Parallel multi-scale reduction of persistent homology filtrations

Rodrigo Mendoza-Smith and Jared Tanner
{mendozasmith,tanner}@maths.ox.ac.uk

The persistent homology pipeline includes the reduction of a, so-called, boundary matrix. We extend the work of [1, 7] where they show how to use dependencies in the boundary matrix to adapt the reduction algorithm presented in [12] in such a way as to reduce its computational cost. Herein we present a number of additional dependencies in the boundary matrices and propose a novel parallel algorithm for the reduction of boundary matrices. In particular, we show: that part of the reduction is immediately apparent, give bounds on the reduction needed for remaining columns, and from these give a framework for which the boundary reduction process can be massively parallelised. Simulations on four synthetic examples show that the computational burden can be conducted in approximately a thousandth the number of iterations needed by traditional methods. Moreover, whereas the traditional boundary reductions reveal barcodes sequentially from a filtration order, this approach gives an alternative method by which barcodes are partly revealed for multiple scales simultaneously and further refined as the algorithm progresses; simulations show that for a Vietoris-Rips filtration with $\sim 10^4$ simplices, an estimate of the essential simplices with 95% precision can be computed in *two iterations* and that the reduction completed to within 1% in about ten iterations of our algorithm as opposed to nearly approximately eight thousand iterations for traditional methods.

1 Introduction

Persistent homology is a technique within topological data analysis, see [12, 13] and references therein, that estimates the topological features of a shape in high-dimensional space from a point-cloud $S \subset \mathbb{R}^d$ sampled from a data manifold $\mathcal{M} \subset \mathbb{R}^d$. The topological information on the shape $\mathcal{M}$ is encoded as a set of *Betti numbers*,

$$\text{Betti}(\mathcal{M}) := \{b_{p,r} : 0 \leq p \leq d, r \in [0, \infty)\} \qquad (1)$$

which geometrically represent the number of p-dimensional holes at scale $r \in [0, \infty)$ in a simplicial complex triangulation of $\mathcal{M}$. Knowledge of the homologies persistent in a dataset can aid interpretation of the data, see for example [3, 4, 5, 6, 14, 22, 18]. In the persistent homology paradigm, the set $\text{Betti}(\mathcal{M})$ is approximated at multiple scales $\{r_1, \dots, r_T\} \subset [0, \infty)$ through a two-step procedure. First, a scale-indexed filtration of simplicial complexes is constructed yielding a simplicial complex K with vertex set S and m simplices. This simplicial complex is represented as a $m \times m$ matrix ∂ defined over the Galois field of two elements $\mathbb{F}_2$ and having $\partial_{i,j} = 1$ iff $\sigma_i \in K$ is a face of $\sigma_j \in K$ with co-dimension 1. The matrix $\partial \in \mathbb{F}_2^{m \times m}$, together with the scale r_ℓ at which each simplex $\sigma_j \in K$ is added to the filtration, encode the necessary information to estimate $\text{Betti}(\mathcal{M})$ via persistent homology. The homologies in the data are then revealed in the second, and final, step of reducing the boundary matrix ∂. Reduction algorithms for persistent homology are so named as they can essentially be viewed as acting on each column of the boundary matrix to minimise the maximum index of its nonzeros while maintaining that the column span of the first j columns of ∂ remains unchanged for all j. That is, following the notation of [7], reduction algorithms entrywise minimise

$$\text{low}_\partial(j) = \begin{cases} \max\{i \in [m] : \partial_{i,j} = 1\} & \text{if } \partial_j \neq 0 \\ 0 & \text{if } \partial_j = 0, \end{cases} \qquad (2)$$

subject to the aforementioned span constraint; we denote the minimum of $\text{low}_\partial(j)$ by $\text{low}^*(j)$. Nonzero values of $\text{low}^*(j)$ reveal a homology persisting from $\sigma_{\text{low}^*(j)}$ to σ_j. As $\text{low}^*(j)$ is a property of the simplicial complex K [11] we omit the explicit reference to the boundary matrix in its notation; moreover, we denote by low_∂ and low^* the vectors of their values in $\{0, 1, \dots, m\}$. Sec. 2.1 and 2.2 provides further details on the construction of the simplicial complex filtration, the persistent homology pipeline, and their connections with the underlying data manifold as they pertain to our main results.

The focus of this manuscript is extending the work [1, 7] where they show how to use dependencies in the boundary matrix to adapt the first boundary matrix reduction algorithm [12], restated in Alg. 1, in such a way as to reduce its computational cost.

Algorithm 1: Standard reduction [12]

```
Data: ∂ ∈ F_2^{m×m}
Result: low* ∈ Z_{m+1}^m
for j ∈ [m] do
    while ∃ j_0 < j : low_∂(j_0) = low_∂(j) do
        ∂_j ← ∂_j + ∂_{j_0};
    end
end
low* ← low_∂;
```

Alg. 1 performs the reduction by sequentially minimising $\text{low}_\partial(j)$ by adding columns $j_0 < j$ for which $\text{low}_\partial(j_0) = \text{low}_\partial(j)$ until either there is no such column j_0 or column j has been set to zero. The computational overhead in Alg. 1 is in computing the left-to-right column operations, and has a worst-case complexity of $\mathcal{O}(m^3)$ which is achieved by an example simplicial complex in [17]. Previous advances in reduction algorithms primarily focus on decreasing the computational cost by exploiting structure in ∂ to reveal some entries in ∂ can be set to zero [1, 7] or by parallelising the reduction by dividing the boundary matrix into blocks and partially reducing each block [1, 2]; further details of these approaches are given in Sec. 2.4. The aforementioned approaches show substantially improved average empirical operation count as compared to Alg. 1.

Our main contribution is by noting both bounds on the values of $\text{low}^*(j)$, which are empirically observed to typically identify a large fraction of the $\text{low}^*(j)$, and moreover presenting a framework by which the known $\text{low}^*(j)$ can be used to to reduce ∂ in a highly parallel fashion.

In addition, the bounds on $\text{low}^*(j)$ suggest priorities by which columns might be reduced. The algorithm is designed to work with some of the previous speedups as suggested in [1]. Moreover, as is discussed in Sec. 3, our parallelisation strategy induces an iterative non-local refinement procedure on low_∂ and makes the algorithm fit for early stopping, see Secs. 4.2 and 4.3. An example of the a resulting reduction algorithm is given in Alg. 2 whose details are explained further in Sec. 3 and numerical experiments for its application are shown in Sec. 4. Additional strategies to speed up the algorithm are given in Secs. 3.2 and 3.3, while Sec. 3.4 suggests further extensions for computer environments with substantially fewer processors than m.

Algorithm 2: Parallel multi-scale reduction

```
Data: ∂ ∈ 𝔽₂^{m×m}; MAX_ITER ∈ [m]
Result: low* ∈ ℤ^m_{m+1}
// Phase 0: Initialisation
Build β_j according to (9);
Pivots ← {j ∈ [m] : β_j = low_∂(j) > 0};
for j ∈ Pivots do
    ∂_{low_∂(j)} ← 0;
end
iter ← 0;
while low_∂ ≠ low* or iter ≤ MAX_ITER do
    // Phase I: Local injections
    for d ∈ [dim(K)] do
        maxCollision ← 0;
        for j ∈ {ℓ ∈ K_d : low_∂(ℓ) > 0} \ Pivots do
            if low_∂(j) > maxCollision then
                if low_∂(j) ∉ low_∂([j − 1]) then
                    Pivots ← Pivots ∪ {j};
                else
                    maxCollision ← low_∂(j);
                end
            end
        end
    end
    // Phase II: Column reduction
    for j_0 ∈ Pivots do
        N(j_0) ← {ℓ ∈ [m] \ [j_0] : low_∂(ℓ) = low_∂(j_0)};
        for j ∈ N(j_0) do
            ∂_j ← ∂_j + ∂_{j_0};
            if low_∂(j) = β_j then
                Pivots ← Pivots ∪ {j};
                ∂_{low_∂(j)} ← 0;
            end
        end
    end
    // Increase iteration
    iter ← iter +1;
end
low* ← low_∂;
```

2 Background

In this section we give an overview of the pipeline for computing persistent homology which includes a short introduction to the topological and algebraic results on which our results are based, and a review of the prior art in boundary matrix reduction algorithms. In what follows we adopt the following notation. If S is a set, we let 2^S denote the power set of S and $|S|$ be the cardinality of S. We also borrow notation from Combinatorics and let $[m] := \{1, \dots, m\}$ for $m \in \mathbb{N}$. We will use $\mathbb{Z}_{m+1}$ as a shorthand for $[m] \cup \{0\}$. For a function $f : A \to B$, we let $f(A) = \{f(a) \in B : a \in A\}$ and $f(\emptyset) = \emptyset$. If $\partial \in \mathbb{F}_2^{m \times m}$ is a matrix, we let $\partial_j \in \mathbb{F}_2^m$ be its j-th column and $\partial_{i,\cdot}$ be its i-th row. The *support* of an m-dimensional vector v is defined as $\text{supp}(v) = \{i \in [m] : v_i \neq 0\}$. If $\partial \in \mathbb{F}_2^{m \times m}$ is a boundary matrix, we let $\text{nnz}(\partial)$ or the number of nonzeros entries in ∂. We reserve the notation $\mathbb{1}\{\cdot\}$ for indicator functions that return 1 if the argument is true and 0 otherwise. If $\mathbb{R}^d$ is the d-dimensional Euclidean space with p-norm $\|\cdot\|_p$ and $S \subset \mathbb{R}^d$, we let $\text{diam}(S) := \max_{x,y \in S} \|x - y\|$ be the *diameter* of S. Finally, for any $c \in \mathbb{R}^d$ and $r \geq 0$, we let $\mathcal{B}_r(c) := \left\{x \in \mathbb{R}^d : \|x - c\|_p \leq r\right\}$ be the d-dimensional p-ball of radius r centred at c.

2.1 Simplicial homology

If S is a finite set and $\alpha \subset \sigma \subset S$, then σ is a *simplex* of S and α is a *face* of σ. A set of simplices $K \subset 2^S$ is a *simplicial complex* if $\sigma \in K$ implies that every face of σ is also in K. The dimension of the simplex is defined as $\dim(\sigma) = |\sigma| - 1$. The dimension of a simplicial complex K will be defined as $\dim(K) = \max\{\dim(\sigma) : \sigma \in K\}$. It will be convenient to assume that the simplices in K are indexed, so that $K = \left\{\sigma_1, \dots, \sigma_{|K|}\right\}$. In this case, the set of *p-simplices* is the set of simplices in K which are indexed by

$$K_p = \{j \in [|K|] : \dim(\sigma_j) = p\}. \tag{3}$$

The boundary of a simplex σ is its set of faces of co-dimension one; symbolically we denote this by $\text{bd}(\sigma) = \{\alpha : \dim(\alpha) = \dim(\sigma) - 1\}$.

In the subsequent paragraphs we describe, for completeness, the algebraic structure of simplicial complexes and how it gives rise to the homology groups; omitting this paragraph does not limit one's ability to understand the new algorithms proposed. The set of *p-chains* $\mathsf{C}_p(K) := 2^{K_p}$ equipped with the symmetric difference operation is an abelian group with neutral element $\emptyset$. For each $p \in \mathbb{N}$, the group $\mathsf{C}_p(K)$ is related to $\mathsf{C}_{p-1}(K)$ by a *boundary map* $d_p : \mathsf{C}_p \to \mathsf{C}_{p-1}$ defined by $d_p(c) = \sum_{\sigma \in c} \text{bd}(\sigma)$. The kernel and image of d_p are geometrically meaningful. Elements in $\mathsf{Z}_p = \text{Ker}\, d_p$ are called *p-cycles*, while elements in $\mathsf{B}_p = \text{Im}\, d_{p+1}$ are called *p-boundaries*. Moreover, for all p and for all $c \in \mathsf{C}_p$, $d_{p-1} \circ d_p(c) = \emptyset$, so $\mathsf{B}_p \subset \mathsf{Z}_p \subset \mathsf{C}_p$, see [11]. The quotient space $\mathsf{H}_p = \mathsf{Z}_p / \mathsf{B}_p$ is called the *p-th homology group* and its elements are called *homology classes*. The p-th *Betti number* is defined

as $b_p = \operatorname{rank} \mathsf{H}_p$ and counts the number of p-dimensional holes of the simplicial complex K.

2.2 Construction of the simplicial complex

In persistent homology, a simplicial complex K is built from a point-cloud $S \subset \mathbb{R}^d$ sampled from a data manifold $\mathcal{M} \subset \mathbb{R}^d$. Given S, the goal of persistent homology is to estimate the relevant homological features of $\mathcal{M}$ at all scales $r \in [0, \infty)$. To do so, a simplicial complex triangulation is computed from S at all scales $r \in \{r_1, \ldots, r_T\} \subset [0, \infty)$ by letting $K = \emptyset$ and adding a simplex $\sigma \subset S$ to K whenever the points in σ are sufficiently close to each other. The notion of closeness is implied by the relevant scale parameter, and is assessed for each scale r via a monotonic function $f_r : 2^S \to \{0, 1\}$ inducing a filtration

$$\mathbf{M}_0 \subset \mathbf{M}_1 \subset \cdots \subset \mathbf{M}_T = K \tag{4}$$

of simplicial complexes $\mathbf{M}_i = \left\{\sigma \in 2^S : f_{r_i}(\sigma) = 1\right\}$. For example, the function

$$f_r(\sigma) = \mathbb{1}\left\{\bigcap_{x_0 \in \sigma} \mathcal{B}_r(x_0)\right\},$$

generates the *Čech complex* filtration, while the function

$$f_r(\sigma) = \mathbb{1}\left\{\operatorname{diam}(\sigma) \leq 2r\right\}$$

generates the *Vietoris-Rips complex* filtration. We shall assume that the filtration (4) has m elements and that the largest set in the filtration is a simplicial complex with simplices given by

$$K = \{\sigma_1, \ldots, \sigma_m\}.$$

It is further assumed that the simplices in K are indexed according to a *compatible ordering*, meaning that the simplices in $\mathbf{M}_\ell$ always precede the ones in $K \setminus \mathbf{M}_\ell$, and that the faces of any given simplex always precede the simplex. Persistent homology tracks how the homology of the filtration changes at each scale r_t or, equivalently, as new simplices are added to the filtration. Indeed, when adding simplex σ_i at scale r_t, the homology of $\mathbf{M}_t$ can change in one of two possible ways [11].

1. A class of dimension $\dim(\sigma_i)$ is created. In this case, σ_i is a *positive* simplex.
2. A class of dimension $\dim(\sigma_i) - 1$ is destroyed. In this case, σ_i is a *negative* simplex.

If σ_j is a negative simplex, then it destroys the class created by a positive simplex σ_i with $i = \operatorname{low}^*(j) < j$, see Lemma 1. This observation induces a natural pairing (σ_i, σ_j) between a negative simplex σ_j and the positive simplex σ_i it destroys. Moreover, it allows us to quantify the lifetime of a particular homology class in the filtration via its *homology persistence* which is the difference between r_t for σ_j and σ_i.

When r_T is sufficiently small, we might find that some simplices are never destroyed; these simplices represent the homology classes that are *persistent* in the filtration up to scale r_T, and we called them *essential*. The persistence pairs are computed by representing the filtration (4) as a boundary matrix $\partial \in \mathbb{F}_2^{m \times m}$ and applying the *reduction algorithm* [12] to it. We discuss this process in Sec. 2.3

2.3 Boundary matrix reduction

If $K = \{\sigma_1, \ldots, \sigma_m\}$ is a simplicial complex constructed as in the previous Sec. 2.2, it can be represented with a boundary matrix $\partial \in \mathbb{F}_2^{m \times m}$ defined by

$$\partial_{i,j} = \begin{cases} 1 & \sigma_i \text{ is a face of } \sigma_j \text{ of co-dimension 1} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The boundary matrix ∂ is sparse, binary, and upper-triangular and has associated a function $\operatorname{low}_\partial : [m] \to \mathbb{Z}_{m+1}$ defined as in (2). The matrix ∂ will be said to be *reduced* when $\operatorname{low}_\partial \in \mathbb{Z}_{m+1}^m$ is an injection over its support, *i.e.* when $\operatorname{low}_\partial(j_1) = \operatorname{low}_\partial(j_2) > 0$ implies that $j_1 = j_2$. We use the notation ∂^* to denote a matrix ∂ with injective $\operatorname{low}_\partial$ and remark that even though ∂ can have several reductions, the injection is a property of the complex K and does not depend on any particular reduction ∂^*, see [11, p.183]. When there is no risk of confusion, we let low^* be the injection of the boundary matrix under consideration.
The vector low^* reveals information about the pairings by virtue of the following Lemma.

Lemma 1 (Pairing [12])**.** *If σ_j is a negative simplex, then $\sigma_{\operatorname{low}^*(j)}$ is a positive simplex.*

Hence, $\operatorname{low}^*(\cdot)$ partitions the simplices in K as,

$$\begin{aligned} \mathsf{Pos} &= \{j \in [m] : \operatorname{low}^*(j) = 0\} \\ \mathsf{Neg} &= \{j \in [m] : \operatorname{low}^*(j) > 0\} \\ \mathsf{Ess} &= \{j \in \mathsf{Pos} : \nexists\, k \text{ s.t. } \operatorname{low}^*(k) = j\} \end{aligned}$$

so $\mathsf{Pos} \cap \mathsf{Neg} = \emptyset$ and $\mathsf{Ess} \subset \mathsf{Pos}$. It will also be convenient to define the set Paired as the union of the set of negative and associated positive simplices

$$\mathsf{Paired} = \left\{j \in [m] : \operatorname{low}^*(j) \in [m]\right\} \cup \left\{\operatorname{low}^*(j) \in [m] : j \in [m]\right\} \tag{6}$$

The technology used to reduce ∂ is known as the *reduction algorithm* (Alg. 1) and was first presented in [12]. The convergence guarantees of Alg. 1 are given in Theorem 2.

Theorem 2 (Convergence of Alg. 1 [12])**.** *Alg. 1 converges for any boundary matrix $\partial \in \mathbb{F}_2^{m \times m}$ in at most $\mathcal{O}(m^3)$ operations.*

We survey the prior art in boundary matrix reduction algorithms in the subsequent Sec. 2.4.

2.4 Boundary matrix algorithm prior art

Let K be the simplicial complex corresponding to a filtration over a grid $\{r_1, \dots, r_T\}$ and point-cloud $S \subset \mathbb{R}^d$. The number of simplices in K is of order $\Omega(2^{|S|})$ as $r_T \to \mathrm{diam}(S)$, which turns the process unfeasible even for moderately large point-clouds S and scales r_T. Hence, though Alg. 1 terminates in $\mathcal{O}(m^3)$ steps, we may find that $m = \mathcal{O}(2^{|S|})$ making the reduction unfeasible. The main computational overhead when reducing a boundary matrix is in performing left-to-right column operations, so many reduction algorithms have implemented strategies that cut the number of column operations required to fully reduce the matrix. One such strategy was given in [7, 1] by reducing columns in blocks of decreasing dimension and observing that, by Lemma 1, column $\mathrm{low}^*(j)$ can be set to zero whenever column j is reduced. As $\dim(\sigma_{\mathrm{low}^*(j)})$ is of one lower dimension that $\dim(\sigma_j)$, clearing results in all positive non-essential columns of dimension less than $\dim(K)$ being set to zero without zeroing them through column additions. Setting a positive column to zero when its corresponding negative pair has been found is often called *clearing*, so we adopt this terminology. Alg. 2 heavily relies on this clearing strategy so we sketch the pseudocode of [7] in Alg. 3.

Algorithm 3: Standard reduction with a twist [7]

```
Data: ∂ ∈ F_2^{m×m}
Result: low* ∈ Z_m^{m+1}
for d ∈ {dim K, dim K − 1, ..., 1} do
    for j ∈ K_d do
        while ∃ j_0 < j : low_∂(j_0) = low_∂(j) do
            ∂_j ← ∂_j + ∂_{j_0};
        end
        if ∂_j ≠ 0 then
            ∂_{low_∂(j)} ← 0;
        end
    end
end
low* ← low_∂;
```

Another strategy to reduce the complexity of Alg. 1 is called *compression* [1] and consists in deriving analytical guarantees to nullify nonzeros in the boundary matrix without affecting the pairing of the simplices. This has the objective of saving arithmetic operations and hence reducing the flop count. However for very large values of m, which are typical for large filtration values where m grows exponentially with $|S|$, it is necessary to also parallelise these approaches in order to be scalable. The idea of distributing the workload of the reduction algorithm has already been explored in [1, 2] where the matrix is partitioned into b blocks of contiguous columns to be independently reduced in a shared-memory or distributed system. The numerical simulations in [2] show that these strategies can indeed bring substantial speed-ups when implemented on a cluster, but also requires the provision of the parameter b as well as a number of design choices for a practical implementation.
Apart from parallelisation, a number of other approaches have been proposed. For instance, [16] adapted the Coppersmith-Winograd algorithm [8] to guarantee reduction in time $\mathcal{O}(m^{2.3755})$. A set of sequential algorithms that exploits duality of vector spaces was given in [9, 10], but as pointed out in [19] is only known to give speed-ups when applied to Vietoris-Rips complexes. Finally, [20] projects ∂ to a low-dimensional space while controlling the error between the resulting barcodes, but doing so in practice can be as costly as reducing the matrix in its ambient space.

3 Main contributions

In this section we describe the theory behind the parallelisation strategy of Alg. 2. The main algorithmic innovation is a strategy to efficiently distribute the workload of Alg. 1 over $\mathcal{O}(m)$ processors to progressively entrywise minimize $\mathrm{low}_\partial \in \mathbb{Z}_{m+1}^m$. Additionally, Alg. 2 is designed to take advantage of some structural patterns in the boundary matrix in order to minimise the total number of left-to-right column operations. The core observation is that, by simple inspection, the rows of the boundary matrix reveal a subset of nonzero entries in ∂ which cannot be modified in the reduction process, and consequently identify nonzero lower bounds on $\mathrm{low}^*(j)$ for a subset of $j \in [m]$. This observation is captured by Definition 3 and Definition 4.

Definition 3 ($\mathrm{left}(\cdot)$)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be a boundary matrix. The* left *function is defined as*

$$\mathrm{left}(i) = \begin{cases} \min\{j \in [m] : \partial_{i,j} = 1\} & \partial_{i,\cdot} \neq 0 \\ 0 & \partial_{i,\cdot} = 0 \end{cases} \tag{7}$$

Definition 4 (β_j)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be a boundary matrix and*

$$\mathcal{L}_j = \{i \in [m] : \mathrm{left}(i) = j\}. \tag{8}$$

Then, for $j \in [m]$ we let

$$\beta_j = \begin{cases} \max \mathcal{L}_j & \mathcal{L}_j \neq \emptyset \\ 0 & \mathcal{L}_j = \emptyset \end{cases} \tag{9}$$

The vector $\beta \in \mathbb{Z}_{m+1}^m$ carries a great deal of information about the nature of each column in the boundary matrix. Some of its properties are explored in Theorem 5.

Theorem 5 (Properties of β_j)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be a boundary matrix and β defined as in* (9)*. Then, the following statements hold.*

P.1 $\beta_j = 0 \Leftrightarrow \nexists i \in [m]\ s.t.\ \mathrm{left}(i) = j$

P.2 β_j *is invariant to left-to-right column operations.*

P.3 $\{j \in [m] : \beta_j > 0\} \subset \mathsf{Neg}$.

P.4 $\mathsf{Pos} \subset \{j \in [m] : \beta_j = 0\}$.

P.5 $\beta_j \leq \text{low}^*(j) \leq \text{low}_\partial(j)$ *for all* $j \in [m]$.

P.6 β *can be computed in time* $\mathcal{O}(\text{nnz}(\partial))$.

Proof.

P.1 This follows directly from the definition of β_j.

P.2 Let $i \in [m]$ be a row of ∂ and $j = \text{left}(i)$. Reduction of ∂_j is achieved by adding to ∂_j columns ∂_ℓ for $\ell < j$ By Definition 8 $\partial_{i,\ell} = 0$ for $\ell < j$ and hence $\partial_{i,j} = 1$ throughout the reduction. Consequently the set $\text{left}(i)$ and values β_j are fixed in the reduction procedure.

P.3 If $\beta_j > 0$, then exists $i \in [m]$ such that $\text{left}(i) = j > 0$. Then, $\partial_{i,j} = 1$ and there does not exist ∂_ℓ with $\ell < j$ such that $\partial_{i,\ell} = 1$. Hence $\partial_{i,j}$ can not be set to zero in the reduction, so $j \in \mathsf{Neg}$.

P.4 This result follows by taking complements on the result P.3.

P.5 By P.2 the set $\mathcal{L}_j$ in 8 are nonzero entries in ∂_j which cannot be modified in the reduction procedure and consequently $\text{low}^*(j) \geq \beta_j$.

$$\text{low}^*(j) \geq \max\{i \in \text{supp}(\partial_j) : \text{left}(i) = j\} = \beta_j$$

P.6 Note that when creating the boundary matrix, β can be constructed with the following procedure,

Algorithm 4: Building β

```
Data: Simplicial complex K = {σ_1, ..., σ_m}
Result: β_j for j ∈ [m]
for σ_j ∈ K do
    β_j ← 0;
    for σ_i ∈ bd(σ_j) do
        if σ_i has not been visited then
            β_j ← max(β_j, i);
            Mark σ_i as visited;
        end
    end
end
```

Alg. 4 finishes in time proportional to $\sum_j |\,\text{bd}(\sigma_j)|$, so it has complexity $\mathcal{O}\left(\text{nnz}(\partial)\right)$.

□

The value of Theorem 5 is most immediately apparent in P.4 and in particular when $\text{low}_\partial(j) = \beta_j$ in which case $\text{low}^*(j)$ is determined. Our numerical experiments in Sec. 4 show empirically that a large number of columns can usually be identified as already being reduced without need of any column operations. Moreover, having access to a large number of reduced columns allows a massive parallelisation as described in Sec. 3.1 and illustrated in Sec. 4. While the calculation of β adds an initial computation to Alg. 2, it can be computed at the matrix-reading cost of $\mathcal{O}(\text{nnz}(\partial))$, so it can often be obtained as a by-product of creating the filtration without penalising the asymptotic computational complexity of constructing the matrix.

Additionally, the vector β gives a sufficient condition for a column to be in Neg. Knowledge that an unreduced column is necessarily negative can be used in numerous ways. For example, as discussed in sections 3.2, 3.3, and 3.4 this is useful to produce more reliable estimations of low^* in the case of early-stopping. Additionally, knowledge that a column is necessarily negative enhances a notion of clearing as discussed in Sec. 3.2.

Apart from Theorem 5, Alg. 2 makes use of the observation that since low_∂ converges to an injection low^*, one can often inspect the sparsity pattern of ∂ to identify regions of $[m]$ where low_∂ is locally an injection. This local injection property is the basis of the following Theorems 6 and 7, which give sufficient conditions to identify sets $T \subset [m]$ such that $\text{low}_\partial(j) = \text{low}^*(j)$ for all $j \in T$. In the remainder of this section it will be convenient to extend some of our prior notation as follows: for a set $T \subset [m]$, we let $\text{low}_\partial(T) = \{\text{low}_\partial(j) : j \in T\}$ and $|\,\text{low}_\partial(T)|$ be its cardinality. Informally, the argument in Theorem 6 is that, for each dimension d, any subset of contiguous columns $T \subset K_d \setminus \{j \in K_d : \text{low}_\partial(j) = 0\}$ with $\min T = \min K_d$ is already reduced if $\text{low}_\partial : T \to [m]$ is an injection or, equivalently, if $|\,\text{low}_\partial(T)| = |T|$.

Theorem 6 (Local injection I)**.** *Let* $\partial \in \mathbb{F}_2^{m\times m}$ *be the boundary matrix of a simplicial complex* K *and let* $d \in [\dim(K)]$ *and* K_d *defined as in* (3)*. Let*

$$T := K_d \setminus \{j \in K_d : \text{low}_\partial(j) = 0\} \neq \emptyset$$

be such that $T = \left\{j_1, \ldots, j_{|T|}\right\}$ *for* $j_1 < \cdots < j_{|T|}$ *and for* $\ell \leq |T|$*, let* $T^\ell = \{j_1, \ldots, j_\ell\}$*. If*

$$\left|\text{low}_\partial\left(T^\ell\right)\right| = \ell, \tag{10}$$

then $\text{low}_\partial(j) = \text{low}^*(j)$ *for all* $j \in T^\ell$.

Proof. Let T be defined as in Theorem 6. If $\ell = 1$, then there ∂_{j_1} is the first column of dimension d that appears in the filtration, so there is no column $j \in [j_1]$ that can reduce it. Hence, suppose that $\ell > 1$ and let $j_t \in T^\ell$ be such that $\text{low}_\partial(j_t) \neq \text{low}^*(j_t)$. Then there is $j < j_t$ such that $\text{low}_\partial(j) = \text{low}_\partial(j_t)$. This implies that $j \in K_d \cap [j_t]$ and that $\text{low}_\partial(j) > 0$, so $j \in T_\ell$. Hence, $|T_\ell| = \ell$, but $|\,\text{low}_\partial(T_\ell)| < \ell$, which contradicts (10). □

Theorem 7 addresses a generalisation of Theorem 6 and argues that for each dimension d, any subset of contiguous columns $T \subset K_d \setminus \{j \in K_d : \text{low}_\partial(j) = 0\}$ is already reduced if $|\,\text{low}_\partial(T)| = |T|$ and there is no column $j \in K_d \setminus T$ with $j < \min T$ such that $\text{low}_\partial(j) \in \text{low}_\partial(T)$.

Theorem 7 (Local injection II)**.** *Let* $\partial \in \mathbb{F}_2^{m\times m}$ *be the boundary matrix of a simplicial complex* K *and let* $d \in$

$[\dim(K)]$. *Define T and T^ℓ as in Theorem 6. Assume that for $k > 1$ and $k \leq \ell$ it holds that*

$$\min \mathrm{low}_\partial(T^\ell \setminus T^{k-1}) > \max \mathrm{low}_\partial(T^{k-1}) \tag{11}$$

and

$$|\mathrm{low}_\partial(\{j_k, \ldots, j_\ell\})| = \ell - k + 1. \tag{12}$$

Then, $\mathrm{low}_\partial(j) = \mathrm{low}^*(j)$ *for all* $j \in \{j_k, \ldots, j_\ell\}$.

Proof. Let T and T^ℓ be as in Theorem 7 so that $T^\ell \setminus T^{k-1} = \{j_k, \ldots, j_\ell\}$. Since $\ell > 0$ and $k > 1$, then $\ell - k + 1 > 0$. If $k = 2$, then $j_2 \in \mathrm{low}_\partial(T^\ell \setminus T^1)$ and invoking (11),

$$\mathrm{low}_\partial(j_2) \geq \min \mathrm{low}_\partial(T^\ell \setminus T^1) > \max \mathrm{low}_\partial(T^1) = \mathrm{low}_\partial(j_1),$$

so the result follows by invoking Theorem 6 with $\ell = 2$. Arguing as in Theorem 6, suppose that $k > 2$ and let $j_t \in \{j_k, \ldots, j_\ell\}$ be such that $\mathrm{low}_\partial(j_t) \neq \mathrm{low}^*(j_t)$. Then, there is $j < j_t$ such that $\mathrm{low}_\partial(j) = \mathrm{low}_\partial(j_t)$. Given that $j_k \leq j < j_t$ and by (11) then $j \in T^\ell \setminus T^{k-1}$. Hence, $|\mathrm{low}_\partial(T_\ell \setminus T^{k-1})| < \ell - k + 1$, which contradicts (12). □

We now describe how the results presented so far in this section interact in Alg. 2.

3.1 Parallel multi-scale reduction

So far, Theorems 5 -7 identify a subset of the columns j in $\partial \in \mathbb{F}_2^{m \times m}$ for which $\mathrm{low}^*(j)$ is known without need for performing column additions. In order to complete the reduction of ∂, that is identify low^*, it is necessary to perform column additions analogous to those in Alg. 1; however,unlike the sequential ordering in Alg. 1, we will propose massive parallelisation of the column additions. Towards this we define the set of columns $j \in [m]$ for which their $\mathrm{low}^*(j)$ is known at any given iteration of a reduction algorithm as the Pivots:

$$\mathsf{Pivots} = \left\{ j \in [m] : \mathrm{low}^*(j) \in [m] \text{ has been identified.} \right\}$$

Moreover, we define the set of column with which a pivot could be added to in order to reduce their low_∂ as the *neighbours* of column j:

$$\mathcal{N}(j) = \left\{ \ell > j : \mathrm{low}_\partial(\ell) = \mathrm{low}^*(j) \right\}. \tag{13}$$

Note that as low^* are an injection, if $j_1, j_2 \in \mathsf{Pivots}$ and $j_1 \neq j_2$, then $\mathcal{N}(j_1) \cap \mathcal{N}(j_2) = \emptyset$ so

$$\left(\bigcup_{j \in \mathsf{Pivots}} \mathcal{N}(j) \right) \bigcup \mathsf{Pivots} \subset [m] \tag{14}$$

is a partition and each column in $\mathcal{N}(j)$ for $j \in \mathsf{Pivots}$ can have their low reduced independently by adding column j to columns $\ell \in \mathcal{N}(j)$. In sequential algorithms like Algs. 1 and 3, the set $\mathcal{N}(j)$ is constructed only after the block $[j-1] \cap K_{\dim(\sigma_j)}$ has been fully reduced. In contrast, Alg. 2 starts by inspecting the sparsity pattern of ∂ to identify a large set of *seed* Pivots that are then used to build (13) for each $j \in \mathsf{Pivots}$. Given that (14) is a partition, each set $\mathcal{N}(j)$ is partially reduced independently, typically in parallel and new columns are appended to Pivots whenever their update is such that their $\mathrm{low}^*(\ell)$ is identified by either by Theorems 5 - 7. We elaborate on these stages of Alg. 2, and its convergence, in the remainder of this section.

3.1.1 Phase 0: Initialising Pivots

The first step of Alg. 2 is to compute the vector β, which can be done at cost $\mathcal{O}(\mathrm{nnz}(\partial))$ by Theorem 5. This vector is then coupled with low_∂ to build the set of initial pivots

$$\mathsf{Pivots} \leftarrow \{ j \in [m] : \beta_j = \mathrm{low}_\partial(j) \},$$

which by Theorem 5, contains indices for columns that are already reduced, e.g. $\mathrm{low}^*(j)$ is known for $j \in \mathsf{Pivots}$. The clearing strategy from Lemma 1 is applied to each of these columns to zero out their corresponding positive pairs. The resulting set of Pivots is then passed on to the main iteration.

3.1.2 Phase I: Finding local injections

In the first phase of the main iteration, the following Alg. 5 is applied to each dimension $d \in [\dim(K)]$.

Algorithm 5: Finding local injections

```
Data: ∂ ∈ F_2^{m×m}; Pivots ⊂ Neg
Result: Pivots′ ⊂ Neg s.t. Pivots ⊂ Pivots′
maxCollision ← 0;
for j ∈ {ℓ ∈ K_d : low_∂(ℓ) > 0} \ Pivots do
    if low_∂(j) > maxCollision then
        if low_∂(j) ∉ low_∂([j − 1]) then
            Pivots ← Pivots ∪ {j};
        else
            maxCollision ← low_∂(j);
        end
    end
end
```

Alg. 5 starts by initialising maxCollision $\leftarrow 0$ and walking through the non-zero columns in K_d that have not been identified as pivots. The idea is to identify regions of K_d where Theorems 6 and 7 can be applied to guarantee that a local injection exists. At each j, the variable maxCollision tracks the largest $i \in \mathrm{low}_\partial([j-1]) \cap \mathrm{low}_\partial(K_d)$ such that $i = \mathrm{low}_\partial(j_1) = \mathrm{low}_\partial(j_2)$ for $j_1 \neq j_2$. In terms of Theorem 7, maxCollision plays the rôle of $\max \mathrm{low}_\partial(T^{j-1})$ in (11), since if $\mathrm{low}_\partial(j) > \text{maxCollision}$ and $\mathrm{low}_\partial(j) \notin \mathrm{low}_\partial([j-1])$, then there is no column $\ell < j$ that can have $\mathrm{low}_\partial(\ell) = \mathrm{low}_\partial(j)$ and consequently $\mathrm{low}_\partial(j)$ cannot be further reduced and therefore its $\mathrm{low}^*(j)$ is known.

3.1.3 Main iteration II: Parallel column reduction

Once the first phase has been completed, the sets of $\mathcal{N}(j)$ are computed for each $j \in \mathsf{Pivots}$. Each column in $\bigcap_{j\in\mathsf{Pivots}}\mathcal{N}(j)$ has their associated pivot added to them; that is, $\partial_\ell \leftarrow \partial_\ell + \partial_j$ is carried out for each $\ell \in \mathcal{N}(j)$ for $j \in \mathsf{Pivots}$. Alg. 2 then marks the column ℓ as reduced if $\mathrm{low}_\partial(\ell) = \beta_\ell$,
ell is added to the set of Pivots, and the clearing strategy is called to set the associated positive column $\partial_{\mathrm{low}^*(\ell)}$ to zero.

3.1.4 Convergence

The convergence of Alg. 2 is a consequence of the interplay between Phases I and II of the main iteration. We describe this mechanism in the following theorem.

Theorem 8 (Convergence of Alg. 2)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be the boundary matrix of a simplicial complex K. Then, Alg. 2 converges to* low^*.

Proof. Let $\mathsf{Pivots}^\ell \subset [m]$ be the set of indices of columns that are known to be already reduced at the start of iteration ℓ, and let $\mathrm{low}_\partial{}^\ell$ be the corresponding estimate of low^*. We show that Alg. 1 converges by showing that if $\mathrm{low}^* \neq \mathrm{low}_\partial{}^\ell$, then either

$$|\,\mathsf{Pivots}^{\ell+1}\,| \geq |\,\mathsf{Pivots}^{\ell}\,| + 1 \tag{15}$$

or,

$$\|\,\mathrm{low}_\partial{}^{\ell+1} - \mathrm{low}^*\,\| < \|\,\mathrm{low}_\partial{}^{\ell} - \mathrm{low}^*\,\|. \tag{16}$$

Note that $|\,\mathsf{Pivots}^0\,| \geq \dim(K) \geq 1$ since at Phase 0 at least the first column of each dimension must be identified as a pivot.
Suppose that at the start of iteration $\ell \geq 0$ the boundary matrix ∂ is still not reduced so that $\mathrm{low}_\partial{}^\ell \neq \mathrm{low}^*$. If a column is identified as reduced at the end of Phase I, then it is marked as a pivot so $|\,\mathsf{Pivots}^{\ell+1}\,| \geq |\,\mathsf{Pivots}^{\ell}\,|+1$ and we are done. Hence, assume that no new pivots are identified. In this case, for each $j \in \mathsf{Pivots}^\ell$ we construct the set $\mathcal{N}(j)$ as in (13).
If $\exists j_0 \in \mathsf{Pivots}^\ell$ such that $|\mathcal{N}(j_0)| \geq 1$, then Alg. 2 enters Phase II to reduce columns in $\mathcal{N}(j_0)$. In this case, (16) holds and we are done. Otherwise, suppose $|\mathcal{N}(j_0)| = 0$ for all $j \in \mathsf{Pivots}^\ell$. In this case, since the matrix is not reduced, the set

$$\mathcal{C} = \{j \in [m] : |\mathcal{N}(j)| > 0\}$$

is not empty. Let,

$$j_{\min} = \min \mathcal{C} \tag{17}$$

and let $d = \dim(\sigma_{j_{\min}})$. There are two possible cases,

1. $j_{\min} = \min K_d$: In this case, the column must have been identified as a pivot at Phase 0, which is a case previously excluded.
2. $j_{\min} > \min K_d$: Let

$$T = K_d \setminus \{j \in K_d : \mathrm{low}_\partial(j) = 0\}.$$

In this case, it must hold that $|\,\mathrm{low}_\partial([j_{\min}] \cap T)| = |[j_{\min}] \cap T|$ since otherwise, there would exist an $\ell \in [j_{\min} - 1] \cap T$ such that $\mathrm{low}_\partial(\ell) = \mathrm{low}_\partial(j_{\min})$ implying that $|\mathcal{N}(\ell)| > 0$ and thus contradicting the minimality of $j_{\min}$. Hence, by Theorem 6, $\mathrm{low}_\partial([j_{\min}] \cap T) = |[j_{\min}] \cap T|$, so $j_{\min}$ must have been identified at Phase I of the iteration. This is again, a contradiction.

Therefore, it is impossible to have $|\mathcal{N}(j_0)| = 0$ for all $j_0 \in \mathsf{Pivots}^\ell$ if no pivots were identified at Phase I of the iteration. Hence, as long as $\mathrm{low}_\partial{}^\ell \neq \mathrm{low}^*$ either (15) or (16) hold, so at each iteration Alg. 2 will increase the number of pivots, decrease some entries in low_∂, or both. □

The multi-scale nature of Alg. 2 is advantageous to implement a number of strategies that allow us progressively refine our knowledge of low^*. We describe some of these strategies in the next subsections.

3.2 Clearing by compression

In this section we describe a clearing strategy based on a column compression arguments inspired by the previous compression arguments presented in [1] to reduce operation flop-count. This strategy, in its simplest form presented in Corollary 10, can be straightforwardly implemented in a parallel architecture in Alg. 2, but is not explicitly listed in Alg. 2 for conciseness. Here, we present a compression result that can in some cases help clear columns in the matrix and in some other cases can give improved bounds on the location of low^*. *In doing so, it will be convenient to define the set* $\mathsf{Paired}^\ell \subset \mathsf{Paired}$*, as the sets of elements in* (6) *that have been identified at the ℓ-th iteration by Alg. 2.*
Note that Paired^ℓ at iteration ℓ include the set of unreduced columns that have been identified as negative. In particular, by Theorem 5,

$$\{j \in [m] : \beta_j > 0\} \subset \mathsf{Paired}^\ell.$$

Lemma 9 (Clearing by local compression)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be a boundary matrix of a simplicial complex K. Let $j \in [m]$ and $d = \dim(\sigma_j) \in [\dim(K)]$. Then at iteration ℓ of Alg. 2,*

$$\mathrm{low}^*(j) \in L_j := \{0\} \cup (K_{d-1} \cap [\beta_j, \mathrm{low}_\partial(j)]) \setminus \mathsf{Paired}^\ell. \tag{18}$$

Proof. If $j \in [m]$, and $d = \dim(\sigma_j)$, then $\mathrm{low}^*(j) \in K_{d-1}$ by the definition of ∂; moreover, $\mathrm{low}^*(j) \in [\beta_j, \mathrm{low}_\partial(j)]$ by Theorem 5. The injection condition on the support of low^* implies that if $\mathrm{low}^*(j)$ can not be equal to any of the observed positive low^*'s. Finally, $\mathrm{low}^*(j)$ can not be equal to the index of any of the identified negative columns as $\partial_{\mathrm{low}^*(j)}$ is necessarily positive, and can not be equal to any of the indices of paired positive columns by the injection condition and Lemma 1. Hence, $\mathrm{low}^*(j) \notin \mathsf{Paired}^\ell$. □

Lemma 9 implies the following corollary,

Corollary 10. *Let $\partial \in \mathbb{F}_2^{m\times m}$ and let L_j be defined as in (18) and such that $|L_j| = 1$. Let $i \in L_j$. If $i = 0$, then $\mathrm{low}^*(j) = 0$. Else, $i = \mathrm{low}^*(j)$ and consequently column j is negative and fully reduced and ∂_i is positive.*

3.3 Essential estimation

Another way to reuse the partial knowledge of negative and associated positive columns, as given by $\mathsf{Paired}^\ell \subset \mathsf{Paired}$ be the set of columns that have been identified as paired at the ℓ-th iteration of Alg. 2, is to estimate the essential simplices

Lemma 11 (Essential estimation)**.** *Let $\partial \in \mathbb{F}_2^{m\times m}$ be a boundary matrix and $j \in [m]$, then at iteration ℓ*

$$\mathsf{Ess} \subset \left\{[m] \setminus \mathsf{Paired}^\ell\right\} \setminus \{\mathrm{low}_\partial(j) : j \in [m]\} \tag{19}$$

Proof. Clearly, $\mathsf{Ess} \cap \mathsf{Paired}^\ell = \emptyset$ for all ℓ by the partition on the columns. We show that $\mathsf{Ess} \cap \{\mathrm{low}_\partial(j) : j \in [m]\} = \emptyset$. To do this, we follow a proof by contradiction and suppose that $\mathrm{low}_\partial(j) \in \mathsf{Ess}$, which implies

$$\nexists p \,\text{s.t.}\, \mathrm{low}^*(p) = \mathrm{low}_\partial(j) \tag{20}$$

as otherwise p is the index of a negative column and $\mathrm{low}^*(p)$ the index of an associated positive non-essential column. We show that column j can not be either positive or negative, which leads to a contradiction.

1. $j \in \mathsf{Pos}$: As we are determining $\mathsf{Ess} \cap \{\mathrm{low}_\partial(j) : j \in [m]\}$, we consider only the j such that $\mathrm{low}_\partial(j) > 0$, but since $j \in \mathsf{Pos}$ we must have $\mathrm{low}^*(j) = 0$. By the definition of j being positive there must exist $q < j$ such that $\mathrm{low}^*(q) = \mathrm{low}_\partial(j)$ in order that j can be reduced to the zero column; however, this contradicts (20) by setting $p = q$.

2. $j \in \mathsf{Neg}$: By Theorem 5 $\mathrm{low}_\partial(j) \geq \mathrm{low}^*(j)$. If $\mathrm{low}_\partial(j) = \mathrm{low}^*(j)$, then (20) is contradicted with $p = j$. Otherwise, if $\mathrm{low}_\partial(j) > \mathrm{low}^*(j)$, then by the definition of $\mathrm{low}^*(j)$ there must exist $q < j$ such that $\mathrm{low}^*(q) = \mathrm{low}_\partial(j)$ in order to reduce column j. This again, contradicts (20)

□

Theorem 19 implies that the set of essential columns can be iteratively estimated by a set $E^\ell \subset [m]$ initialised at $E^0 = [m]$ and iteratively updated as,

$$E^{\ell+1} \leftarrow \left(E^\ell \setminus \mathsf{Paired}^\ell\right) \setminus \{\mathrm{low}_\partial(j) : j \in [m]\}. \tag{21}$$

3.4 Distributing the workload

Central to the efficacy of Alg. 2 is the typically large number of pivots available to reduce subsequent columns coupled with the ability of all column additions in its Phase 2 to be performed in parallel. We do not explicitly describe parallelisation strategies for Phase 2 due to architecture specificity, but make a few remarks regarding communication minimisation and early termination in Secs. 3.4.1 and 3.4.2 respectively.

3.4.1 Prioritise reduction of sets $\mathcal{N}(j)$ with large cardinality

Let,

$$\mathcal{C} = \{j \in \mathsf{Pivots} : |\mathcal{N}(j)| > 0\}, \tag{22}$$

be the set of columns that can be used in a given iteration to decrease low_∂.
If only $p < |\mathcal{C}|$ processors are available, and communication of these active pivots is the dominant cost, then it may be desirable to implement only a portion of the possible column additions available in Phase 2 duration an iteration. In such a setting, a priority queue can be used to order the sets $\mathcal{N}(j)$ based on their cardinality and act on multiples of p sets of active columns per iteration of Phase 2.

3.4.2 Prioritise reduction of $\mathcal{N}(j) \cap \mathsf{Neg}$

In settings where low^* cannot be fully determined due to computational or time constraints, the reduction ordering can be prioritised to minimize various objectives. One such objective is to minimize $\|\mathrm{low}_\partial - \mathrm{low}^*\|$ in a suitable norm as rapidly as possible. Recall, Theorem 5 gives bounds on the value of low^* and low_∂ is available by inspection. Due to the the identification of fully reduced negative column, say column j allows clearing of column $\mathrm{low}^*(j)$ there is benefit in prioritising the reduction of columns which are known to be negative, e.g. $\{j \in [m] : \beta_j > 0\} \subset \mathsf{Neg}$. Alternative prioritisations would be application specific, but can include such information as prioritising values for which the persistence is greater or improving the estimation of essential columns as described in Sec. 3.3.

4 Numerical experiments

In this section we evaluate the efficacy of our parallel multi-scale reduction Alg. 2 empirically by comparing its performance against the standard reduction Alg. 1 and standard reduction with a twist Alg. 3. Our evaluation is on a set of synthetic simplicial complexes from point-clouds sampled from a set of a predefined set of ensembles. The point-clouds and their corresponding Rips-Vietoris simplicial complexes are generated with the `Javaplex` [21] library. When building the simplicial complex, we supply the following parameters,

1. Number of points (N): The number of points in the point-cloud to be generated.

2. Maximum dimension ($\dim K$): The maximum dimension of the resulting simplicial complex K.

3. Maximum filtration value (r_T): The maximum radius in a predefined grid of scales $\{r_1, \ldots, r_T\} \subset [0, \infty)$ used to build the filtration.

4. Number of divisions (h): The grid of scales is uniform with spacing h, that is $h := r_{i+1} - r_i$ for $i \in [T-1]$.

Ensemble	N	$\dim K$	r_T	h
Gaussian Points in $\mathbb{R}^3$	15	5	5	10
Figure-8	15	5	5	10
Trefoil Knot	15	5	5	10
$\mathbb{S}^2 \times \mathbb{S}^2$	15	5	5	10

Central to our numerical evaluations in this manuscript is the number of iterations required. The reduction in Algs. 1 and 3 is an *horizontal* procedure in the sense that no column in a given dimension is reduced before the preceeding columns in this dimension have been reduced. In contrast, Alg. 2 is a *diagonal* procedure as it implements an horizontal type of reduction in Phases I and II, but rather than completely reducing the columns in order, it progressively prunes the matrix *vertically* by doing left-to-right column operations every time there is enough information to guarantee that this is possible. Given this crucial difference in design, it becomes difficult to find appropriate benchmarking scenarios and, more fundamentally, to provide a notion of *iteration* that is agnostic to the algorithm's architecture. Hence, in our numerical experiments, we let an iteration be the set of operations that are performed at each cycle of the outer-most loop of the algorithm. This is a reasonable convention as it is consistent with each of the algorithm's concurrency and is also natural for the high-level pseudo-code description of their design. However, we note that this notion of iteration can be optimistic or pessimistic depending on the unit of computational overhead that is being measured. In our experiments, *left-to-right* column operations are chosen as the main unit of computational overhead and, indeed, our algorithm has been designed to minimise this kind of operations. Given the embarrassingly parallel nature of Phase II of Alg. 2, this model is generous with our algorithm as it does not consider possible limitations to the number of processors available and it does not account for the communication overhead between processors. Finally, as our algorithm's practical performance is greatly limited by the hardware architecture, we point out that our numerical experiments should serve as a proof-of-concept of a design that can yield a powerful production implementation rather than a trustworthy comparison of the run-time or flop-count in the general case.
The numerical results presented in this section simulate a parallel implementation and are available at [15]. A truly parallel implementation is being developed.

4.1 Operations are packed in fewer iterations

For each ensemble we sample three point-clouds and reduce their corresponding boundary matrices using each of Algs. 1 - 3. Fig. 1 illustrates the computational complexity, as the number of column additions performed at each iteration and also the number of nontrivial *xor* operations when adding columns in the matrix; that is, the number of scalar operations of the form

$$\{1 \oplus 1, 1 \oplus 0, 0 \oplus 1\}.$$

Scalar operations are relevant because in some storage models like the one given in [7] the computational cost of updating the matrix depends super-linearly on the number of non-zeros in the columns being added. Figs. 1a, 1g and 1j show the number of column additions per iteration of each algorithm performs to reduce each of the simplicial compelexes under consideration. Specifically, they illustrate how that Alg. 2 allocates most of the column additions at the earliest iterations due to its highly parallel nature. On the other hand, Figs. 1b, 1h, and 1k as well as Tab. 1 show that while Alg. 2 reduces the number of iterations, it has a total number of column additions indistingushable to Alg. 3 and about half that of Alg. 1. This shows that Alg. 2 is successful in packing the number of left-to-right column operations into considerably few independent iterations.

4.2 low^* is approximated at multiple scales

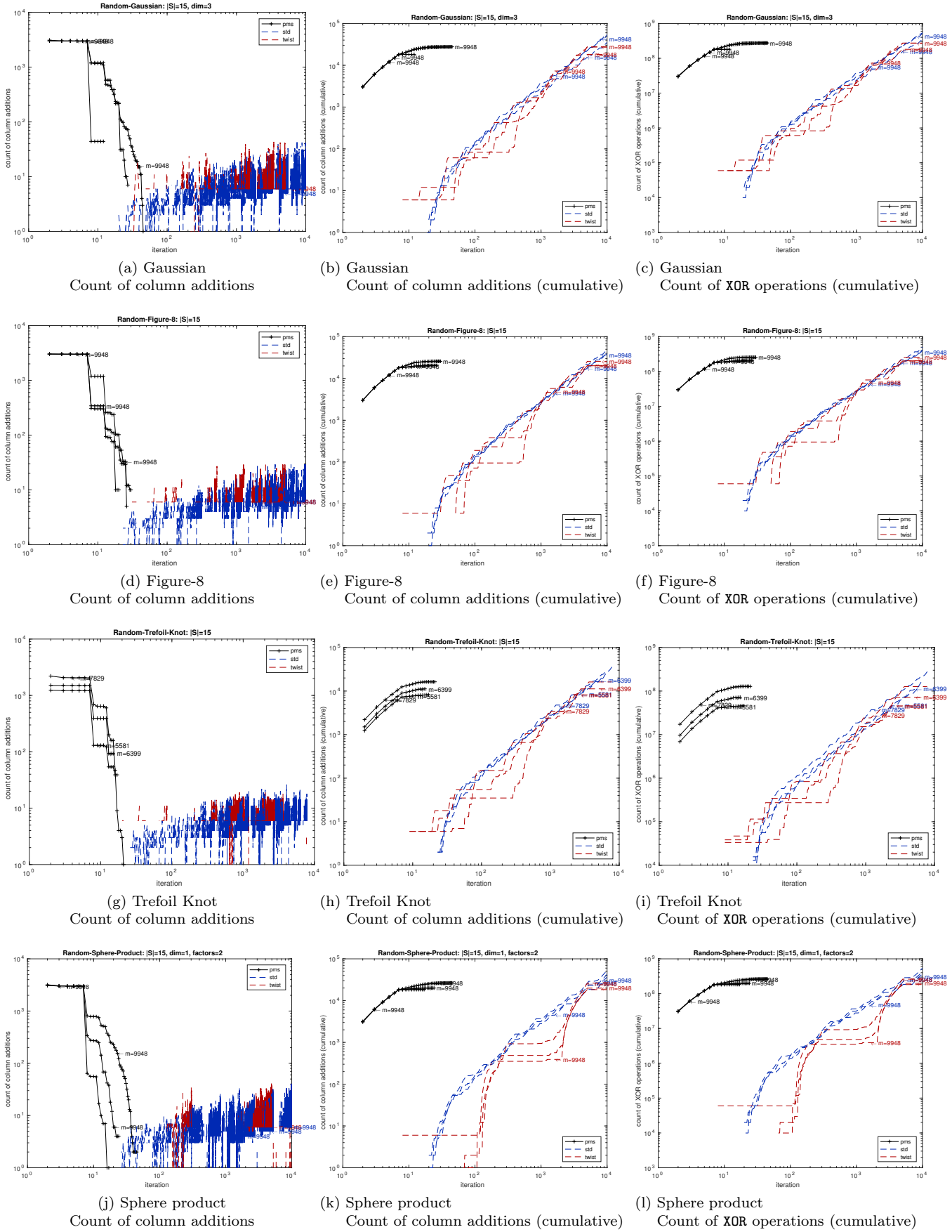
Let $\mathrm{low}_\partial{}^\ell$ be the estimate of low^* at the ℓ-th iteration of an algorithm. We evaluate the quality of the approximation by computing the relative ℓ_1-error as

$$\mathrm{error}^\ell = \frac{\| \mathrm{low}_\partial{}^\ell - \mathrm{low}^* \|_1}{\| \mathrm{low}^* \|_1}. \tag{23}$$

Fig. 2 illustrates an improved rate of reduction in error^k per iteration as each of the algorithms progresses. This is achieved without resorting to further prioritisation of this reduction as described in Sec. 3.4.2 as we are simulating a number of processors in excess of the number of column additions needed per iteration. Tab. 2 shows the precise number of iterations each of Algs. 1 - 3 need to achieve (23) less than 10^{-k} for $k = 1, 2, 3, 4$ and complete reduction. Tab. 2 shows the rapid reduction of (23) by Alg. 2 compared to only appreciable reduction by Algs. 1 and 3 near their complete reductions; e.g. wheras a relative reduction of (23) to 1% is achieved by Alg. 2 in less than 20 iterations, Algs. 1 and 3 take approximately ten and nine thousand iterations respectively for the same reduction. Fig. 3 and Tab. 3 show a similarly early decrease in the fraction of the number of columns which are fully reduced. Remarkably, Alg. 2 has reduced at least 50% of the columns in only two iterations, and all but 1% within no more than 20 iterations; this is in contrast with between approximately two and a half and ten thousand iterations for compreable reductions by Algs. 1 and 3.

4.3 The set Ess can be reliably estimated after a few iterations

Finally, we show how the efficacy of Lemma 11 in estimating the the set of essential columns. Let E^ℓ be defined as



(a) Gaussian
Count of column additions

(b) Gaussian
Count of column additions (cumulative)

(c) Gaussian
Count of `XOR` operations (cumulative)

(d) Figure-8
Count of column additions

(e) Figure-8
Count of column additions (cumulative)

(f) Figure-8
Count of `XOR` operations (cumulative)

(g) Trefoil Knot
Count of column additions

(h) Trefoil Knot
Count of column additions (cumulative)

(i) Trefoil Knot
Count of `XOR` operations (cumulative)

(j) Sphere product
Count of column additions

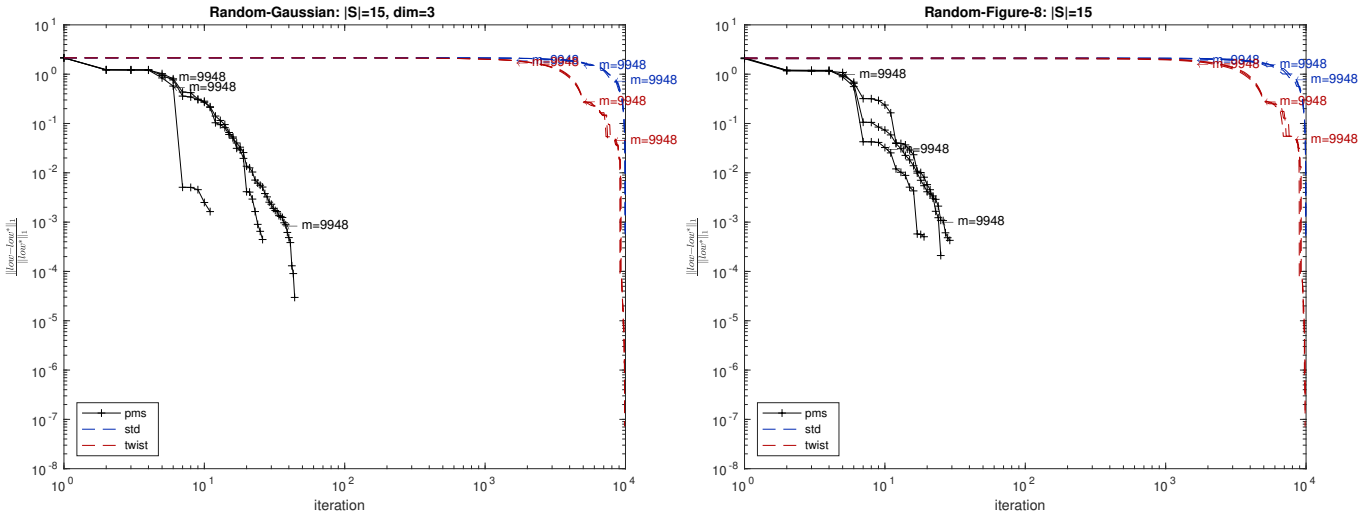
(k) Sphere product
Count of column additions (cumulative)

(l) Sphere product
Count of `XOR` operations (cumulative)

Figure 1: **Benchmarking on column operation overhead.** For each ensemble, three point clouds are sampled and their corresponding simplicial complexes are reduced with Algs. 1 - 3. Their performance is benchmarked in three different ways: through the number of column additions done *at each* iteration (Figs. 1a, 1d, 1g, 1j); through the cumulative number of column additions *up to* a given iteration (Figs. 1b, 1e, 1h, 1k); and through the cumulative number of `XOR` operations done *up to* a given iteration (Figs. 1c, 1f, 1i, 1l).

Sample	Gaussian		Figure-8		Trefoil-knot		Sphere-product	
	std	twist	std	twist	std	twist	std	twist
1	0.52	1.00	0.49	1.00	0.46	1.00	0.50	1.00
2	0.52	1.00	0.51	1.00	0.40	1.00	0.49	1.01
3	0.50	1.00	0.50	1.00	0.44	1.00	0.47	1.00

Table 1: **Ratio of total column additions.** For each ensemble, three point clouds are sampled and the corresponding simplicial complexes are reduced with each algorithm. The *ratio of total column additions* between Alg. 2 and both Algs. 1 and 3 is reported.



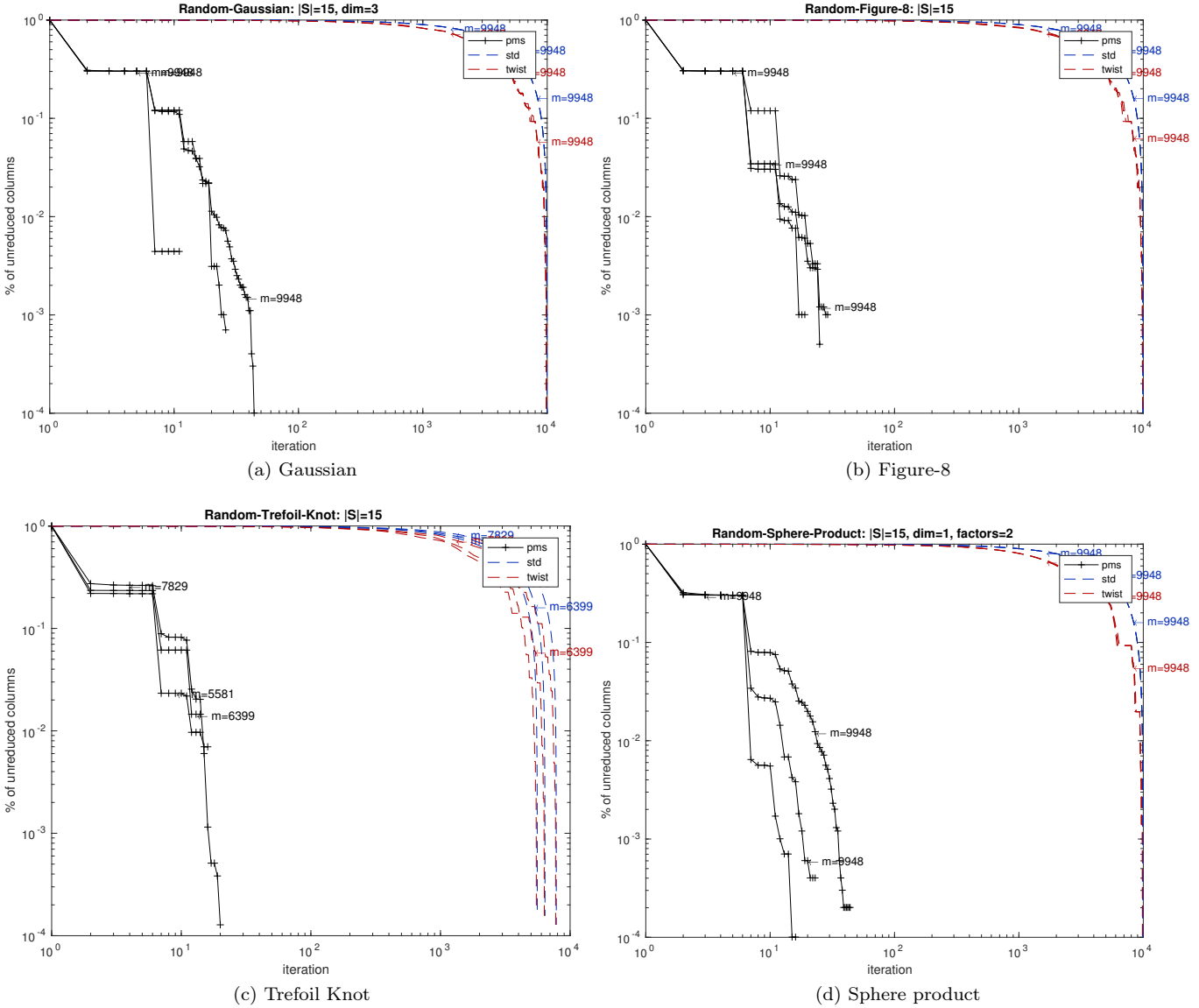
(a) Gaussian (b) Figure-8

(c) Trefoil Knot (d) Sphere product

Figure 2: **Relative ℓ_1-error per iteration.** For each ensemble, three point clouds are sampled and the corresponding simplicial complexes are reduced with each of the algorithms. The relative ℓ_1-error between low_∂ and low^*, as given in (23), is tracked.

$\frac{\|\text{low}_\partial - \text{low}^*\|_1}{\|\text{low}^*\|_1}$	Gaussian			Figure-8			Trefoil-knot			Sphere-product		
	pms	std	twist	pms	std	twist	pms	std	twist	pms	std	twist
0.1	14	9786	7366	9	9756	6759	12	7687	5519	7	9751	5589
0.01	20	9933	9180	17	9930	8916	15	7816	7087	10	9930	8485
0.001	24	9948	9197	25	9948	8942	16	7829	7122	13	9948	8564
0.0001	27	9949	9198	26	9949	8944	20	7830	7309	15	9949	9388
0	27	9949	9858	26	9949	9869	21	7830	7732	17	9949	9840

Table 2: **Iterations to relative ℓ_1-error.** For each ensemble, one point cloud is sampled and the corresponding simplicial complex is reduced with each algorithm. The number of iterations to achieve a given *relative ℓ_1-error level* between low_∂ and low^*, as given by (23), is reported for of Algs. 1 - 3.



(a) Gaussian (b) Figure-8

(c) Trefoil Knot (d) Sphere product

Figure 3: **Proportion of unreduced columns per iteration.** For each ensemble, three point clouds are sampled and the corresponding simplicial complexes are reduced with each of Algs. 1 - 3. The proportion of unreduced columns is presented at each iteration.

Proportion	Gaussian			Figure-8			Trefoil-knot			Sphere-product		
	pms	std	twist	pms	std	twist	pms	std	twist	pms	std	twist
0.90	2	996	528	2	996	499	2	784	392	2	996	519
0.50	2	4975	3405	2	4975	2974	2	3916	2536	2	4975	2958
0.10	12	8955	7424	7	8955	6849	7	7048	6140	7	8955	6115
0.05	15	9452	8432	7	9452	8618	12	7439	6738	7	9452	8404
0.01	20	9850	9471	17	9850	9494	15	7752	7427	7	9850	9461

Table 3: **Iterations to unreduced percentage.** For each ensemble, one point cloud is sampled and the corresponding simplicial complex is reduced with each of Algs. 1 - 3. The number of iterations to achieve a given *proportion of unreduced columns* is presented for each algorithm.

in (21), by Lemma 11, $\mathsf{Ess} \subset E^\ell$ for every ℓ. If E^ℓ is our estimation at iteration ℓ, then the number of true positives, false positives and false negatives in the estimation is, respectively,

$$\begin{aligned} \mathrm{TP} &= |\,\mathsf{Ess} \cap E^\ell| = |\,\mathsf{Ess}\,|, \\ \mathrm{FP} &= |E^\ell \setminus \mathsf{Ess}\,| = |E^\ell| - |\,\mathsf{Ess}\,|, \\ \mathrm{FN} &= |\emptyset| = 0. \end{aligned}$$

To evaluate the quality of the estimation, we compute the *precision* as defined by

$$\text{precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \frac{|\,\mathsf{Ess}\,|}{|E^\ell|}$$

and note that the *recall* $= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$ is equal to 1 due to the estimate giving no false negatives. Fig. 4 and Tab. 4 show the remarkably few iterations needed by Alg. 2 to achieve precision near one while Algs. 1 and 3 show increase in the precision to one only in the later iterations. Specifically, Alg. 2 achieves precision of 95% with two iterations and complete precision within 8 iterations whereas Algs. 1 and 3 require between seven and ten thousand iterations for similar precisions.

5 Conclusions

We have presented a massively parallel algorithm, Alg. 2 for the reduction of boundary matrices in the scalable computation of persistent homology. This work extends the foundational algorithms [1, 7, 12] using many of the same notions, but allowing a dramatically greater distribution of the necessary operations. Our numerical experiments show that Alg 2, as compared with Algs. 1 and 3, is able to pack more operations into few iterations, approximates low* simultaneously at all scales of the simplicial complex filtration, and determines the essential columns in remarkably few iterations. This massively parallel algorithm suggests the reduction of dramatically larger boundary matrices will now be possible, and moreover allows early termination with accurate results when computational constraints are reached. Implementation of Alg. 2 in the leading software packages, as reported in [19], is underway and we expect to report dramatic reduction in computational times in a subsequent manuscript.

6 Acknowledgements

The authors would like to thank Vidit Nanda for his helpful comments. This work was supported by The Alan Turing Institute under the EPSRC grant EP/N510129/1. RMS acknowledges the support of CONACyT.

References

[1] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization III*, pages 103–117. Springer, 2014.

[2] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 31–38. SIAM, 2014.

[3] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

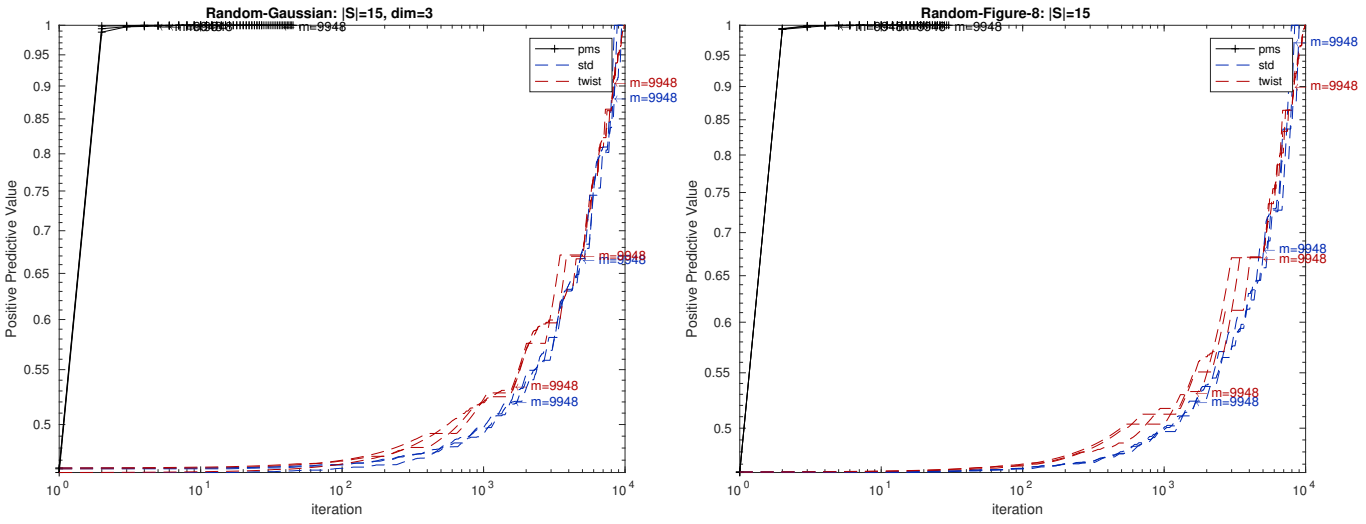
[4] Gunnar Carlsson. Topological pattern recognition for point cloud data. *Acta Numerica*, 23:289, 2014.

[5] Gunnar Carlsson, Tigran Ishkhanov, Vin De Silva, and Afra Zomorodian. On the local behavior of spaces of natural images. *International journal of computer vision*, 76(1):1–12, 2008.

[6] Joseph Minhow Chan, Gunnar Carlsson, and Raul Rabadan. Topology of viral evolution. *Proceedings of the National Academy of Sciences*, 110(46):18566–18571, 2013.

[7] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, volume 11, 2011.

[8] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.



(a) Gaussian (b) Figure-8

(c) Trefoil Knot (d) Sphere product

Figure 4: **Precision of Ess estimation.** For each ensemble, three point clouds are sampled and the corresponding simplicial complexes are reduced with each algorithm. At each iteration, Ess is estimated and its precision is tracked.

Precision	Gaussian			Figure-8			Trefoil-knot			Sphere-product		
	pms	std	twist	pms	std	twist	pms	std	twist	pms	std	twist
0.10	1	1	1	1	1	1	1	1	1	1	1	1
0.50	2	1033	559	2	1231	493	2	2132	1374	2	1016	557
0.90	2	8461	8244	2	7599	8238	2	6691	6445	2	6675	8222
0.95	2	8636	8938	2	7774	8892	2	6871	7123	2	6850	8519
1.00	8	8794	9858	5	7932	9869	8	7004	7732	7	7008	9866

Table 4: **Iterations to essential-estimation precision.** For each ensemble, one point cloud is sampled and the corresponding simplicial complex is reduced with each algorithm. The number of iterations to achieve a given *precision* of the set Ess is given for each algorithm.

[9] Vin De Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Dualities in persistent (co) homology. *Inverse Problems*, 27(12):124003, 2011.

[10] Vin De Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete & Computational Geometry*, 45(4):737–759, 2011.

[11] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[12] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[13] Robert Ghrist. Elementary applied topology. *Book in preperation*, 2014.

[14] PY Lum, G Singh, A Lehman, T Ishkanov, Mikael Vejdemo-Johansson, M Alagappan, J Carlsson, and G Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3, 2013.

[15] Rodrigo Mendoza-Smith. Parallel multiscale reduction of ph filtrations. `https://github.com/rodrgo/tda`, 2017.

[16] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh annual symposium on Computational geometry*, pages 216–225. ACM, 2011.

[17] Dmitriy Morozov. Persistence algorithm takes cubic time in worst case. *BioGeometry News, Dept. Comput. Sci., Duke Univ*, 2, 2005.

[18] Monica Nicolau, Arnold J Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011.

[19] Nina Otter, Mason A Porter, Ulrike Tillmann, Peter Grindrod, and Heather A Harrington. A roadmap for the computation of persistent homology. *arXiv preprint arXiv:1506.08903*, 2015.

[20] Donald R Sheehy. The persistent homology of distance functions under random projection. In *Proceedings of the thirtieth annual symposium on Computational geometry*, page 328. ACM, 2014.

[21] Andrew Tausz, Mikael Vejdemo-Johansson, and Henry Adams. JavaPlex: A research software package for persistent (co)homology. In Han Hong and Chee Yap, editors, *Proceedings of ICMS 2014*, Lecture Notes in Computer Science 8592, pages 129–136, 2014. Software available at `http://appliedtopology.github.io/javaplex/`.

[22] Dane Taylor, Florian Klimm, Heather A Harrington, Miroslav Kramár, Konstantin Mischaikow, Mason A Porter, and Peter J Mucha. Topological data analysis of contagion maps for examining spreading processes on networks. *Nature communications*, 6, 2015.