 27] and data structures [12], in particular for cubical cell complexes [101]) and others [69, 92]. Certain
packages have a special focus towards the persistent homology of Rips filtrations of high-dimensional point clouds, such as Ripser [3] (adapted to cubical complexes [56]) or Eirene [52], [53]. They have been integrated in several data analysis libraries [81], [93]. Among the above techniques, several documented accelerations share some conceptual similarities with our approach. Specifically, Bauer et al. [4] accelerate the global computation by pre-computing persistence pairs (Section II-D) on localized chunks of the data. Similarly, Bauer [3] introduces the notion of apparent pairs, which are persistence pairs involving a simplex and one of its co-facets, and which can be efficiently detected in a pre-process. Our approach uses discrete Morse theory [34] to serve a similar purpose. As further detailed in Section IV, our pre-computation focuses on simplices involved in zero-persistence pairs (specifically, within each vertex lower star), but then pair them greedily according to the expansion-based discrete gradient [80] of the data. Bauer et al. [4] introduce a stratification strategy (“clearing and compression”) when computing the persistence diagram in dimension \( p \), which discards \((p + 1)\)-simplices (respectively \( p \)-simplices) which already create (respectively destroy) persistence pairs (typically computed within the above chunks). Our stratification strategy can be interpreted as an aggressive specialization of the above strategy for three-dimensional data. Specifically, when computing for the dimension 1, not only does our algorithm discard from the computation the simplices involved in zero-persistence pairs, but it also discards all the simplices involved in zero and two-dimensional persistence pairs. Some methods support parallel computations [5], [6], [70], [73]. All the above approaches are included in our benchmark (Section IX-C).

For low dimensional data, such as typical 3D scalar fields, specific stratification strategies can be considered. In their seminal paper introducing the algorithm “PairSimplices” [31], Edelsbrunner et al. observe that the persistence diagram can be efficiently computed for the dimension 0 with a Union-Find data structure [22]. They also observe that, by symmetry, a Union-Find can also be used for the dimension 2 (without specifying, however, how these computations interact with the diagram of dimension 1). This strategy has been the default computation method in TTK [9], [94] since its initial release, where the sub-level set components of \( f \) and \(-f\) are efficiently tracked with a parallel merge tree algorithm [38], [39]. Recently, Vidal et al. presented progressive [99] and approximate [100] variants, based on a multiresolution representation of the input. In this paper, we revisit this stratification strategy (originally introduced by Edelsbrunner et al. [31]), but from the perspective of discrete Morse theory [34], and we further accelerate it by restricting the sub-level set connectivity tracking to the unstable (and stable) sets of 1 and \((d - 1)\) saddles. This acceleration can be interpreted as an adaptation to discrete Morse theory of earlier work on monotone paths for merge tree construction [18], [21], [65], [85], [99].

Similarly to our work, previous approaches have investigated Morse theory [67], [71], specifically its discrete version [34], to accelerate the computation of persistence diagrams of scalar data. To apply discrete Morse theory (DMT), one first needs to compute a discrete gradient (Section II-F) from the input scalar data. For this, several algorithms have been proposed [43], [44], [45], [80], [82], [83], [94]. Shivashankar et al. [82], [83] introduced the “AssignGradient” algorithm, which constructs discrete vectors equivalent to the apparent pairs of Bauer [3]. However, in practice, even for smooth datasets, this algorithm generates a number of critical simplices several orders of magnitude larger than the number of piecewise linear (PL) critical points [94], resulting in a large majority of spurious zero-persistence pairs in the persistence diagrams (i.e., pairs not captured by the lower star filtration, Section II-B). To address this, Tierny et al. [94] performed an explicit reversal of the \(v\)-paths (Section II-F) involving spurious critical simplices (i.e., not located in the star of any PL critical point). Robins et al. [26], [80] introduced an algorithm for computing a discrete gradient field, including an implicit, localized gradient reversal (called “ProcessLowerStars”). This algorithm provides the nice property that it generates no spurious critical simplex: all the resulting critical simplices are guaranteed to be located within the star of a PL critical point (we directly exploit this property in our work, Section IV). Hence, these simplices exactly coincide with the topological changes of the lower star filtration of the data (Section II-B). Then, all the topological events occurring in the filtration of the scalar data can be equivalently encoded with a filtration of its discrete Morse complex. As this complex is usually smaller in practice than the input data, this pre-process reduction procedure accelerates subsequent, traditional algorithms for persistent homology [103]. Extensions of this idea have been investigated [41], [55], [68], [72], e.g. for the support of high dimensional data. In contrast, our work specifically takes advantage of the low dimensionality of the data to expedite the process (to avoid the computation of the full Morse complex), with a stratification strategy adapted from “PairSimplices” to discrete Morse theory, which we further accelerate by restricting a Union-Find processing to the unstable (and stable) sets of 1 and \((d - 1)\) saddles (Section V).

B. Contributions

This paper makes the following new contributions:

1) A fast algorithm for the computation of persistence diagrams for 1D, 2D or 3D scalar data:

- Our approach revisits the algorithm “PairSimplices” [31], [103] with discrete Morse theory [34], [80]. Simplices involved in zero-persistence pairs are efficiently skipped, which drastically reduces the number of input simplices to consider (Section IV).
- The persistence diagrams for the dimensions 0 and \((d - 1)\) are efficiently computed by restricting a Union-Find [22] processing to the unstable (and stable) sets of 1 and \((d - 1)\) saddles (Section V). This enables an aggressive specialization of [4] to the 3D case, which accelerates the computation in dimension 1 by further reducing the number of input simplices.

Our algorithm provides practical gains over reference methods. It is output sensitive and its sub-routines can be accelerated with shared-memory parallelism.
2) An open-source implementation: For reproduction purposes, we provide a C++ implementation of our approach, which is officially integrated in the source tree of TTK [9], [94] (Github commit: bb3089f).

3) A reproducible benchmark: We provide a Python benchmark package (https://github.com/pierre-guilhou/pdiags_bench), which uses three-dimensional data from a public repository [58] and compares the running times, memory footprints and output diagrams of a variety of publicly available implementations for persistence diagram computation. This reproducible benchmark may be used as a reference experiment for future developments on the topic.

II. PRELIMINARIES

This section presents the theoretical background of our work. It contains definitions adapted from the Topology ToolKit [9], [94]. We refer the reader to textbooks [30], [103] for comprehensive introductions to computational topology.

A. Input Data

The input data is provided as a piecewise linear (PL) scalar field \( f: \mathcal{K} \to \mathbb{R} \) defined on \( d \)-dimensional simplicial complex \( \mathcal{K} \), with \( d \leq 3 \). As discussed in the introduction, this input representation generically and homogeneously supports all types of typical scalar data, in 1D, 2D or 3D, coming from either acquisitions or numerical simulations. When the data is given on arbitrary cell complexes, cells are subdivided into simplices. In particular, regular grids are triangulated according to the Freudenthal triangulation [49], [54] (yielding a 6-vertex neighborhood in 2D and a 14-vertex neighborhood in 3D). Note that this triangulation is performed implicitly (i.e., no memory overhead), by emulating the simplicial structure upon traversal queries [94]. The input scalar field \( f \) is typically provided on the vertices of \( \mathcal{K} \) and interpolated on the simplices of higher dimension. \( f \) is also assumed to be injective on the vertices of \( \mathcal{K} \), which is easily achieved in practice with a symbolic perturbation inspired from Simulation of Simplicity [32].

B. Lexicographic Filtration

Given the input function \( f \), a global order between the simplices of \( \mathcal{K} \) can be introduced by considering the so-called lexicographic comparison, as detailed below.

Given a \( d \)-simplex \( \sigma \in \mathcal{K} \), let us consider the sequence \( \{f(v_0(\sigma)), f(v_1(\sigma)), \ldots, f(v_d(\sigma))\} \) of its vertex data values, sorted in decreasing order, where \( f(v_i(\sigma)) \) denotes the \( i \)-th largest value among its vertices, i.e., \( f(v_0(\sigma)) > f(v_1(\sigma)) > \cdots > f(v_d(\sigma)) \).

Then, an order can be established between any two simplices \( \sigma_i \) and \( \sigma_j \) by comparing the above sorted sequences. In particular, \( \sigma_i \) will be considered smaller than \( \sigma_j \) if \( f(v_0(\sigma_i)) < f(v_0(\sigma_j)) \). On the contrary, if \( f(v_0(\sigma_i)) > f(v_0(\sigma_j)) \), \( \sigma_i \) will be considered greater than \( \sigma_j \). Otherwise, if \( f(v_0(\sigma_i)) = f(v_0(\sigma_j)) \), a tiebreak needs to be performed and the order will be decided by iteratively considering, similarly, the following vertices in the sequence (i.e., \( v_k(\sigma_i) \) and \( v_k(\sigma_j) \), with \( k \in \{1, \ldots, d\} \)) until the conditions \( f(v_k(\sigma_i)) < f(v_k(\sigma_j)) \) (i.e., \( \sigma_i \) is smaller than \( \sigma_j \)) or \( f(v_k(\sigma_i)) > f(v_k(\sigma_j)) \) (i.e., \( \sigma_i \) is greater than \( \sigma_j \)) are satisfied. In the case where the dimensions \( d_i \) and \( d_j \) of \( \sigma_i \) and \( \sigma_j \) are such that \( d_i < d_j \) and that \( f(v_k(\sigma_i)) = f(v_k(\sigma_j)) \), \( \forall k \in \{0, \ldots, d_i\} \) (i.e., \( \sigma_i \) is a face of \( \sigma_j \)), then \( \sigma_i \) is considered smaller than \( \sigma_j \).

Since \( f \) is injective on the vertices of \( \mathcal{K} \) (Section II-A), this lexicographic comparison guarantees a strict total order on the set of simplices of \( \mathcal{K} \), such that all the faces of a simplex \( \sigma \) are by construction smaller than \( \sigma \).

Let \( \mathcal{K}_n \) be the union of the first \( i \) simplices of \( \mathcal{K} \), given the above comparison. Then, the global lexicographic order induces a nested sequence of simplicial complexes \( \emptyset = \mathcal{K}_0 \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}_n = \mathcal{K} \) (where \( n \) is the number of simplices of \( \mathcal{K} \)), which we call the lexicographic filtration of \( \mathcal{K} \). Intuitively, it can be seen as a time-varying process, where the simplices of \( \mathcal{K} \) are added one by one, given the lexicographic comparison of the vertex data values.

A central idea in Topological Data Analysis consists in encoding the evolution of the topological structures of \( \mathcal{K}_i \) (for typical scalar data: its connected components, handles and voids, see Section II-C) along the filtration, as \( i \) increases from 0 to \( n \). In particular, as shown in Fig. 2, connected components progressively merge, handles get closed and voids get filled. This evolution is captured by the persistence diagram introduced later (Section II-D).

We now discuss an alternate filtration, often considered in previous work [30], [80] and we describe its relation (used in Section IV) to the lexicographic filtration considered here. Let \( \mathcal{S}(v) \) be the star of a vertex \( v \), i.e., the set of all its co-faces \( \sigma: \mathcal{S}(v) = \{ \sigma \in \mathcal{K} \mid v \subset \sigma \} \). Let \( \mathcal{L}(v) \) be the lower star of \( v \). It is the subset of the simplices of the star of \( v \), for which \( v \) is the vertex with highest \( f \) value: \( \mathcal{L}(v) = \{ \sigma \in \mathcal{S}(v) \mid \forall u \in \sigma, f(u) \leq f(v) \} \). Since \( f \) is assumed to be injective on the vertices of \( \mathcal{K} \), it follows that each simplex \( \sigma \in \mathcal{K} \) belongs to a unique lower star. Let \( \mathcal{K}_i' \) be the union of the first \( i \) lower stars, i.e., the union of the lower stars of \( j \)-th lowest vertex \( f \). Then, the nested sequence of simplicial complexes \( \emptyset = \mathcal{K}_0' \subset \mathcal{K}_1' \subset \cdots \subset \mathcal{K}_n' = \mathcal{K} \) (where \( n_v \) is the number of vertices in \( \mathcal{K} \)) is called the lower star filtration of \( f \) [30]. \( \mathcal{K}_i' \) is homotopy equivalent to the sub-level sets of \( f(v_i) \) [30] and the topological changes occurring in \( \mathcal{K}_i' \) during the lower star filtration thus precisely occur at the PL critical points [1] of \( f \).

Given the above definition, it follows that each sub-complex \( \mathcal{K}_j' \) of the lower star filtration is equal to the sub-complex \( \mathcal{K}_{i-1} \) of the lexicographic filtration, where \( \sigma_i \) is the vertex immediately after \( v_i \) in the global vertex order. In other words, the lower star filtration introduces simplices by chunks of lower stars (Fig. 9), while the lexicographic filtration introduces them one by one, yet in a compatible order. Then, it follows that each PL critical point includes in its lower star a simplex whose introduction via the lexicographic filtration changes the topology of \( \mathcal{K}_i \).

C. Homology Groups

The topology of a simplicial complex can be described with its homology groups, briefly summarized here from [30].
We call a \( p \)-chain \( c \) a formal sum (with modulo 2 coefficients) of \( p \)-simplices \( \sigma_i \) of \( K \):  

\[
\text{c} = \sum \alpha_i \sigma_i, \quad \text{with } \alpha_i \in \{0, 1\}.
\]

Two \( p \)-chains \( \alpha = \sum \alpha_i \sigma_i \) and \( \beta = \sum \beta_i \sigma_i \) can be summed together componentwise to form a new \( p \)-chain \( \alpha + \beta = \sum (\alpha_i + \beta_i) \sigma_i \), where \( \alpha_i + \beta_i \in \{0, 1\} \). Intuitively, a \( p \)-chain can be interpreted as a selection of \( p \)-simplices, modeled with a bit mask, where a \( p \)-simplex is present in the selection (i.e., with its coefficient valued at 1) only if it has been added an odd number of times. Then, the set of all possible \( p \)-chains of \( K \) (along with their modulo 2 addition) forms the group of chains, noted \( C_p(K) \).

The boundary of a \( p \)-simplex \( \sigma_i \), noted \( \partial \sigma_i \), is given by the sum of its faces of dimension \( p - 1 \). Then, the boundary of a \( p \)-chain \( c \), noted \( \partial c \), is the sum of the boundaries of the simplices of \( c \):  

\[
\partial c = \sum \alpha_i \partial \sigma_i.
\]

Note that \( \partial \sigma_i \) is itself a \( (p - 1) \)-chain, i.e., \( \partial : C_p(K) \to C_{p-1}(K) \), and that the boundary operator commutes with addition, i.e., \( \partial(c + c') = \partial c + \partial c' \).

A \( p \)-cycle \( c \) is a \( p \)-chain such that \( \partial c = 0 \) and the group of all possible \( p \)-cycles is noted \( Z_p(K) \). A \( p \)-boundary is a \( p \)-chain \( c \in C_p(K) \) which is the boundary of a \( (p+1) \)-chain \( c' \in C_{p+1}(K) \):  

\[
c = \partial c'.
\]

The group of \( p \)-boundaries is noted \( B_p(K) \). The fundamental lemma of homology states that \( \partial \partial c = 0 \) for every \( p \)-chain \( c \), for any \( p \) [30]. This implies that \( p \)-boundaries are necessarily \( p \)-cycles (\( B_p \subseteq Z_p \)), but not the other way around: all \( p \)-cycles are not necessarily \( p \)-boundaries. Such cycles are specifically captured with the notion of homology group, which is the quotient group given by:  

\[
H_p(K) = Z_p(K)/B_p(K).
\]

Specifically, two \( p \)-cycles \( a \) and \( b \) of \( Z_p \) are called homologous (noted \( a \sim b \)), if \( b = a + \partial c \) where \( \partial c \) is a \( p \)-boundary (\( \partial c \in B_p \)). Intuitively, this means that two cycles \( a \) and \( b \) are homologous if one can be transformed into the other, by the addition of the boundary of a \( (p + 1) \)-chain \( c \) (Fig. 3), as further discussed in Section II-E.

The set of all cycles which are homologous defines a homology class (from which anyone can be chosen as a representative). The order of \( H_p(K) \) is given by its cardinality, i.e., the number of homology classes. Given the modulo-2 addition between representatives, the rank of \( H_p(K) \) is given by the maximum number of linearly independent classes (called generators) and it is called the \( p \)-th Betti number of \( K \), noted \( \beta_p(K) \). Intuitively, the \( p \)-th Betti number gives the number of \( p \)-dimensional holes in \( K \), which cannot be filled with a \( (p + 1) \)-chain of \( K \). In practice, given a 3-dimensional simplicial complex \( K \) embedded in \( \mathbb{R}^3 \), \( \beta_0(K) \) corresponds to its number of connected components, \( \beta_1(K) \) is its number of handles and \( \beta_2(K) \) is its number of voids.

**D. Persistence Diagrams**

Persistent diagrams are concise topological data representations which track the evolution of the homology groups during a filtration. In the remainder, we focus on the lexicographic filtration introduced in Section II-B. Since \( K_i \subseteq K_j \) for any \( 0 \leq i \leq j \leq n \), it follows that there exists a homomorphism [30] between the homology groups \( H_p(K_i) \) and \( H_p(K_j) \), noted \( f_p^{i,j} : H_p(K_i) \to H_p(K_j) \). This homomorphism \( f_p^{i,j} \) keeps track of the relations between the homology classes along the filtration, from \( K_i \) to \( K_j \) (Fig. 4).
Fig. 4. Tracking homology classes along the filtration with homomorphisms (dark green arrows, illustration adapted from [30]). The class $\gamma$ is born in $K_i$; it is not the image through the homomorphism $f^{i-1}_p$ of any pre-existing class (i.e., it does not belong to $\mathcal{H}_p(K_{i-1},)$, blue set). $\gamma$ dies in $K_j$ as it merges with a pre-existing class, i.e., the image through the homomorphism $f^{j-1}_p$ of a class already existing at step $i = 1$ and still persistent at step $j$ (blue set). In Fig. 2, the dark green cycle $c_f$ has a similar trajectory: it is born at step (f) and at step (g), it merges with the pre-existing cycle $c_e$ (i.e., $c_f$ becomes homologous to $c_e$).

Formally, for any $0 \leq i \leq j \leq n$, the $p$-th persistent homology group, noted $\mathcal{H}_p(K_{i,j})$, is the image of the homomorphism $f^{i-1}_p$, noted $\mathcal{H}_p(K_{i,j}) = f^{i-1}_p(\mathcal{H}_p(K_i))$.

Specifically, we say that a homology class $\gamma$ is born at $K_i$ if $\gamma \in \mathcal{H}_p(K_i)$ and $\gamma \notin \mathcal{H}_p(K_{i-1},)$ (see Fig. 4); $\gamma$ is present in $\mathcal{H}_p(K_i)$ but it is not included in the image by $f^{i-1}_p$ of the homology groups of the previous complex in the filtration, $K_{i-1}$. In other words, $\gamma$ is present in $\mathcal{H}_p(K_i)$ but it is not associated to any pre-existing class of $\mathcal{H}_p(K_{i-1})$ by $f^{i-1}_p$.

Symmetrically, we say that a homology class $\gamma$ born at $K_i$ dies at $K_j$ if (i) $f^{j-1}_p(\gamma) \notin \mathcal{H}_p(K_{i,j+1})$ and (ii) it merged (through $f^{j-1}_p$) with another, pre-existing class $\gamma'$, itself created before $i$. This destruction of a class upon its merge with another older class is often called the Elder rule [30]. Note that the birth of a $p$-dimensional homology class $\gamma$ occurs on a $p$-simplex $\sigma_1$ of $K_i$, while its death occurs on a $(p+1)$-simplex $\sigma_j$ of $K_j$. The pair $\sigma_i, \sigma_j$ is called a persistence pair.

The persistence of a homology class $\gamma$ which is born in $K_i$ and which dies in $K_j$ is given by the difference in the corresponding scalar values $P(\gamma) = f(v_i) - f(v_j)$, where $f(v_i)$ and $f(v_j)$ are respectively the maximum vertex data values of the simplices $\sigma_j$ and $\sigma_i$. Note that a homology class $\gamma$ which was born in $K_i$ and whose image by $f^{p,n}$ is still included in $\mathcal{H}_p(K_{i,n})$ is said to have infinite persistence (i.e., it is still present in the final complex $K_n = K$).

The persistence diagram of dimension $p$, noted $\mathcal{D}_p(f)$, is a concise encoding of the $p$-dimensional persistent homology groups. In particular, it embeds each persistent generator $\gamma$ in the 2D birth/death plane at position $(f(v_i), f(v_j))$ and its persistence can be therefore directly read from its height to the diagonal. This has the practical implication that generators with large persistence (typically corresponding to salient features in the data) are located far away from the diagonal, whereas generators with small persistence (typically corresponding to noise) are located in the vicinity of the diagonal, as illustrated in Fig. 5.

Algorithm 1: Reference “PairSimplices” [31], [103].

```
Input: Lexicographic filtration of $K$ by $f$.
Output: Persistence diagrams $\mathcal{D}_0(f)$, $\mathcal{D}_1(f)$ and $\mathcal{D}_2(f)$.
1: for $j \in [1, n]$ do
2: // Process the $(d_i + 1)$-simplex $\sigma_j$
3: $\sigma_j \leftarrow \emptyset$
4: Chain($\sigma_j$) $\leftarrow \sigma_j$
5: // Homologous propagation of $\partial \sigma_j$
6: while $\partial(\text{Chain}(\sigma_j)) \not= 0$ do
7: $\tau \leftarrow \text{max}(\partial(\text{Chain}(\sigma_j)))$
8: if $\text{Pair}(\tau) = $ \emptyset then
9: // $\tau$ created a $(d_i)$-cycle
10: break
11: else
12: // Expand chain (with homologous boundary)
13: Chain($\sigma_j$) $\leftarrow$ Chain($\sigma_j$) + Chain($\tau$)
14: end if
15: end while
16: if $\partial(\text{Chain}(\sigma_j)) \not= 0$ then
17: // A non-trivial cycle homologous to $\partial \sigma_j$ exists (l.
18: $\tau \leftarrow \text{max}(\partial(\text{Chain}(\sigma_j)))$
19: $\text{Pair}(\sigma_j) \leftarrow \tau$
20: $\text{Pair}(\tau) \leftarrow \sigma_j$
21: $\mathcal{D}_{d_i}(f) \leftarrow \mathcal{D}_{d_i}(f) \cup (\tau, \sigma_j)$
22: end if
23: end for
```

E. The Algorithm “PairSimplices”

Edelsbrunner et al. [30], [31], [103] describe an iterative algorithm called “PairSimplices” for the computation of persistence diagrams. We sketch its main steps here as our approach builds on top of it. This description is adapted from Zomorodian’s textbook [103]. This algorithm (Algorithm 1) observes, for each step $i$ of the input filtration, the effect of the insertion of a $d_i$-simplex $\sigma_i$ on the set of $(d_i - 1)$-cycles homologous to its boundary $\partial \sigma_i$. In particular, if $\partial \sigma_i$ was not already trivial (i.e., homologous to an empty cycle) in $K_{i-1}$, then the insertion of $\sigma_i$...
in \( K \) will now make \( \partial \sigma_1 \) trivial (\( \partial \sigma_1 \sim 0 \)). By transitivity, all the cycles \( c \) homologous to \( \partial \sigma_1 \) (its homology class) now become trivial as well, hence completing a persistence pair in \( D_{d_i-1}(f) \), that is, filling a \( d_{i-1} \)-dimensional hole of \( K_{i-1} \).

Thus, for each step \( i \) of the filtration, the algorithm reconstructs \( (d_i - 1) \)-cycles in \( K \), which are homologous to \( \partial \sigma_i \). This is achieved by a process that we call homologous propagation (Fig. 6), which iteratively expands a chain \( \text{Chain}(\sigma_i) \), whose boundary \( \partial(\text{Chain}(\sigma_i)) \) is homologous to \( \partial \sigma_i \), by construction (Algorithm 1, lines 6 to 15). This propagation is achieved by considering \( (d_i - 1) \)-simplices in decreasing filtration order, i.e., by selecting at each iteration the highest simplex (Algorithm 1, line 7), and by stopping at the first unpaired \( (d_i - 1) \)-simplex \( \tau \) (Algorithm 1, line 10), responsible for the creation of the latest (i.e., youngest) homologous \( (d_i - 1) \)-cycle in \( K_{i-1} \), hence effectively enforcing the Elder rule (Section II-D). Then the persistence pair \((\tau, \sigma_i)\) is created in \( D_{d_i-1}(f) \). If \( \partial \sigma_1 \) was trivial initially in \( K_{i-1} \) when starting the propagation, \( \text{Chain}(\sigma_i) \) is extended until its boundary becomes empty and no pair will be created in \( D_{d_i-1}(f) \). Note that the above algorithm can be viewed as a geometric interpretation of boundary matrix reduction, which is at the core of most modern approaches (Section I-A). In particular, selecting the highest boundary simplex \( \tau \) (Algorithm 1, line 7) is equivalent to the identification, during matrix reduction, of the lowest non-zero entry for the column \( j \) of the boundary matrix. Considering the boundary of the chain sum (line 13) is equivalent to summing columns [30] (Section VII-A).

**F. Discrete Morse Theory (DMT)**

We now conclude this section of preliminaries with notions of discrete Morse theory [34], or DMT for short (which we restrict here to simplicial complexes), as it is instrumental in our approach to accelerate the algorithm “PairSimplices”.

We call a discrete vector a pair formed by a simplex \( \sigma_i \in K \) (of dimension \( i \)) and one of its co-faces \( \sigma_{i+1} \) (i.e., one of its co-faces of dimension \( i + 1 \)), noted \( \{\sigma_i < \sigma_{i+1}\} \). \( \sigma_{i+1} \) is usually referred to as the head of the vector, while \( \sigma_i \) is its tail. Examples of discrete vectors include a pair between a vertex and one of its incident edges, or a pair between an edge and a triangle containing it (see Fig. 7). A discrete vector field on \( K \) is defined as a collection \( V \) of pairs \( \{\sigma_i < \sigma_{i+1}\} \) such that each simplex of \( K \) is involved in at most one pair. A simplex \( \sigma_i \) which is involved in no discrete vector of \( V \) is called a critical simplex.

A discrete integral line, or \( v \)-path, is a sequence of discrete vectors \( \{\sigma_i^0 < \sigma_i^{j+1}\}, \ldots, \{\sigma_i^k < \sigma_i^{j+1}\} \) such that (i) \( \sigma_i^j \neq \sigma_i^{j+1} \) (i.e., the tails of two consecutive vectors are distinct) and (ii) \( \sigma_i^{j+1} < \sigma_i^{j+1} \) (the tail of a vector in the sequence is a face of the head of the previous vector in the sequence) for any \( 0 < j < k \). We say that a discrete integral line terminates at a critical simplex \( \sigma_i \) if \( \sigma_i \) is a facet of the head of its last vector \( \{\sigma_i^k < \sigma_i^{k+1}\} \) (i.e., \( \sigma_i < \sigma_i^{k+1} \)). Symmetrically, we say that a discrete integral line starts at a critical simplex \( \sigma_{i+1} \) if \( \sigma_{i+1} \) is a co-facet of the tail of its first vector \( \sigma_i^0 \) (i.e., \( \sigma_i^0 < \sigma_i^{i+1} \)). By analogy with the smooth setting, this notion of discrete integral lines therefore starts and terminates at critical points. The collection of all the discrete integral lines terminating in a given critical simplex \( \sigma_i \) is called the discrete stable set of \( \sigma_i \) and it is noted \( K^\circ(\sigma_i) \). Symmetrically, the collection of all the discrete integral lines starting at a given critical simplex \( \sigma_i \) is called the discrete unstable set of \( \sigma_i \) (Fig. 7) and it is noted \( K^\bullet(\sigma_i) \).

A discrete vector field such that all of its possible discrete integral lines are acyclic is called a discrete gradient field [34], noted \( G \). Then, the critical simplices of \( G \) are discrete analogs to the critical points from the smooth setting [67, 71]. Their dimension \( i \) corresponds to the smooth notion of index (number of negative eigenvalues of the Hessian): local minima occur on vertices, \( i \)-saddles on \( i \)-simplices and maxima on \( d \)-simplices. For typical scalar data, the input is generally provided as a PL scalar field \( f \) (Section II-A). Given this input, Robins et al. introduced an algorithm based on expansions [80] (in the sense of simple homotopy theory), which guarantees that each resulting critical \( d_i \)-simplex \( \sigma_i \) belongs to the lower star of a PL critical point of index \( d_i \).

**III. OVERVIEW**

Fig. 8 provides an overview of our approach. We assume that \( K \) is connected (otherwise, each connected component is processed independently by our algorithm). While
we focus on simplicial complexes in our work (Section I),
our algorithm can be applied in principle to arbitrary cell
complexes.

First, the discrete gradient of the input data is computed along
with its critical simplices via expansions [80]. The rest of our
approach consists in grouping the resulting critical simplices
into persistence pairs. We describe for that an extension (Sec-
tion IV) of the algorithm “PairSimplices” [31], [103], which
is expressed in the DMT framework for improved performances,
and that we call “PairCriticalSimplices” (Algorithm 3). While
each diagram \( D_0(f) \), \( D_1(f) \) and \( D_2(f) \) could be computed
with this algorithm, we describe instead a stratification strategy,
called sandwiching, (described below), which further improves
performances.

Second (Fig. 8, left), the diagram \( D_0(f) \) is obtained by
processing the unstable sets of the 1-saddles of \( f \) (Section V-A).

Third (Fig. 8, center) the diagram \( D_{d-1}(f) \) is obtained by pro-
cessing the stable sets of the \((d-1)\)-saddles of \( f \) (Section V-B).

Next (Fig. 8, right) for 3D data only, the diagram \( D_1(f) \)
is computed by restricting our novel algorithm “PairCri-
ticalSimplices” (Section IV) to the remaining set of unpaired
2-saddles.

Last, the remaining critical simplices are necessarily involved
in classes of infinite persistence, capturing the (infinitely per-
sistent) homology groups of \( K \) (Section VI).

IV. PAIRING CRITICAL SIMPLICES

This section presents our adaptation of the seminal algo-
rithm “PairSimplices” (Section II-E) to the DMT setting (Sec-
tion II-F), resulting in substantial performance gains.

A. Observations

This section describes three main observations regarding the
algorithm “PairSimplices” (Algorithm 1), which are at the basis
of our adaptation, described in Section IV-B.

(a) Dimension Separability. First, one can observe that the
different persistence diagrams \( D_0(f) \), \( D_1(f) \) and \( D_2(f) \) can be
computed in a separated manner, one after the other. Indeed, a
given \( d_i \)-simplex \( \sigma_i \) can only be involved in (i) the destruction
of a \((d_i-1)\)-cycle (if \( \partial \sigma_i \) was not trivial), or (ii) the creation
of a \( d_i \)-cycle (if \( \partial \sigma_i \) was trivial). For instance, the addition of
a 1-simplex in the lexicographic filtration connects two vertices
belonging either (i) to distinct connected components (in which
case a persistence pair is added to \( D_0(f) \), line 21, Algorithm 1)
or (ii) to the same connected component (in which case a new
1-cycle is created line 10, Algorithm 1), to be later added to
\( D_1(f) \). Thus, if the persistence diagram \( D_{i-1}(f) \) is available,
the diagram \( D_i(f) \) can be efficiently computed by restricting
Algorithm 1 to the \( i+1 \) simplices of \( K \) (still processed in
lexicographic order). Then, each \( i \)-simplex which has not been
paired yet in \( D_{i-1}(f) \) will be guaranteed to be the creator of an
\( i \)-cycle, and thus involved in a persistence pair of \( D_i(f) \). This
dimension separability is at the basis of our sandwiching strat-
ification strategy. This observation is not novel. It is also at the
basis of previous approaches, e.g., the clearing and compression
acceleration [4].

(b) Boundary Caching. As it is described in Zomorodian's
 textbook [103], the algorithm “PairSimplices” proceeds to the
homologous propagation of \( d_i \)-cycles by iteratively growing
\((d_i+1)\)-chains (line 13, Algorithm 1), and by explicitly ex-
tracting their boundary when needed (e.g. line 6, Algorithm 1).
However, each of these extractions requires a pass which is linear
with the size of the chain. This can be improved (as described
in Section IV-B) by caching the boundary of the chain created
at each simplex, and by manipulating boundaries directly in the
propagation process, instead of manipulating chains (similarly
to approaches based on boundary matrix reduction). This adap-
tation of the propagation process still requires a linear pass (for
the modulo-2 addition of boundary simplices), but this time on a
much smaller set (boundaries are in practice much smaller than
their chains).
(c) Zero-Persistence Skip. By definition of a filtration (Section II-B), a given simplex cannot be inserted in the filtration before its facets. A practical implication of this observation is that many, zero-persistence pairs are created by the algorithm “PairSimplices”. For instance, the insertion of the last edge $e_1$ of a given triangle $t_1$ often creates a new 1-cycle (step $K_{e_1}$, Fig. 9) which is immediately filled by the subsequent insertion of $t_1$ (step $K_{t_1}$, Fig. 9), creating a persistence pair $(e_1, t_1) \in D_1(f)$. However, since the persistence of a pair is given by the difference between the maximum vertex data values of $e_1$ and $t_1$ (Section II-D), we have in such cases $\partial(e_1, t_1) = 0$, as $e_1$ and $t_1$ have to share the highest vertex of $t_1$, given the lexicographic order. Thus, a significant time (c.f. Section IX-B) is spent in practice by the algorithm “PairSimplices” to construct persistence pairs with zero-persistence, which consequently do not contribute any information to the output diagram. We address this issue with DMT. In particular, the discrete gradient $G$ computed via expansions [80] guarantees that each critical simplex belongs to the lower star of a PL critical point of $f$, which exactly coincide themselves to changes in the homology groups of the lower star filtration (Section II-F). Then, all the remaining regular simplices (involved in a discrete vector of $G$) induce homology changes which are not captured by the lower star filtration (steps marked with a black frame in Fig. 9, $K'_{e_1}$ and $K'_{t_1}$), and which, equivalently, have zero-persistence. Then, it follows that all the zero-persistence pairs of the lexicographic filtration can be efficiently skipped in a pre-process, by discarding from the computation all the simplices which are not critical, as only the critical simplices will induce non-zero persistence homology changes (i.e., captured by the lower star filtration, Section II-F).

B. Algorithm

Our adaptation of the algorithm “PairSimplices” to the DMT setting, called “PairCriticalSimplices”, directly results from the above observations.

(a) Zero-Persistence Skip. First, Algorithm 2 is used in a pre-process to skip the zero-persistence pairs of the lexicographic filtration. This algorithm first computes the discrete gradient field $G$ given the input PL scalar field $f: \mathcal{K} \rightarrow \mathbb{R}$ with expansions [80] (line 2). At this stage, for any discrete vector $\{\sigma_i, \sigma_{i+1}\} \in G$, it is guaranteed that both $\sigma_i$ and $\sigma_{i+1}$ are involved in zero-persistence pairs (Section IV-A). Thus, by convention, we pair $\sigma_i$ and $\sigma_{i+1}$ together (line 9). Otherwise, critical $d_i$-simplices are marked as unpaired (line 12) and are added to the set $C_d$ of critical $d_i$-simplices (line 13). Once all discrete vectors have been processed, each set $C_d$ is sorted by increasing lexicographic order (line 17).

(b) Pair Critical Simplices. We now present our algorithm “PairCriticalSimplices” (Algorithm 3). For a given simplex dimension $d_i + 1$, this algorithm takes as an input the ordered set $C_{d_i+1}$ of critical $(d_i + 1)$-simplices and produces the diagram $D_{d_i+1}(f)$. This assumes that the diagram $D_{d_i+1}(f)$ has already been computed (see the Dimension separability property, Section IV-A) and that consequently, the critical $d_i$-simplices involved in $D_{d_i+1}(f)$ have already been paired. Since all the regular simplices inserted in between two critical simplices by the lexicographic filtration are guaranteed to belong to zero-persistence pairs (Zero-persistence skip property, Section IV-A), Algorithm 3 simply processes the critical simplices of $C_{d_i+1}$ in increasing lexicographic order. For each critical simplex $\sigma_j$, the standard, downwards homologous propagation of the classic algorithm “PairSimplices” is employed (line 5), possibly visiting in the process some simplices $\tau$ (line 6) which are not critical. In that case, $Pair(\tau)$ will return the simplex with which $\tau$ forms a discrete vector (c.f. Algorithm 2). As discussed in Section IV-A (Boundary caching property), our algorithm directly manipulates boundaries instead of the corresponding chains. Then, when a boundary homologous to $\partial\sigma_j$ is expanded (line 12), a modulo-2 addition is employed by manipulating a bit mask (indicating if a simplex $\sigma_j$ is already present in Boundary($\sigma_j$)). The rest of the algorithm is identical to the original algorithm “PairSimplices”: if the expanded boundary for the simplex $\sigma_j$ is not-empty (line 16), this means that a critical simplex $\tau$, creating a $d_i$-cycle, has been found during the downward homologous propagation (line 9). Then, a persistence pair $(\tau, \sigma_j)$ is created between $\sigma_j$ and the highest $d_i$-simplex $\tau$ of its expanded boundary Boundary($\sigma_j$) (line 21). Then,
Algorithm 3: Our Algorithm “PairCriticalSimplices”.

Input: Ordered set $C_{d+1}$ of critical $(d+1)$-simplices
Output: Persistence diagrams $D_d(f)$.
1: for $j \in C_{d+1}$ do
2: // Process the $(d+1)$-simplex $\sigma_j$
3: $\text{Boundary}(\sigma_j) \leftarrow \partial \sigma_j$
4: // Homologous propagation of $\partial \sigma_j$
5: while $\text{Boundary}(\sigma_j) \neq \emptyset$ do
6: $\tau \leftarrow \max(\text{Boundary}(\sigma_j))$
7: if $\text{Pair}(\tau) = \emptyset$ then
8: // $\tau$ is unpaired and thus created a $d_i$-cycle.
9: break
10: else
11: // Expand boundary
12: $\text{Boundary}(\sigma_j) \leftarrow \text{Boundary}(\sigma_j) + \text{Boundary}(\text{Pair}(\tau))$
13: end if
14: end while
15: if $\text{Boundary}(\sigma_j) \neq \emptyset$ then
16: // A non-trivial cycle homologous to $\partial \sigma_j$ exists (I. 9)
17: $	au \leftarrow \max(\text{Boundary}(\sigma_j))$
18: $\text{Pair}(\sigma_j) \leftarrow \tau$
19: $\text{Pair}(\tau) \leftarrow \sigma_j$
20: $D_d(f) \leftarrow D_d(f) \cup (\tau, \sigma_j)$
21: end if
22: end for

our algorithm “PairCriticalSimplices” is a direct geometric interpretation of boundary matrix reduction (as discussed for “PairSimplices”, Section II-E), but restricted to the columns of the boundary matrix corresponding to critical simplices.

V. EXTREMUM-SADDLE PERSISTENCE PAIRS

Our algorithm “PairCriticalSimplices” (Section IV-B) could be used as-is to compute the diagrams $D_0(f)$, $D_1(f)$ and $D_2(f)$ one after the other, already resulting in substantial performance gains over the seminal algorithm “PairSimplices” (see Section IX-B). In this section, we further exploit the Dimension separability property (Section IV) to further speedup the process.

A. Minimum-Saddle Persistence Pairs

This section introduces a faster alternative to the algorithm “PairCriticalSimplices”, for the specific case of $D_0(f)$.

(a) Unstable Set Restriction. This algorithm is based on the key observation that, for the specific case of $D_0(f)$, given a critical 1-simplex $\sigma^0_1$, the homologous propagation described in Algorithm 3 exactly coincides with the discrete unstable set (Section II-F) of $\sigma^0_0$ (see Fig. 10). In particular, at the first iteration of the algorithm, $\tau$ will be selected as one of the two vertices of $\sigma^0_0$, noted $\sigma^0_0$. If $\sigma^0_0$ is not a minimum itself, it has to be paired (given the discrete gradient $G$, Algorithm 2) with another edge $\sigma^1_0$, being one of its co-facets, with $\sigma^1_0 \neq \sigma^0_0$.

Since in simplicial complexes, edges are guaranteed to connect distinct vertices, we then have the property that the only other facet of $\sigma^1_0$ is another vertex $\sigma^0_0 \neq \sigma^0_0$. Thus, so far, the first iteration of the homologous propagation visited a sequence of edges and vertices $\{\sigma^0_0, \sigma^0_0, \sigma^1_0, \sigma^1_0\}$, such that for each item $\sigma^0_i$ in this sequence we have: (i) $\sigma^0_i \neq \sigma^0_{i+1}$ and (ii) $\sigma^0_i = \sigma^0_{i+1} < \sigma^0_{i+1}$, which exactly coincides with the definition of a discrete integral line (Section II-F). Then, along the iterations of Algorithm 3, two integral lines, started at each vertex of $\sigma^1_0$, will be iteratively constructed, by selecting at each iteration the highest extremity of the two integral lines (line 6). This process terminates when one of the two integral lines reaches a minimum $\sigma^0_0$ (i.e., an unpaired vertex, line 9). At this point, we have $\text{Boundary}(\sigma^0_0) = \{\sigma^0_0 + \sigma^0_0\}$, where $\sigma^0_0$ is the extremity of the other integral line (Fig. 10, left). Then, if $\sigma^0_0 \neq \sigma^0_0$ (i.e., $\sigma^0_0$ did not create a 1-cycle), we have $\text{Boundary}(\sigma^0_0) = \emptyset$ (line 16) and a persistence pair $(\sigma^0_0, \sigma^0_0)$ is created (line 21). Then, at this stage, the boundary propagation completed the first integral line from $\sigma^0_0$ down to $\sigma^0_0$ and paused the second integral line at $\sigma^0_0$ and we have, by construction, $\text{Boundary}(\sigma^0_0) \sim \sigma^0_0$. Next, it is possible that later in the algorithm, the expanded boundary $\text{Boundary}(\sigma^0_1)$ of another critical 1-simplex $\sigma^1_0$ hits the minimum $\sigma^0_0$. In such a case, the expanded boundary of its paired simplex (i.e., $\text{Boundary}(\sigma^1_0)$) will then be added (modulo-2) to $\text{Boundary}(\sigma^0_1)$ (line 13) and the second integral line started in $\sigma^0_1$ (paused at $\sigma^0_0$) will eventually be resumed from $\sigma^0_0$ until it hits another minimum (Fig. 10, center).

Overall, for $D_0(f)$, Algorithm 3 will exactly visit the edges and vertices of $K$ which are located on the unstable sets of the critical 1-simplices. It follows that the homologous propagation of a critical 1-simplex $\sigma_1$ can be accelerated by directly considering its unstable set, whose boundary $\{(\sigma^0_0 + \sigma^0_0')$, c.f. above} is homologous by construction to $\partial \sigma_1$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Note that this observation no longer holds in higher dimensions. For instance, when constructing $D_1(f)$ on a 3-dimensional simplicial complex, in contrast to the case of $D_0(f)$ described above, the unstable set of a critical 2-simplex $\sigma_2$ may become non-manifold (as described by Gyulassy and Pascucci [48] in the study of Morse-Smale complexes, yielding multiple integral lines between a given pair of critical simplices). In such a case, the boundary of the unstable set of $\sigma_2$ is no longer exactly homological to $\partial \sigma_2$ (due to the non-manifold elements of the surface) and the acceleration described above is no longer applicable.

(b) Unstable Set Compression. The computation of $D_0(f)$ can be further accelerated by compressing all unstable sets. Given the unstable sets of the critical 1-simplices, we collapse all their regular edges (which are involved in zero-persistence pairs, Section IV-A). This collapse eventually results in a graph $G_{D_0(f)}$ (Fig. 10, center), whose nodes and arcs respectively correspond to the vertices and edges of $K$ which are left unpaired by $G$ and whose adjacency relations are determined by the input unstable sets. At this point, $G_{D_0(f)}$ can be directly given as an input to Algorithm 3 to compute $D_0(f)$.

(c) Connectivity Tracking. Given $G_{D_0(f)}$, we further accelerate the process and simplify Algorithm 3 by exploiting the specific dimensionality of $D_0(f)$. In particular, in the case of $D_0(f)$, Algorithm 3 visits the arcs of $G_{D_0(f)}$ in increasing (original) lexicographic order. For a given arc $\sigma_1$, two cases can occur.

First (i), the highest node $\sigma_0$ of $\sigma_1$ has not been visited yet by any propagation (line 7) and a persistence pair $(\sigma_0, \sigma_1)$ is created in $D_0(f)$ (line 21). Then the arc $\sigma_1$ can be collapsed (similarly to regular edge compression, above paragraph (b)) to indicate that it can no longer be paired by the algorithm. This collapse can be modeled by a union operation, indicating that the other node $\sigma_0'$ of $\sigma_1$ becomes the representative of $\sigma_0$ (which can no longer be paired).

Second (ii), the highest vertex $\sigma_0$ of $\sigma_1$ has already been visited by a prior propagation, in which case we need to efficiently find its other boundary vertex $\sigma_0'$ (line 13) to resume the propagation there.

Overall, $D_0(f)$ can be computed from $G_{D_0(f)}$ by collapsing its arcs as they are visited and recording these collapses with a union operation, such that boundary nodes can later be retrieved with a find operation. This can be efficiently implemented with a Union-Find data structure [22], since for $D_0(f)$, each node needs to record only one representative (the representative of the other node of its paired arc).

(d) Summary. Overall (Fig. 10), our algorithm computes $D_0(f)$ by first constructing the unstable sets of each critical 1-simplex. Next, each regular edge in these unstable sets is collapsed to create the graph $G_{D_0(f)}$. Finally, $G_{D_0(f)}$ is processed with a Union-Find data structure [22] to compute $D_0(f)$. Initially a Union-Find node $UF(\sigma_0)$ is created for each node $\sigma_0$ of $G_{D_0(f)}$ and the arcs of $G_{D_0(f)}$ are processed in increasing (original) lexicographic order. Given an arc $\sigma_1$, its two expanded boundary nodes $\sigma_0$ and $\sigma_0'$ are efficiently retrieved by applying the find operation on the two nodes of $\sigma_1$. Then, if $\sigma_0$ is strictly higher than $\sigma_0'$, the persistence pair $(\sigma_0, \sigma_1)$ is created in $D_0(f)$ and a union operation is performed between the nodes $UF(\sigma_0)$ and $UF(\sigma_0')$, and the unpaired node $UF(\sigma_0')$ is used as a representative.

B. Saddle-Maximum Persistence Pairs

In this section, we detail our strategy for the computation of $D_{d-1}(f)$. In particular, we exploit within the DMT setting the duality argument discussed by Edelsbrunner et al. [30], [31], recently revisited for general cell complexes in higher dimensions [36], and we document its implementation in the DMT setting. This duality argument (Fig. 11) states that, at a given step $i$ of the filtration, the $(d-1)$-dimensional voids of $K_i$, under certain conditions, exactly coincide with the connected components of the complement $K_i^*$ of $K_i$.

Formally, let $K^*$ be the dual cell complex of $K$. Specifically, each $(d-i)$-simplex $\sigma_i$ of $K$ is represented by an $i$-dimensional cell $\sigma_i^*$ in $K^*$. Moreover, given two simplices $\sigma_i < \sigma_j$ in $K$, we have $\sigma_i^* < \sigma_j^*$ in $K^*$ (i.e., face-coface relations are reversed). Then, it follows that each diagram $D_{d-k-1}(f)$ of the lexicographic filtration of $K$ is equal to the opposite of the diagram $D_k(\neg f)$ (i.e., the diagram of the backward lexicographic filtration, $-f$) of $K^*$ (see Garin et al. [36], Theorem 2.1). This implies in particular that $D_{d-1}(f)$ can be computed very efficiently by applying the algorithm for $D_0(f)$ described in Section V-A to the backward filtration (i.e., $-f$, reverse order) of the dual $K^*$ of $K$. This observation nicely translates to the DMT setting, as described next.

A dual discrete gradient vector field $G^*$ (Fig. 12) can be easily defined on the dual $K^*$ of $K$ by inverting each discrete vector of $G$. In particular, each discrete vector $\{\sigma_{d-1}, \sigma_d\}$ between a $(d-1)$-simplex $\sigma_{d-1}$ of $K$ and one its cofaces can be inverted into $\{\sigma_d^*, \sigma_d^{d-1}\}$, where $\sigma_d^*$ and $\sigma_d^{d-1}$ are the simplices dual to $\sigma_d$ and $\sigma_{d-1}$ in $K^*$. Then $\{\sigma_d^*, \sigma_d^{d-1}\}$ is a discrete vector between a 0-simplex ($\sigma_d^*$) and a 1-simplex ($\sigma_d^{d-1}$). Once this is established, the algorithm described in Section V-A can be applied as-is on $G^*$. Additionally, one can observe that the critical 1-simplices of $G^*$ will be, by construction, critical $(d-1)$-simplices of $G$ and that their unstable sets in $G^*$ will exactly coincide to stable sets in $G$.

Thus, our algorithm for computing $D_0(f)$ (Section V-A) can be easily adapted to compute $D_{d-1}(f)$ as follows. The stable sets of each critical $(d-1)$-simplex are first constructed. Next,
Fig. 12. Given a discrete gradient field \( \mathcal{G} \) defined on \( \mathcal{K} \) (left), its dual discrete gradient field \( \mathcal{G}^* \) (right) is obtained by considering the dual cell complex \( \mathcal{K}^* \) and reverting each arrow of \( \mathcal{G} \): each vertex-edge arrow (blue, left) becomes an edge-face arrow (green, right) while each edge-triangle arrow (green, left) becomes a vertex-edge arrow (blue, right), along which unstable sets can be easily defined and computed.

each discrete vector in these stable sets is collapsed, to create a graph \( G_{D_{d-1}}(f) \), where each node represents a critical \( d \)-simplex and each arc a critical \( (d-1) \)-simplex. Finally, \( G_{D_{d-1}}(f) \) is processed with a Union-Find data structure (Section V-A), but in decreasing lexicographic order, and a persistence pair \((\sigma_d, 1), (\sigma_d, 0)\) is created in \( D_{d-1}(f) \) for each connected component of \( G_{D_{d-1}}(f) \) created in \( \sigma_d \) and merged into another by the addition of the arc representing \( \sigma_{d-1} \).

Domains With Boundary. When \( \mathcal{K} \) is not closed, a slight variation of the above algorithm is considered. For domains with boundary, in specific configurations, the connected components of the backward lexicographic filtration of \( \mathcal{K}^* \) may no longer exactly coincide with the voids of the forward lexicographic filtration of \( \mathcal{K} \). In particular, when a connected component of the backward filtration of \( \mathcal{K}^* \) first hits the outer boundary component of \( \mathcal{K} \) (by construction, on a critical \( (d-1) \)-simplex), it no longer describes a void inside the object, as it merges with the rest of the outside space (thus deleting the corresponding cavity). To take this into account, we assign a virtual discrete maximum with infinite function value to the outer boundary component of \( \mathcal{K} \) (representing the outside space) and apply the rest of the above algorithm as-is. Then, when a connected component of backward filtration hits the outer boundary, it is considered, given the above adjustment, to die there as it merged with an (infinitely) older component (the outside space).

Note that this specific adjustment comes with no additional computational overhead as the rest of our algorithm is used as-is (only one, extra virtual maximum is considered by the algorithm). In our implementation, this adjustment is optional as its practical relevance can be questionable for real-life data, as illustrated in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2023.3238008.

VI. CRITICAL SIMPLICIES OF INFINITE PERSISTENCE

To summarize, our overall approach first computes \( D_0(f) \) (Section V-A) and \( D_{d-1}(f) \) (Section V-B). Finally, if \( d = 3 \), \( D_1(f) \) is computed with our novel algorithm “PairCriticalSimplices” (Section IV). During this process, certain critical simplices may remain unpaired after the above algorithms have finished. These correspond to homology classes with infinite persistence, which exactly characterize the homology of \( \mathcal{K} \). Specifically, each remaining unpaired \( d \)-simplex \( \sigma_i \) yields a persistence class with infinite persistence in \( D_i(f) \), which we embed, by convention at location \((f(\sigma_i), f^*)\), where \( f^* \) denotes the maximum \( f \) value. Such points in the diagrams are marked with a specific flag (Fig. 5), as they describe more the domain \( \mathcal{K} \) itself than the data \( f \) defined on it.

VII. COMPUTATIONAL ASPECTS

This section details the computational aspects of our algorithm, including time complexity and parallelism.

A. Time Complexity

The first stage of our approach consists in establishing the lexicographic filtration with a global sort in \( O(n \log(n)) \) steps (where \( n \) is the total number of simplices in \( \mathcal{K} \)).

The second stage computes a discrete gradient by expansions [80]. This operation takes \( O(n_v) \) where \( n_v \) is the number of vertices in \( \mathcal{K} \).

The third stage consists in computing \( D_0(f) \) (Section V-A). The first step of this algorithm computes the unstable sets of each 1-saddle to construct the graph \( G_{D_0}(f) \), which is done in \( O(n_v) \) steps in practice, where \( n_v \) is the number of edges in \( \mathcal{K} \). Processing \( G_{D_0}(f) \) with a Union-Find data-structure to finally construct \( D_0(f) \) takes \( O(n_v \alpha(n_v)) \) steps, where \( \alpha() \) is the extremely slowly-growing inverse of the Ackermann function. Computing \( D_{d-1}(f) \) requires the same steps.

The fourth stage of our approach, only for \( d = 3 \), applies our algorithm “PairCriticalSimplices” (Algorithm 3). Similarly to the seminal algorithm “PairSimplices” [31], [103], our algorithm requires \( O(n^3) \) steps in the worst case. For each critical simplex (in the worst case, \( n \) steps, line 1), an homologous propagation is performed (in the worst case, in \( n \) steps, lines 5 to 15), which itself requires at each step a possibly linear pass to expand the boundary of the current critical simplex with modulo-2 additions (line 12). However, as documented in Section IX-B, our algorithm “PairCriticalSimplices” performs in practice significantly faster than the algorithm “PairSimplices” since: (i) it only considers the critical simplices (and not all the simplices of \( \mathcal{K} \)), (ii) the critical simplices already present in \( D_0(f) \) and \( D_{d-1}(f) \) are discarded from the computation (which provides further accelerations), (iii) it maintains the expanded boundary of the considered critical simplex and not its expanded chain (which is significantly bigger).

Overall, our approach has the advantage of being output-sensitive. In particular, the size (number of nodes) of the graphs \( G_{D_0}(f) \) and \( G_{D_{d-1}}(f) \) corresponds to the number of minima and maxima of \( f \) and consequently to the size of \( D_0(f) \) and \( D_{d-1}(f) \).

The time complexity of our algorithm “PairCriticalSimplices” is parameterized by the number of remaining saddle-saddle pairs, which corresponds to the size of \( D_1(f) \). Then our approach will provide superior performances when considering smooth data sets, as typically found in various simulation domains.
B. Shared Memory Parallelism

Our approach can benefit from further accelerations thanks to shared-memory parallelism. The first stage (establishing the lexicographic filtration) can be done with parallel sorting (see the GNU parallel sort for an implementation example). The second stage (discrete gradient computation [80]) is trivially parallelizable on the vertices of \(K\). Regarding the third stage, computing \(D_0(f)\), the computation of the unstable sets is parallelized on a per 1-saddle basis and the processing of \(G_{D_0}(f)\) with the Union-Find data-structure is then done sequentially. In practice \(D_0(f)\) and \(D_{d-1}(f)\) are computed in parallel thanks to a task pool mechanism. Regarding the fourth stage (Algorithm 3), the homologous propagations can be computed in parallel for each 2-saddle, using lightweight synchronizations [39], [70]. Specifically, when the homologous propagation of a 2-saddle \(\sigma_2\) hits an unpaired 1-saddle \(\sigma_1\), a temporary persistence pair \((\sigma_1, \sigma_2)\) is created in \(D_1(f)\). Later, if the homologous propagation of another 2-saddle \(\sigma_2'\) also hits \(\sigma_1\), an atomic compare-and-swap operation is performed. Specifically, if \(\sigma_2'\) is anterior to \(\sigma_2\) in the lexicographic filtration, the pair \((\sigma_1, \sigma_2)\) is updated into \((\sigma_1, \sigma_2')\) and the propagation of \(\sigma_2\) is resumed at \(\sigma_1\).

VIII. APPLICATION TO GENERATOR EXTRACTION

This section presents an application of our contributions to the fast extraction of persistent 1-dimensional generators. While the topological persistence computed by our algorithm is a central simplification criterion in data visualization (Fig. 1), the information maintained by our algorithm can additionally be exploited directly for visualization purposes. Specifically, Iurichich [55] suggested to extract, for a given persistence pair \((\sigma_i, \sigma_j)\), a representative \(d_i\)-cycle homologous to \(\partial \sigma_j\), specifically, the earliest homologous \(d_i\)-cycle, created at \(\sigma_i\). For that, Iurichich introduced a specific post-processing algorithm [55], requiring the persistence diagram to be computed in a pre-processing step.

In contrast, in our work, this information is precisely maintained throughout the entire computation, for all 1-dimensional persistence pairs, and is then readily available when our persistence diagram computation algorithm has finished, resulting in further accelerations. Specifically, for each critical 2-simplex \(\sigma_j\), the homologous propagation of Algorithm 3 (lines 5 to 15) iteratively reconstructs with \(Boundary(\sigma_j)\) a sequence of 1-cycles homologous to \(\partial \sigma_j\) and any of these can be chosen as a representative generator. Specifically, we store the earliest cycle (which is not necessarily optimal [28]), which is the cycle precisely obtained at the end of the homologous propagation (line 16), when the first unpaired simplex \(\tau\) is visited. Note that the problem of extracting persistent 0 and \((d-1)\)-dimensional generators is significantly simpler and has already been addressed via merge tree based segmentations [10], [17], [38].

Figs. 13, 14, 15 and 16 illustrate the ability of persistent 1-cycles to robustly capture circular patterns on surfaces, volume data and high-dimensional point clouds respectively.

IX. RESULTS

This section presents experimental results obtained with a C++/OpenMP implementation of our approach, integrated in TTK (commit: bb3089f). Our experiments were mostly run on a commodity desktop computer with two Xeon CPUs (3.0 GHz, 2x4 cores, 64 GB of RAM), while specific scalability experiments were run on a large shared-memory system with 128 Xeon CPUs (2.6 GHz, 128x8 cores, 16TB of RAM).

A. Experimental Data

We consider a list of 34 scalar datasets available on a public repository [58], provided as 3D regular grids of various resolutions and data types. These datasets come from diverse...
The intermediate layer of diameter $k$ projected

The 2D Rips complex (computed in $\mathbb{R}^{32^2}$) to infer the structure of the space sampled by the point cloud, by adding a triangle in the complex if its diameter (the maximum pairwise distance between its vertices) is smaller than a threshold $\epsilon$. The Rips complex is shown in 3D (via MDS projection), although it is computed in $\mathbb{R}^{32^2}$.

$(d)$ The infinitely persistent 1-cycle of the diameter function (for each vertex, average of the diameter of its adjacent triangles) robustly captures the circular pattern synthetically injected in the data (Figure 16(a)), hence confirming the ability of persistent 1-cycles to recover circular patterns in high-dimensional data.

fields (bio-imaging, material sciences, combustion, quantum chemistry, fluid dynamics) and have been either acquired (e.g., CT scans) or simulated. We also consider two extreme cases: (i) an elevation function (yielding the smallest possible output, a single bar in $D_0(f)$, with infinite persistence), (ii) a random function (yielding the largest outputs in practice). Since our approach is output sensitive, we re-sampled all datasets to a common resolution (192$^3$), to better observe runtime variations solely based on the output size. This common resolution has been chosen such that most of the implementations considered in our benchmark (Section IX-C) could run on our setup. Some available datasets were too large to fit in the memory of our desktop computer and could not be downsampled to the common resolution. These have not been considered in the benchmark as we believe our desktop computer to be representative of the machines used by potential benchmark users.

We generated 2D datasets by taking a slice of each original 3D regular grid along the Z-coordinate (at mid-value). These 2D datasets were re-sampled to 4,096$^2$. Finally, we generated 1D datasets by considering a line of each 2D dataset (Y-coordinate, mid-value). These 1D datasets were re-sampled to a common resolution of 1,048,576 vertices.

Each of these 1D, 2D and 3D datasets were then triangulated into a simplicial complex by breaking up each cell into two triangles in 2D, and five tetrahedra in 3D. As discussed in Section I, our approach focuses on this generic input representation based on simplicial complexes and we will therefore consider these representations for our experiments. This results overall in 108 input datasets.

As described in Section IX-C, some of the public implementations considered in our benchmark are specialized (or include specialized backends) for regular grids. However, they do not all interpret the input data in a consistent manner. For instance, some implementations (such as Gudhi) consider the input scalars to be defined on a per voxel basis, while others (such as Dipha or CubicalRipser) consider them as defined on a per vertex basis, which results in cell complexes of significantly different sizes (in particular, penalizing Gudhi). Moreover, some implementations (such as Oineus, PairSimplices, PersistenceCycles, TTK-FTM, DMS) implicitly triangulate the input regular grid data, which changes the size of the input complex. First, since these internal data representations differ, the generated outputs will, consequently, not be exactly identical. Second, since these differences in internal representation result in cell complexes of significantly different sizes, they also induce a strong bias in runtime comparison. For these reasons, we decided to focus our analysis on the methods which natively support simplicial complexes, for which a direct and unbiased comparison can be performed. For completeness, we provide performance numbers for regular grids in Appendix B, available in the online supplemental material, but we stress that the inconsistencies in the internal representations and in the generated outputs prevent a direct and unbiased comparison.

### B. Performance Analysis

This section evaluates the time performance of our overall approach, named “DiscreteMorseSandwich” (DMS), and details the gains provided by each step of our algorithm, in comparison to the original algorithm “PairSimplices”.

Fig. 17 provides time performance curves for the 1D, 2D and 3D versions of our 36 input datasets, where computation speeds (in simplices per second, log scale) are reported as a function of the output size, and where the seminal algorithm “PairSimplices” is compared to four variants of our approach, to evaluate the performance gain of each acceleration introduced in our algorithm.

This figure confirms the output-sensitive behavior of our overall approach (DMS, blue curves), as computation speeds decrease for increased output sizes. As expected by our time complexity analysis (Section VII-A), the lowest speeds occur for 3D datasets ($2.95 \times 10^7$ simplices/sec on average) since there, the computation of $D_1(f)$ (the intermediate layer of the sandwich) has a less favorable time complexity than for
Computation speeds (simplices per second, log scale), as a function of the output size, for the distinct accelerations of our algorithm, in comparison to the seminal algorithm “PairSimplices” (red, Algorithm 1). Each pair of curves corresponds to a specific variant of our algorithm: “PairCriticalSimplices” (green, Algorithm 3), with boundary caching (yellow, Section IV-B), with “Sandwiching” (purple, Section V-A), in parallel (blue, 8 cores, Section VII-B). On average, our parallel algorithm computes in 0.04 (1D), 1.10 (2D) and 8.91 (3D) seconds, and achieves a parallel efficiency of 30.68% (1D), 59.50% (2D), and 78.89% (3D) for an overall speedup over “PairSimplices” of ×8 in 1D, ×1,046 in 2D, and ×323 in 3D.

C. Performance Benchmark

This section describes our benchmark for evaluating and comparing various public implementations for persistent diagram computation. Our Python benchmark package (https://github.com/pierre-guillou/pdiags_bench) (i) downloads and prepares the benchmark data (Section IX-A), (ii) downloads, builds and executes each implementation (Section IX-C) and (iii) aggregates the output information to produce the results provided in this section.

1) Implementations: Our benchmark includes the implementation of our algorithm “DiscreteMorseSandwich” (DMS) as well as 14 other implementations, whose specifications are reported in Table I. A few clarifications are needed regarding certain implementations. In particular, TTK-FTM only computes \(D_0(f)\) and \(D_{d-1}(f)\). Ripser and its scikit-tda version both reported integer overflows for relatively large inputs (issue communicated to the authors). A number of implementations (among the category “Boundary Matrix Reduction”) require an explicit boundary matrix as an input, which we compute in a pre-processing stage with TTK.

2) Output Comparison: To check the correctness of our implementation, we compute the \(L_2\)-Wasserstein distance [30] between our output and the one computed by each implementation included in the benchmark, for each dataset, for each output dimension. To enable the direct comparison of this distance across datasets, we consider as input scalar field \(f\) the vertex order, after sorting all input data values (i.e., \(f(v) \in [0, n_v - 1]\)). The average distance for all datasets, for all output dimensions, is reported in the column “Distance” of Table I. These numbers show that our implementation generates outputs which are strictly identical to most other implementations (PairSimplices, Dionysus2, DIPHA, Javaplex, etc.). Variations from 0 seem to indicate slight inaccuracies for the corresponding implementation. For instance, for handling boundary effects, TTK-FTM only implements the second strategy described in Appendix A, available in the online supplemental material, (i.e., it ignores the virtual maximum on the boundary), which impacts distance evaluations for \(D_{d-1}(f)\) (see Appendix A, available in the online supplemental material). Note that this distance is only reported for the (non timed out) implementations natively supporting simplicial complexes, as outputs differ significantly in the case of regular grids (depending on the implementation’s data interpretation, see Section IX-A).
3) Performance Metrics: We evaluate performance along two major aspects: computation time and memory requirement. Regarding computation time, we consider the timings reported by each implementation, from which we remove the input/output times (for reading the input from disk and writing the output to disk). We also do not include the pre-processing time dedicated to boundary matrix computation, for the implementations which require this input form. Thus, our timings only include the core computation phase and we report in the following computation speeds, expressed in number of simplices per second. To enable an acceptable overall runtime (for the entire benchmark), we decided to interrupt all computations after a pre-defined timeout threshold of 15 minutes for all experiments.

Memory usage is evaluated with Python’s standard library https://docs.python.org/3/library/resource.html resources and we report the maximum resident set size of each implementation (run in a dedicated subprocess).

4) Benchmark Results: Fig. 18 first reports, for each input dimension, the computation speed in sequential mode for all the (non timed out) implementations supporting simplicial complexes natively, for which no parallelism was observed or for which the number of threads could be explicitly set to one. There, as can be expected, the only non-C++ implementation (Eirene.jl) provides the lowest speeds. Overall, the other C++ based implementations report computation speeds between $10^5$ and $10^8$ simplices per second. PHAT and our method DMS report the fastest sequential runtimes for all dimensions, with DMS improving over PHAT in 1D and 2D by 48% and 51% respectively, while PHAT provides the best average sequential times in 3D, with a gain of 38% over DMS. In Fig. 18, PersistenceCycles [55] is the method which is the most related to our approach. However, similar to other DMT-based approaches [41], [68], [72], [80], it computes the full Morse complex (prior to running a standard boundary matrix reduction on it). In contrast, our approach computes only the necessary subparts of the Morse complex: the (1D) unstable sets of 1-saddles for $D_0(f)$, the (1D) stable sets of 2-saddles for $D_2(f)$ – which are much smaller than their (2D) unstable sets, and the (2D) unstable sets for only a small subset of the 2-saddles, specifically, only those involved in $D_2(f)$. This careful selection already provides a significant performance gain (Appendix C), available in the online supplemental material. Next, the construction of $D_0(f)$ and $D_2(f)$ uses a Union-Find data-structure, which is much more efficient than boundary matrix reduction. Overall, this results in runtime gains of 92% and 89% of DMS over PersistenceCycles in 2D and 3D respectively.

Next, Fig. 19 reports, for each input dimension, the computation speed in parallel mode (using 8 cores) for all the (non timed out) implementations supporting simplicial complexes natively, for which parallelism was observed (typically using all available cores, in certain phases of the algorithm, for instance sorting) or for which the number of threads could be controlled explicitly. In this figure, note that only the pre-processing sorting step of Gudhi benefits from parallelism, the core of its algorithm being sequential. Similarly to the sequential case, the only non-C++ implementation (JavaPlex) provides the lowest speeds, as can be expected. The other C++ based implementations all report computation speed increases when parallelism is activated. In comparison to PersistenceCycles specifically (the other method of Fig. 19 based on DMT), our method DMS improves runtimes by 87% in 2D and 92% in 3D. Overall, DMS reports the fastest parallel runtimes for all dimensions, improving runtimes by 74%, 77% and 56% over the fastest competing technique, in 1D, 2D and 3D respectively. These fastest runtimes can be explained by an improved parallel efficiency over competing techniques. In particular, the discrete gradient computation is trivially parallel, while several other steps of our approach are also parallelized efficiently (Section VII-B).

We further investigate parallel scalability in Fig. 20, which reports for the most efficient parallel implementations, in 2D (left) and 3D (right), their average computation speeds (average over all datasets) when increasing the number of used cores. This figure indicates that no implementation scales significantly when increasing the number of cores and that most implementations reach a high plateau of speed (essentially due to the remaining sequential parts of the algorithms) beyond 32 cores in 2D and 96 cores in 3D. Moreover, PHAT, which presented encouraging parallel performances on a desktop computer, seems to suffer
drastically from NUMA effects on our large system, resulting in an absence of parallel acceleration. Overall, DMS reports the fastest performances on this system, improving runtimes with 128 cores by 76% and 52% over the fastest competing technique, in 2D and 3D respectively.

Together, Figs. 19 and 20 also indicate that DMS is more versatile than other approaches, as it outperforms the most appropriate implementation for each system (PHAT for the desktop computer, and DIPHA for the large system).

Finally, Table II reports the memory footprint for all the implementations supporting simplicial complexes natively, and for which the computation completed successfully. There, one can observe that the methods supporting parallelism have a very similar (if not identical) memory footprint when parallelism is activated. Overall, the methods taking a boundary matrix as an input tend to have the largest memory footprints. In contrast, DMS uses TTK’s internal triangulation data-structure [94] for modeling the input simplicial complex, which can be interpreted as a sparse representation of the boundary matrix, resulting in substantial improvements over the most competitive techniques, by 25%, 5% and 15% in 1D, 2D and 3D respectively.

D. Limitations

Similarly to the original algorithm “PairSimplices” [31, [103], our variant “PairCriticalSimplices” can work in principle in arbitrary dimension. However, Robin’s expansion algorithm [80] provides strong guarantees – regarding the correspondence between critical simplices and PL critical points – for input datasets in up to three dimensions. Beyond, such a correspondence is no longer guaranteed and our zero-persistence skip procedure (Algorithm 2) may no longer be valid.

X. CONCLUSION

This paper introduced an efficient algorithm for the computation of persistence diagrams for scalar data. Specifically, we documented a stratification strategy, which (i) computes the easiest diagrams first ($D_1(f)$ and $D_{d-1}(f)$) with an efficient Union-Find based processing applied to a carefully selected subset of the stable and unstable sets of saddles and which then (ii) efficiently computes the remaining diagram ($D_1(f)$) by revisiting the seminal algorithm “PairSimplices” [31, [103] in the context of discrete Morse theory. Experiments on 36 public datasets validated the performance improvements of our approach over the algorithm “PairSimplices”, with two orders of magnitude speedups in 3D. A comprehensive benchmark including 14 public implementations for persistent homology computation indicated that our approach provides the lowest memory footprints, as well as the fastest parallel performances. Additionally, our experiments illustrated the versatility of our approach, as it outperforms (in 1D, 2D and 3D) the most appropriate methods for each tested system (PHAT for the desktop computer and DIPHA for the large system), providing users with performance confidence irrespective of their system.

In the future, we will investigate alternative strategies for discrete gradient computation, in order to extend our zero-persistence skip procedure to arbitrary dimensions. Moreover, we will explore strategies for distributed computation, to further improve parallel scalability on large systems.

ACKNOWLEDGMENTS

The authors thank Joshua Levine for proofreading, and the reviewers for their constructive and helpful feedback.

REFERENCES

[1] T. F. Banchoff, “Critical points and curvature for embedded polyhedral surfaces,” Amer. Math. Monthly, vol. 77, pp. 475–485, 1970.
[2] S. Barannikoff, “Frmed Morse complexes and its invariants,” Adv. Sov. Math., vol. 21, pp. 93–116, 1994.
[3] U. Bauer, “Ripser: Efficient computation of Vietoris-Rips persistence barcodes,” 2019. [Online]. Available: https://github.com/Ripser/ripser/
[4] U. Bauer, M. Kerber, and J. Reininghaus, “Clear and compress: Computing persistent homology in chunks,” in Proc. Topol. Methods Data Anal. Vis., 2014, pp. 103–117.
[5] U. Bauer, M. Kerber, and J. Reininghaus, “Distributed computation of persistent homology,” in Proc. Meeting Algorithm Eng. Expermiments, 2014, pp. 31–38.
[6] U. Bauer, M. Kerber, J. Reininghaus, and H. Wagner, “PHAT - Persistent homology algorithms toolbox,” J. Symbolic Comput., vol. 78, pp. 76–90, 2017.
[7] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer, “TopoMS: Comprehensive topological exploration for molecular and condensed-matter systems,” J. Comput. Chem., vol. 39, pp. 936–952, 2018.
[8] S. Biasotti, D. Giorgio, M. Spagnuolo, and B. Falcidieno, “Reeb graphs for shape analysis and applications,” Theor. Comput. Sci., vol. 392, pp. 5–22, 2008.
[9] T. Bin Masood et al., “An overview of TTK,” in Proc. Topol. Methods Data Anal. Vis., 2019, pp. 327–342.
[10] A. Bock, H. Doraiswamy, A. Summers, and C. T. Silva, “TopoAngler: Interactive topology-based extraction of fishes,” IEEE Trans. Vis. Comput. Graph., vol. 24, no. 1, pp. 812–821, Jan. 2018.
[11] J. Boissonnat, T. K. Dey, and C. Maria, “The compressed annotation matrix: An efficient data structure for computing persistent cohomology,” Algorithmica, vol. 73, pp. 607–619, 2015.
[12] J. Boissonnat and C. Maria, “The simplex tree: An efficient data structure for general simplicial complexes,” Algorithmica, vol. 70, pp. 406–427, 2014.
[13] J. Boissonnat and S. Pritam, “Edge collapse and persistence of flag complexes,” in Proc. Int. Symp. Comput. Geometry, 2020, pp. 19:1–19:15.
[14] P. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci, “A topological hierarchy for functions on triangulated surfaces,” IEEE Trans. Vis. Comput. Graph., vol. 10, no. 4, pp. 385–396, Jul/Aug. 2004.
[15] P. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell, “Interactive exploration and analysis of large scale simulations using topology-based data segmentation,” IEEE Trans. Vis. Comput. Graph., vol. 17, no. 9, pp. 1307–1324, Sep. 2011.
[16] H. Carr, J. Snoeyink, and U. Axen, “Computing contour trees in all dimensions,” in proc. Ann. ACM-SIAM Symp. Discrete Algorithms, 2000, pp. 918–926.
[17] H. A. Carr, J. Snoeyink, and M. van de Panne, “Simplifying flexible isosurfaces using local geometric measures,” in Proc. IEEE Vis., 2004, pp. 497–504.
[67] J. Milnor, *Morse Theory*, Princeton, NJ, USA: Princeton Univ. Press, 1963.

[68] K. Mischaikow and V. Nanda, “Morse theory for filtrations and efficient computation of persistent homology,” *Discrete Comput. Geometry*, vol. 50, pp. 330–353, 2013.

[69] D. Morozov, “Dionysus2,” 2017. Accessed: May 19, 2021. [Online]. Available: http://www.mrzv.org/software/dionysus2

[70] D. Morozov and A. Nigmetov, “Brief announcement: Towards lock-free persistent homology,” in *Proc. Symp. Parallelism Algorithms Archit.*, 2020, pp. 555–557.

[71] M. Morse, *The Calculus of Variations in the Large*. New York, NY, USA: Amer. Math. Soc., 1934.

[72] V. Nanda, “Perseus, the persistent homology software,” 2021. Accessed: May 18, 2021. [Online]. Available: http://people.maths.ox.ac.uk/nanda/perseus/

[73] A. Nigmetov and D. Morozov, “Oineus,” 2021. Accessed: May 19, 2021. [Online]. Available: https://github.com/grey-narv/oineus

[74] P. Oesterling, C. Heine, J. Jänicke, G. Scheuermann, and G. Heyer, “Visualization of high-dimensional point clouds using their density distribution’s topology,” IEEE Trans. Vis. Comput. Graph., vol. 17, no. 11, pp. 1547–1559, Nov. 2011.

[75] M. Olejniczak, A. S. P. Gomes, and J. Tierny, “A topological data analysis perspective on non-covalent interactions in relativistic calculations,” *Int. J. Quantum Chem.*, vol. 120, no. 20, Art. no. e26133.

[76] S. Parsa, “A deterministic \(O(n \log n)\) time algorithm for the Reeb graph,” in *Proc. Int. Symp. Comput. Geometry*, 2012, pp. 269–276.

[77] V. Pascucci, G. Scorzelli, P. T. Bremer, and A. Mascarenhas, “Robust on-line computation of Reeb graphs: Simplicity and speed,” ACM Trans. Graph., vol. 26, pp. 58–es, 2007.

[78] G. Reeb, “Sur les points singuliers d’une forme de Pfaff complétement intégrable ou d’une fonction numérique,” *Comptes Rendus des séances de l’Académie des Sci.*, vol. 222, no. 849–849, 1946, Art. no. 76.

[79] V. Robins, “Toward computing homology from finite approximations,” *Topol. Proc.*, vol. 24, pp. 503–532, 1999.

[80] V. Robins, P. J. Wood, and A. P. Sheppard, “Theory and algorithms for constructing discrete Morse complexes from grayscale digital images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 8, pp. 1646–1658, Aug. 2011.

[81] N. Saul and C. Tralie, “Scikit-TDA: Topological data analysis for python,” 2019. [Online]. Available: https://doi.org/10.5281/zenodo.2533369

[82] N. Shivashankar and V. N. M. Senthilnathan, “Parallel computation of 2D Morse-Smale complexes,” *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 10, pp. 1757–1770, Oct. 2012.

[83] N. Shivashankar and V. Natarajan, “Parallel computation of 3D Morse-Smale complexes,” *Comput. Graph. Forum*, vol. 31, pp. 965–974, 2012.

[84] N. Shivashankar, P. Pranav, V. Natarajan, R. van de Weygaert, E. P. Bos, and S. Rieder, “Felix: A topology based framework for visual exploration of cosmic filaments,” *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 6, pp. 1745–1759, Jun. 2016.

[85] D. Smirnov and D. Morozov, “Triplet merge trees,” in *Proc. Topological Methods Data Anal. Vis.*, 2017, pp. 19–36.

[86] M. Soler, M. Pettitfere, G. Darche, M. Plainchault, B. Conche, and J. Tierny, “Ranking viscous finger simulations to an acquired ground truth with topology-aware matchings,” in *Proc. IEEE 9th Symp. Large Data Anal. Vis.*, 2019, pp. 62–72.

[87] M. Soler, M. Plainchault, B. Conche, and J. Tierny, “Lifted Wasserstein matcher for fast and robust topology tracking,” in *Proc. IEEE 8th Symp. Large Data Anal. Vis.*, 2018, pp. 23–33.

[88] J. Vidal, M. Petitfrere, G. Darche, M. Plainchault, B. Conche, and J. Tierny, “Topologically controlled lossy compression,” in *Proc. IEEE Pacific Vis. Symp.*, 2018, pp. 46–55.

[89] T. Sousbie, “The persistent cosmic web and its filamentary structure: Theory and implementations,” *Monthly Notices Roy. Astronomical Soc.*, vol. 414, pp. 350–383, 2011.

[90] A. Tausz, M. Vejdemo-Johansson, and H. Adams, “JavaPat: A research software package for persistent (Co)homology,” in *Proc. Int. Congr. Math. Softw.*, 2014, pp. 129–136. [Online]. Available: http://appliedtopology.github.io/javaplex/

[91] G. Tausz et al., “giotto-tda: A topological data analysis toolkit for machine learning and data exploration,” *J. Mach. Learn. Res.*, vol. 22, 2021, Art. no. 39.

[92] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux, “The topology ToolKit,” *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 832–842, Jan. 2018. [Online]. Available: https://topology-tool-kit.github.io/

[93] J. Tierny and V. Natarajan, “Generalized topological simplification of scalar fields on surfaces,” *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2005–2013, Dec. 2012.

[94] C. Tralie, N. Saul, and R. Bar-On, “Ripser.py: A lean persistent homology library for python.” *J. Open Source Softw.*, vol. 3, no. 29, Sep. 2018, Art. no. 925, doi: 10.21105/joss.00925.

[95] M. van Kreveld, R. van Oostrom, C. Bajaj, V. Pasucci, and D. Schikore, “Contour trees and small seed sets for isosurface traversal,” in *Proc. Int. Symp. Comput. Geometry*, 1997, pp. 212–220.

[96] J. Vidal, J. Budin, and J. Tierny, “Progressive Wasserstein barycenters of persistence diagrams,” *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 151–161, Jan. 2020.

[97] J. Vidal, P. Guillou, and J. Tierny, “A progressive approach to scalar field topology,” *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 6, pp. 2833–2850, Jun. 2021.

[98] J. Vidal and J. Tierny, “Fast approximation of persistence diagrams with guarantees,” in *Proc. IEEE 11th Symp. Large Data Anal. Vis.*, 2021, pp. 1–11.

[99] H. Wagner, C. Chen, and E. Vaccini, “Efficient computation of persistent homology for cubical data,” in *Proc. Workshop Topol.-Based Methods Data Anal. Vis.*, 2011, pp. 91–106.

[100] G. H. Weber, S. E. Dillard, H. A. Carr, V. Pasucci, and B. Hamann, “Topology-controlled volume rendering,” *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 2, pp. 330–341, Mar./Apr. 2007.

[101] A. J. Zomorodian, “Topology for computing,” in *Algorithms and Theory of Computation Handbook*, M. J. Atallah and M. Blanton, Eds., 2nd ed. Boca Raton, FL, USA: CRC Press, 2010, ch. 3, pp. 82–112.

Pierre Guillou received the graduate and PhD degrees from MINES ParisTech, in 2013 and 2016, respectively. He is a research engineer with Sorbonne Université. His PhD work revolved around parallel image processing algorithms for embedded accelerators. Since 2019, he has been an active contributor to TTK and the author of many modules created for the VESTEC and TORI projects.

Jules Vidal received the engineering degree from ENSTA Paris, in 2018, and the PhD degree from Sorbonne Université, in 2021. He is currently a post-doctoral researcher with Sorbonne Université. He is an active contributor to the Topology ToolKit (TTK), an open source library for topological data analysis. His notable contributions to TTK include the efficient and progressive approximation of distances, barycenters, and clusterings of persistence diagrams.

Julien Tierny received the PhD degree in computer science from the University of Lille, in 2008, and the habilitation degree (HDR) from Sorbonne University, in 2016. He is currently a CNRS research director, affiliated with Sorbonne University. Prior to his CNRS tenure, he held a Fulbright fellowship (U.S. Department of State) and was a post-doctoral researcher with the Scientific Computing and Imaging Institute, University of Utah. His research expertise lies in topological methods for data analysis and visualization. He is the founder and lead developer of the Topology ToolKit (TTK), an open source library for topological data analysis.