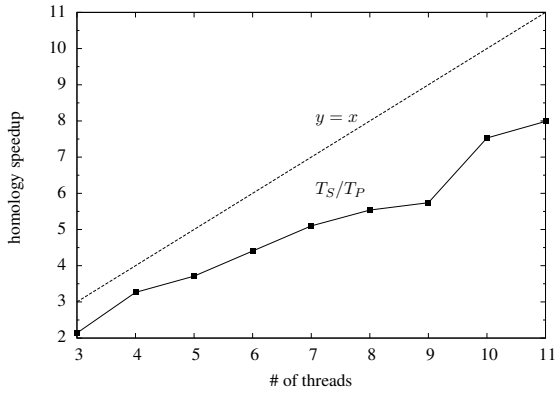
MULTICORE HOMOLOGY

Ryan H. Lewis,[*]*Afra Zomorodian*[†]

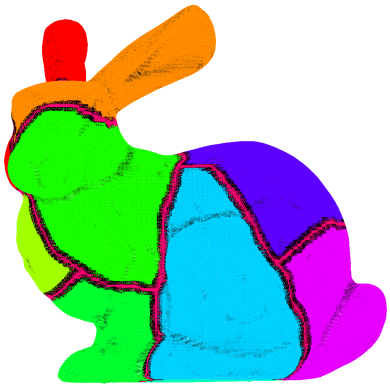
ABSTRACT. In this work we design and implement a framework for the parallel computation of field homology using the Mayer-Vietoris principle. We begin with a cover of the input space by overlapping subspaces. Then we use this cover to build the Mayer-Vietoris blowup complex, a topological space, which organizes the various subspaces needed for employing the Mayer-Vietoris principle. Finally, we compute the homology of each piece of the blowup complex in parallel and use the persistence algorithm to glue these results together. We present a simplistic model for the problem of finding covers which will lend themselves to parallelism and show that finding such covers is NP-HARD. We describe an algorithm for producing covers with a simple structure and bounded overlap based on graph partitions. This approach allows us to avoid the explicit construction of the blowup complex saving time and space. We implement our algorithms for multicore computers, and demonstrate their efficacy with a suite of experiments. For example, we achieve roughly an $8\times$ speedup of the homology computations on a 3-dimensional complex with about 10 million simplices using 11 cores.

We include line numbers for our reviewers' ease of reference.

[*]*Stanford University*, rhl@stanford.edu
[†]*D.E. Shaw*, afra@cs.dartmouth.edu



(a) We achieve a 68.9% efficiency on 11 cores. The line $y = x$ displays perfect efficiency.



(b) A visualization of the input space, and its decomposition into 11 pieces.

Figure 1: Homology speedup on a 3-dimensional complex with 9.7 M simplices.

1 Introduction

In this paper, we present a fast multicore algorithm for computing the homology of arbitrary dimensional cell complexes over field coefficients. Figure 1(a) shows the speedup factor of our algorithm for computing homology over $\mathbb{Z}_2$ coefficients of the data set B, shown in Section 5. By decomposing the space into the 11 pieces visualized in Figure 1(b), we are able to compute the homology of the input space in 3.73 seconds, approximately eight times faster than the 29.8 seconds necessary for serial computation. When comparing the total running time including data preprocessing for both algorithms our parallel algorithm takes 9.38 seconds as compared to 35.67 seconds in serial. All our timings are done on a 64-Bit GNU/Linux machine with dual, six core, 3.47Ghz Intel X5690 CPUs, and hyperthreading disabled.

1.1 Motivation

We are motivated by *topological data analysis* which attempts to extract a topological understanding of scientific data from finite sets of samples. Usually data analysis assumes that the input point cloud comes from some underlying geometric space. Topological data analysis focuses on the recovery of the lost topology of this underlying space [3]. The classic pipeline for this analysis follows a two step process. First, we compute a combinatorial model approximating the structure of the underlying space. Second we compute topological invariants on these structures. One popular invariant, persistent homology [6, 23], captures multiscale topological structure. Computing field homology, especially over $\mathbb{Z}_2$ coefficients, is an integral part of topological data analysis.

Computing homology of complicated spaces requires both space and time. In this paper, we focus on developing a parallel algorithm to compute homology on multicore shared-memory machines. This algorithm is a first step toward a distributed-memory algorithm that will allow us to compute the persistent homology of massive structures on computer clusters.

1.2 Prior Work

There is a large literature on serial computation of integer homology. Dumas et al. review algorithms for computing integer homology that take advantage of the sparsity of boundary matrices derived from simplicial complexes [4]. Their software is available within the GAP software package [9]. Joswig surveys the computation of invariants, including homology for simplicial spaces with a focus on manifolds [11]. Kaczyński et al. develop heuristics to compute cubical homology [12]. Kaltofen et al. provides a theoretical investigation of randomized parallel algorithms for computing the Smith normal form [13, 14] over finite fields and $\mathbb{Q}$, however, these algorithms are not useful in practice [5].

Any parallel computation of homology would require a decomposition of the space into pieces. The theory of *spectral sequences* explains how to compute the homology of a space from its pieces. In our setting we decompose our input space into pieces using a *cover*; the Mayer-Vietoris spectral sequence expresses the relationship between the homology of these pieces of a space and the homology of the space itself. This makes the Mayer-Vietoris spectral sequence a natural gadget to study when developing algorithms for parallel homology [10]. The Mayer-Vietoris blowup complex is a topological analogue of the Mayer-Vietoris spectral sequence and it is used by Zomorodian and Carlsson to compute localized homology [24]. Merino et al. use the Mayer-Vietoris sequence to compute the homology of three-dimensional simplicial complexes [2]. Lipsky et al. use the Mayer-Vietoris spectral sequence in an attempt to derive a parallel algorithm [16]. All works are theoretical in nature. The researchers do not address the algorithmic issues of finding covers; Nor do they provide implementations of their algorithms or any empirical results.

The Mayer-Vietoris spectral sequence is not the only algebraic tool which is useful for parallel homology computation. The spectral sequence of a filtration shows how a sequence of *relative homology* computations may be carried out in parallel on contiguous chunks of a boundary matrix to arrive at the homology of a space. Bauer, Kerber, and Reininghaus explore this approach to computing homology in parallel [1].

1.3 Our Work

In this paper we design and implement a divide and conquer framework for computing the field homology of a cellular space in parallel. Field homology is popular in topological data analysis since it has polynomial complexity and the persistence algorithm exhibits linear-time behavior in practice [6, 23]. Our framework relies on the Mayer-Vietoris blowup complex a spatial version of the Mayer-Vietoris spectral sequence [24]. The Mayer-Vietoris blowup complex is the *total complex* of the terms of the first page of the Mayer-Vietoris spectral sequence and it's homology groups are isomorphic to that of the original space. In this work we show how to build the Mayer-Vietoris blowup complex and compute it's homology in parallel using the persistence algorithm. Our approach may be viewed as parallel computation of relative homology on chunks of the boundary matrix for the blowup complex. However, because of the structure of the blowup complex, many relative chunks do not need to be further reduced against each other, unlike the algorithm presented by Bauer, Kerber, and Reininghaus [1]. Since this pipeline requires a cover of the input space we investigate the general problem of finding covers of spaces. In Section 4 we identify a class of covers which lend themselves to efficient parallel algorithms and model the problem of finding covers

in this class as an optimization problem. We then show that solving this problem is NP-HARD. Motivated by this result we instead provide a algorithm for producing covers with bounded overlap, based on graph partitioning, in Section 4.2. We observe that since we are limiting our viewpoint to ordinary homology the special nature of covers generated by our algorithm allows us to avoid explicitly constructing the blowup complex. Nevertheless, the blowup complex is useful for organizing the work needed to be carried out in the general case. In Section 5 we present the results of a suite of experiments from a multicore version of our parallel algorithms and provide preliminary experimental results. All of the techniques in this paper are deterministic. Our software is included in a publicly available library.

2 Background

We begin with a review of simplicial complexes, homology, and blowup complexes. We refer a non-specialist in algebraic topology to Hatcher [10] and to Zomorodian [21, Chapter 13] for computational topology. In principle the methods outlined in this paper generalize to any type of cellular space, however we restrict ourselves to simplicial complexes.

2.1 Preliminaries

Let $[n] = \{0, 1, \ldots, n\}$ be the first $n+1$ natural numbers. This definition is not conventional but we adopt the notation used in previous work for continuity with prior work [24]. A *multiset* is a pair (A, c) where $c : A \to \mathbb{N}$. A *decomposition* of a set S is a collection of nonempty subsets of S whose union is S. A *partition* of a set S is a decomposition of S by disjoint sets. A *graph* $G = (V, E)$ is a set V of *vertices*, and a set $E \subseteq V \times V$ of *edges*. Suppose we have a graph $G = (V, E)$. A *graph partition* is a partition $P = \{P_i\}_{i \in [n-1]}$ of V into n subsets. A *cut* is a partition of V into two sets A and B.

A *simplicial complex* is a collection K of finite sets called *simplices* such that if $\sigma \in K$ and $\tau \subseteq \sigma$ then $\sigma \in K$. We say that τ is a *face* of σ, its *coface*. A simplex is *maximal* if it has no proper coface in K. The set of maximal cells of a simplicial complex K is $M(K)$. If $|\sigma| = k+1$ then σ is a k-simplex, it has *dimension* k, denoted $\dim \sigma = k$. We say that K is *d-dimensional* if $d = \max_{\sigma \in K} \dim \sigma$. Given a simplicial complex K the set of maximal cells can be enumerated in $O(md)$ time.

Suppose we have a subset $L \subseteq K$. L is a *subcomplex* if it is a simplicial complex. The *closure* of L is $\mathrm{Cl}(L) = \{\tau \mid \tau \subseteq \sigma \in L\}$. The *$k$-skeleton* of a complex K is the set of all simplices of dimension less than or equal to k. Note that the 1-skeleton of any complex may be viewed as a graph. Let Δ^n be the n-simplex defined on $[n]$. We note that Δ^n is traditionally defined in a geometric setting and is called the *standard n-simplex* [10], although we are using an abstract version here for our purposes. For any *indexing set* $J \subseteq [n]$, Δ^J is the $(|J|-1)$ dimensional face of Δ^n that is defined on J. We define a *filtration* of K to be a partial ordering on the simplices of K such that every prefix of the ordering is a subcomplex and denote it as $\leq_K$. Given a simplicial complex K, An *open cover* of K is a decomposition of K and a *closed cover* U of K, is a decomposition of K by subcomplexes. Except where explicitly specified all covers in this work are closed. The *nerve* $N(U)$ of a cover U is the simplicial complex on $[|U|-1]$ whose

k-simplices represent the non-trivial intersections of subsets of U of size $k+1$. The nerve is a subcomplex of the standard n-simplex and so we denote its simplices by Δ^J where $J \subseteq [|U|-1]$. It is convenient to encode the cover U as a map from K to $N(U)$ where each simplex $\sigma \in K$ is mapped to the maximal cell in $N(U)$ which represents the indices of all the cover sets containing σ.

A simplicial complex may be viewed as the result of gluing simplices of different dimensions along common faces. Other types of complexes are defined similarly using different types of *cells*. Such *cellular* complexes include Δ-complexes, *cubical* complexes, *simplicial sets*, and *CW-complexes*, to name a few [7, 10, 12, 17]. In this paper, we restrict to simplicial complexes as input, although our methods generalize easily to other types of complexes.

2.2 Homology

In this section, we describe the homology of cellular spaces over field coefficients. Homology, however, is an invariant of arbitrary topological spaces and may be computed over arbitrary coefficient rings [10]. Suppose we are given a finite cellular complex K and a field k. The *nth chain vector space* C_n is the k-vector space generated by the set of n-dimensional cells of K, its *canonical basis*. Suppose we are given a linear *boundary operator* $\partial_n \colon C_n \to C_{n-1}$ such that $\partial_n \circ \partial_{n-1} \equiv 0$ for any n. The boundary operator connects the chain vector space into a *chain complex* C_*:

$$\cdots \to C_{n+1} \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \to \cdots .$$

Given any chain complex, the *nth homology vector space* H_n is:

$$H_n = \ker \partial_n \,/\, \operatorname{im} \partial_{n+1}, \tag{1}$$

where $\ker$ and im are the *kernel* and *image* of ∂, respectively. Each homology vector space is characterized fully by its *Betti number*, $\beta_n = \dim H_n$. We now only need to define boundary operators to get homology. For simplicial complexes, we begin by describing its action on any n-simplex $[v_0, \ldots, v_n] \in K$:

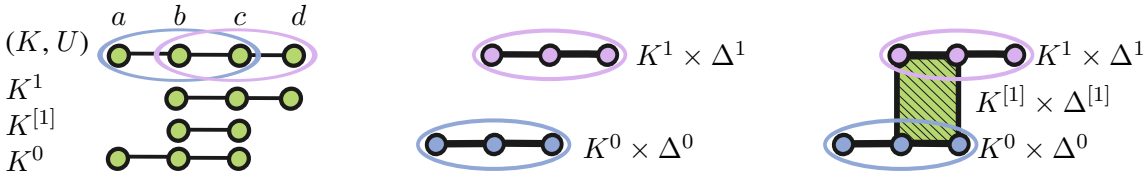
$$\partial_n [v_0, \ldots, v_n] = \sum_i (-1)^i [v_0, \ldots, \hat{v_i}, \ldots, v_n],$$

where $\hat{v_i}$ indicates that v_i is deleted from the vertex sequence. The boundary operator is the linear extension of the above action.

Over field coefficients, homology is a vector space characterized by its dimension, so we may compute homology using *Gaussian elimination* [20]. In practice, we use the *persistence algorithm* [6, 23]. This algorithm can compute the homology of any *based persistence complex* [24], a class that includes simplicial complexes as well as the blowup complex. As input, this algorithm requires a basis for the chain complex C_*, a boundary operator ∂_n, and a filtration on the basis elements. As such, we focus on characterizing these three inputs for computing the homology of a blowup complex using the persistence algorithm.

2.3 Blowup Complex

Like homology, the blowup complex may be defined for arbitrary topological spaces [24], but in this paper, we focus on blowups of simplicial complexes. For a longer exposition of the Mayer-Vietoris



(a) Space and Cover

(b) Local pieces of the blowup complex.

(c) The blowup complex.

Figure 2: Our approach. We are given a space equipped with a cover (a), the former represented by a path with four vertices and three edges and the latter represented by ovals. First, at time $(t = 0)$ we blowup up the space into local pieces (b), each local piece is a copy of the corresponding cover set, then, at $(t = 1)$ we glue together duplicated simplices by adding in the blowup cells, rendering them homologically equivalent, which gives us the blowup complex (c).

blowup complex we refer the reader to Zomorodian & Carlsson [24]. Given a simplicial complex K and cover $U = \{U^i\}_i$ of n subcomplexes, let $K^J = \cap_{k \in J} U^j$. The *Mayer-Vietoris blowup complex* is:

$$K^U = \bigcup_{\emptyset \neq J \subseteq [n-1]} K^J \times \Delta^J,$$

where $\times$ is the Cartesian product [24] and Δ^J is a face of $N(U)$.

Example 2.3.1. Suppose we have a space K with cover $U = \{U^0, U^1\}$ as is shown on the top of Figure 2(a), where we use a line as a representative space and ovals to indicate cover sets, and the four vertices of the line are labeled from left to right as a, b, c, d respectively. The cover defines the intersection $K^{[1]} = K^{\{0,1\}}$. The corresponding blowup is shown in in Figure 2(c). We list each of the relevant pieces of K^U as well as the nerve of the cover where we denote simplices as strings for brevity.

$$\begin{aligned} N(U) &= \{0, 1, 01\} \\ K^0 \times \Delta^{\{0\}} &= \{a, b, c, ab, bc\} \times \{0\}, \\ K^1 \times \Delta^{\{1\}} &= \{b, c, d, bc, bd\} \times \{0\}, \\ K^{[1]} \times \Delta^{[1]} &= \{b, c, bc\} \times \{01\}. \end{aligned}$$

Our work is based on the following key property. The blowup complex K^U has the same homology as its base complex K in any dimension: $H_n(K^U) \cong H_n(K)$ for any n [24, Lemma 1]. Our approach then is to compute homology of the blowup complex instead of the base complex. The blowup has a structure that allows for computation in parallel, unlike the base complex.

To compute the homology of the blowup complex, we may interpret the definition above in two different ways. At the space level, we may view each cell of the blowup complex as a product of two simplices $\sigma \times \Delta^J$, where $\sigma \in K$ and $\Delta^J \in \Delta^n$. For example, the product of two edges, $bc \times \{01\}$, gives us a quadrilateral cell in Example 2.3.1. We may then triangulate the blowup complex to get a simplicial complex in order to compute its homology. While this approach is theoretically correct, it is computationally prohibitive in practice due to the need for triangulation and unnecessary. We use an alternative interpretation to avoid this work.

Alternatively, we examine the chain complex attached to the blowup complex. A basis for $C_n(K^U)$ is the set composed of elements $\sigma \otimes \Delta^J$ for all $\emptyset \neq J \subseteq [n-1]$ and simplices $\sigma \in K^J$ where $\dim \sigma + \dim \Delta^J = n$. We define the boundary operator as [24, Lemma 4]:

$$\partial(\sigma \otimes \Delta^J) = \partial\sigma \otimes \Delta^J + (-1)^{\dim \sigma} \sigma \otimes \partial\Delta^J.$$

Here, we are defining a boundary operator for the blowup complex on the left using the boundary operators on the right, all of which are simplicial and were defined in the previous section.

Example 2.3.2. The boundary of the quadrilateral cell $bc \times \{01\}$: in Example 2.3.1 is:

$$\begin{aligned} \partial(bc \otimes \{01\}) &= \partial(bc) \otimes \{01\} - bc \otimes \partial(\{01\}) \\ &= c \otimes \{01\} - b \otimes \{01\} - bc \otimes \{1\} + bc \otimes \{0\}. \end{aligned}$$

Having specified the basis for the chain complex and a boundary operator, we now need a filtration on the basis elements in order to use the persistence algorithm. In principle an arbitrary filtration will do. But for computing homology in parallel, we will specify a particular filtration whose structure mirrors the structure of the blowup complex.

3 Blowup Structure

The construction of the blowup complex has two phases, the *local* and the *global* phase. In the local phase, the complex explodes into multiple pieces, as in Figure 2(b). This means that we have potentially multiple versions of a simplex represented by the various sets in the cover. For example, since edge bc falls within both sets in the cover in Figure 2(a), it is represented by two cells $bc \times 0$ and $bc \times 1$. The pieces at the local stage are disjoint, so we may compute the homology of the pieces in parallel.

The global phase specifies cells that glue the different versions of the original simplices together, rendering them homologically equivalent. For example, in Figure 2(c), the cell $b \times 01$ connects $b \times 0$ and $b \times 1$.

To describe a filtration on the blowup complex, we assume that we have an arbitrary filtration $\leq_K$ on the simplices of our input complex K. In practice, we often label the vertices of a complex using numbers or letters and use the lexicographic ordering of the vertices to generate a filtration on the complex. We use the same procedure with $N(U)$ as its vertices are numbered by definition.

Given a filtration $\leq_K$ on K and $\leq_{N(U)}$ on Δ^n we define a partial order $\leq_{K^U}$ by ordering all cells in the local phase before those in the global phase. This amounts to comparing two cells $\sigma \times \Delta^M$ and $\tau \times \Delta^N$ by comparing the second factor according to $\leq_{N(U)}$. We may complete this partial order to a filtration by then comparing the first factor according to $\leq_K$.

Example 3.0.3. Figure 2(c) has the following filtration:

$$(\overbrace{a \times 0, b \times 0, c \times 0, ab \times 0, bc \times 0}^{\text{Local Piece \#0 } (t=0)}, \overbrace{b \times 1, c \times 1, d \times 1, bc \times 1, cd \times 1}^{\text{Local Piece \#1 } (t=0)}, \overbrace{b \times 01, c \times 01, bc \times 01}^{\text{Global Piece } (t=1)}).$$

MULTICORE-HOMOLOGY(K, p)
1 $U \leftarrow$ COVER(K, p)
2 $K^U \leftarrow$ BUILD-BLOWUP-COMPLEX(K, U)
3 **parallel for** $\Delta^J \in N(U)$
4 **do** PAIR-CELLS$(K^J \times \Delta^J)$
5 **for** $\Delta^J \in N(U)$ a d-cell for $d > 0$.
6 **do** PAIR-CELLS$(\mathrm{Cl}\,(K^J \times \Delta^J))$

BUILD-BLOWUP-COMPLEX(K, U)
1 $K^U \leftarrow \emptyset$
2 **parallel for** $\sigma \in K$
3 **do for** $\tau \subseteq U[\sigma]$
4 **do** $K^U \leftarrow K^U \cup (\sigma \times \tau)$

Figure 3: Psuedocode for computing the blowup complex and it's homology in parallel. COVER can be any algorithm for generating a cover of K by p subspaces. We encode the cover U as a map from K to $N(U)$ where each simplex is mapped to the maximal cell in $N(U)$ to which it is a member of. The procedure PAIR-CELLS is defined in the Computational Topology section of the Algorithms and Theory of Computation Handbook [21].

Algorithm 3 shows how to build the blowup complex and compute it's homology in parallel. The procedure BUILD-BLOWUP-COMPLEX runs in parallel and has parallel running time $O(2m/p + p)$ time where $m = |K^U|$ and p is the number of processors available. In practice BUILD-BLOWUP-COMPLEX not only produces a blowup complex but also the filtration of the blowup complex prescribed above. Aside from its output BUILD-BLOWUP-COMPLEX only uses $O(p)$ extra space. While we have written that the second for-loop in MULTICORE-HOMOLOGY is to be carried out in serial more parallelism is possible. By ordering the iterations of the loop according to the topology of $N(U)$. We leave this added parallelism for future work.

The size of the blowup complex depends on the cover. In the worst case, all of the simplices in a space K are contained within all n sets of the cover U. In this case, for each simplex $\sigma \in K$ we have a corresponding product cell $\sigma \times \Delta^n$, which has 2^n faces. That is, the blowup complex *blows up* K to be 2^n times larger, thus deserving its name. Therefore, it is imperative to find a cover which minimizes blowup but whose nerve also allows for parallelism.

4 Covers

Given a simplicial complex K, our goal is to compute its homology. Our approach, as illustrated in Figure 2, is to find a cover, build the associated blowup complex, and compute the homology of the blowup complex in parallel. We have now explained all the steps of this approach except how to find a cover. We begin in Section 4.1 by identifying properties of covers that lend themselves to efficient computation. We continue by stating an optimization problem over covers which minimizes the explosive size of the blowup of a complex. We then show this optimization problem to be NP-HARD. In Section 4.2, we describe an algorithm that generates covers which have a simple structure, and bounded overlap based on graph partitions. We end this section by showing how a partition of the 0-cells of a complex can be lifted to a partition of a filtration on the complex which can be used to compute homology in parallel without building the blowup complex.

4.1 Minimum Blowups

In this section, we formalize the problem of finding covers that minimize blowup size. We show that this problem is NP-HARD, and its decision-variant, NP-COMPLETE.

It should be clear that seek a cover which does not yield a large blowup complex. To quantify blowup, we define the *blowup factor* as the ratio: $|K^U|/|K|$. We search for a cover U of size p that minimizes the blowup factor. Since we intend to compute the homology of each local piece in parallel, the number of local pieces of the blowup should be the number p of available processors. Finally, each local piece should be approximately the same size. There are many ways of modeling this last constraint. We model it by enforcing that no cover set should be larger than a fixed fraction α of the size of the input complex, where $\alpha \in (1/p, 1)$. Putting together all of the desired properties of blowups, we have the following optimization problem stated for $p = 2$ and $\alpha \in (1/2, 1)$:

PROBLEM: α-BALANCED-MINIMUM-BLOWUP
INSTANCE: A simplicial complex K
GOAL: Find a cover U of K with 2 elements such that:

$$\max |U_i| \leq \alpha |K| \text{ and } |K^U|/|K| \text{ is minimized.}$$

Our goal is to show that this problem is NP-HARD and it's decision problem variant NP-COMPLETE. For the decision problem variant to be NP-COMPLETE we need to show that $|K^U|/|K|$ may be evaluated in polynomial time. Recall that K^U might be exponentially larger than K. For covers by two sets we may employ the following lemma.

Lemma 1. *Let K be a complex and let $U = \{U_1, U_2\}$ be a cover of K by two sub complexes. Then:*

$$|K^U|/|K| = 1 + 2\frac{|U_1 \cap U_2|}{|K|}.$$

Proof. This follows directly from the product cell definition of K^U. □

Now we observe an important necessary condition of optimal solutions to α-BALANCED-MINIMUM-BLOWUP.

Lemma 2. *Given a complex K and $U = \{U_1, U_2\}$ be an optimal solution of α-BALANCED-MINIMUM-BLOWUP, then U may be viewed as a partition of $M(K)$ the maximal cells of K.*

Proof. If $\sigma \in U_i \cap U_j$ is a maximal cell, then consider the cover U' obtained by removing σ from the set of larger cardinality. U' is certainly a cover satisfying α-balance but by Lemma 1 the blowup factor has decreased which contradicts the optimality of U. □

Suppose the input to α-BALANCED-MINIMUM-BLOWUP is a graph G. In this context any cover U of G is a pair of subgraphs G_1, G_2. Lemma 2 tells us that in any optimal solution the intersection $I = G_1 \cap G_2$ of these two subgraphs is a set of vertices. The requirement that U is a cover implies that I is a vertex separator. α-SUBGRAPH-BALANCED-VERTEX-SEPARATOR is a similar problem on graphs. For any $\alpha \in (\frac{1}{2}, 1)$:

PROBLEM: α-SUBGRAPH-BALANCED-VERTEX-SEPARATOR
INSTANCE: A graph G
GOAL: Find a vertex separator (V_1, V_2, I) of G such that:

$$|I| \text{ is minimized. subject to } \max_i (|V_i| + |E_i|) + |I| \leq \alpha(|V| + |E|)$$

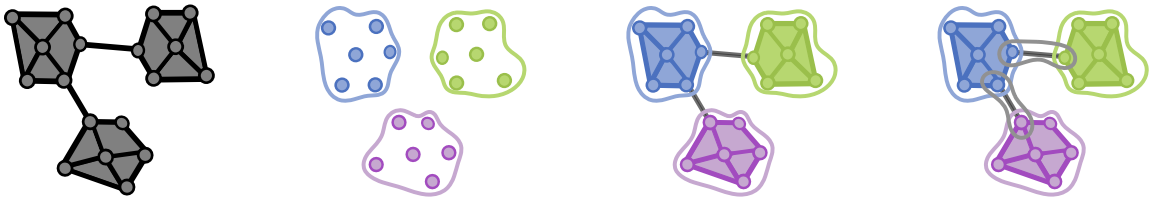
where E_i is the set of edges with at least one endpoint in V_i. α-SUBGRAPH-BALANCED-VERTEX-SEPARATOR is NP-HARD for any $\alpha \in (\frac{1}{2}, 1)$ and it's decision problem variant is NP-COMPLETE [**?**].

Theorem 1. *For any $\alpha \in (1/2, 1)$ the optimization problem α-*BALANCED-MINIMUM-BLOWUP *is* NP-HARD *and it's decision problem variant* NP-COMPLETE.

Proof. We restrict our problem to graph instances of α-SUBGRAPH-BALANCED-VERTEX-SEPARATOR and the problems become equivalent. □

This procedure shows us how finding optimal blowups of graphs finds us a partition of that graph. In the next section we show how given a complex K finding partition of it's 1-skeleton can be used to produce a cover of the entire complex.

4.2 Partition-Based Covers



(a) Input Complex K (b) Partition P (c) Open Cover $\tilde{U}$ (d) Cover U

Figure 4: Our heuristic algorithm for partition based cover construction. Given the input complex K shown in (a) we first, partition the vertex set of the underlying graph G as shown in (b) then, extend this to an open cover $\tilde{U}$ of K (c). Finally, we produce, U a cover (d).

In this section we describe an algorithm for generating covers on an arbitrary complex from a partition of its one skeleton. We emphasize that while we propose a specific algorithm for generating these covers any procedure for generating covers suffices. There may be a better approach for generating covers than the one presented. Recall that in the worst case, a cover may produce an exponentially large blowup. However, the heuristic presented in this section guarantees that $|K^U|/|K| < 3$.

There are many algorithms for generating covers, and they are all valid inputs to our parallel algorithms. Zomorodian & Carlsson consider two naïve methods for cover enumeration, *random ϵ-balls* and *tilings* [24]. In a less general setting one might consider algorithms based on *Voronoi diagrams* or level sets of *Morse functions*. However, the former method is computationally intractable in high dimensions and the latter is unable to control the intersection patterns of cover sets. There

is a natural way of generating a cover of an arbitrary simplicial complex from a partition of its one skeleton with a simple intersection pattern.

The algorithm PARTITION-BASED-COVER, illustrated in Figure 4, takes a complex K and positive integer $p \geq 2$ as input and produces a cover U of size $p+1$ as output. First, we extract the one-skeleton of K and represent it as a graph G. Second, we find a graph partition P of G of size p. Third, we extend P to an *open cover* $\tilde{U}$. Finally, we extend $\tilde{U}$ to a cover U. The algorithms for producing these two covers are called OPEN-COVER and CLOSE-COVER, respectively.

There are many algorithms for computing partitions of graphs which fall into four major classes of algorithms: geometric, non-geometric, spectral, and hybrid methods [8]. In pratice, we use METIS, a hybrid method, since it tends to produce balanced partitions quickly [15]. Again any partitioning scheme will work. Next, we describe OPEN-COVER(K, P), which extends a partition of G to a cover of K.

The procedure OPEN-COVER(K, P) is given in Algorithm 5 and outputs an open cover $\tilde{U} = \{\tilde{U}_i\}_{i\in[p]}$ which is a partition of K. Given a partition $P = \{P_i\}_{i\in[p-1]}$ of the vertex set of G we expand P to $\tilde{U}$. Specifically we first create sets $\tilde{U} = \{\tilde{U}_i\}_{i\in[p]}$ where a simplex σ is placed into $\tilde{U}_i$ for $i \in [p-1]$ if all of it's vertices lie in P_i and is added to $\tilde{U}_p$ otherwise.

In the procedure CLOSE-COVER we replace $\tilde{C}_i$ with $U_i = \mathrm{Cl}\,(\tilde{U}_i)$. However, $\tilde{U}_i$ for $i \in [p-1]$ is closed by construction so we only close the last set. We have the following lemma:

Lemma 3. *Given a complex K, $p \geq 2$, PARTITION-BASED-COVER(K, p) generates a cover U with $|K^U|/|K| < 3$.*

Proof. For a complex K and $p \geq 2$, let U be the output of PARTITION-BASED-COVER(K, p). The first p cover sets are disjoint since they are formed from disjoint sets of vertices. Therefore there can be at most pairwise intersection so by the proof of Lemma 1 we have that $|K^U|/|K| < 3$. □

Lemma 3 ensures that the necessary condition in Lemma 2, is satisfied. Finally, both OPEN-COVER and CLOSE-COVER can be implemented in parallel. The nerve of the covers gen-

Input: A complex K, and a graph partition P.
Output: A cover U, of size $|P|+1$.

OPEN-COVER(K, P)
1 $U \leftarrow \emptyset$
2 **parallel for** $\sigma \leftarrow \sigma_1$ **to** $\sigma_m \in K$
3 **do** $U[\sigma] \leftarrow$ PARTITION-CELL(P, σ)
4 **return** U

Input: A graph partition P of size p, and simplex σ
Output: The index $i \in [p]$ of $\tilde{U}$ to place σ.

PARTITION-CELL$(P, \sigma = [v_0, \ldots, v_d])$
1 $R \leftarrow \emptyset$
2 **for** $v \leftarrow v_0$ **to** $v_d \in \sigma$
3 **do** $R \leftarrow P(v_0)$
4 **if** $|R| = 1$ **return** $R[0]$
5 **else return** $|P|$

Figure 5: The pseudocode for OPEN-COVER which runs in $O(md/p)$ time, where m is the number of simplices in K a d-dimensional complex, and p is the maximum number of available cores. P is indexed starting at 0. For a vertex v, $P(v)$ denotes the index of the partition set of P containing v.

erated by PARTITION-BASED-COVER have the topology of a star. Observe that in this particular

	Input Statistics					Phase I Runtimes			Phase II Runtimes			
D	$\|D\|$	ϵ, p	$\|E\|$	d	$\|K\|$	T_F	T_C	T_B	T_S	T_{ST}	T_P^{11}	T_{PT}^{11}
M	249,920	-	1,272,319	10	46,530,559	11.02	8.13	5.46	67.08	97.31	11.91	36.38
C	20	-	190	19	1,048,575	.20	.42	.12	3.86	4.45	4.46	5.27
B	34,837	0.05	489,876	3	9,714,912	2.18	2.09	1.26	29.81	35.67	3.73	9.38
S	50,000	0.18	546,388	8	19,134,612	5.04	5.09	2.89	43.37	59.45	8.77	21.89
G	5000	0.05	624,677	4	3,639,351	0.0	0.09	0.01	21.37	21.40	23.63	23.73

Table 1: Input Statistics: The name D, and size of each data set $|D|$, as well as input parameter ϵ, embedding dimension $d = \dim D$, size $|K|$, and edge-set size $|E|$, of each complex, K. Phase I Runtimes: Time for each part of the preprocessing phase of the parallel algorithm: building the initial filtration T_F, cover T_C, and blowup complex T_B with its corresponding filtration. The only preprocessing that the serial algorithm performs is to compute a filtration on the input. Phase II Runtimes: Time for homology computations in serial and parallel on 11 cores T_S, T_P^{11}, and the total runtimes for these algorithms T_{ST}, T_{PT}^{11}.

situation since we are only interested in the homology of K we may avoid the construction of the blowup complex and use the cover to place a filtration on K. In particular, consider the filtration on K obtaining by ordering $\tilde{U}_i < \tilde{U}_p$ for $i \in [p-1]$. In the next section we compare our two parallel algorithms against the serial one on a series of examples.

5 Experiments

In this section, we describe the implementation of our algorithm and explore its performance on real and synthetic data. We compare our performance against our existing serial software. Our implementation is in C++ using the generic programming paradigm. We rely on the METIS library for computing graph partitions [15], the Intel Threading Building Blocks Library [19] for parallelism, and our own library for homology computation. Our parallel implementation computes an initial filtration on K, a cover U, blowup complex K^U and associated filtration in a parallel fashion, independent of the number of partitions. Our serial implementation only computes an identical initial filtration, and then homology. The *makespan* of a program is the longest running thread time. We divide the running time of our algorithm into two *phases*. We refer to the segment of code leading to homology computation as *Phase I* and the makespan of this segment the *preprocessing time*. The time spent computing homology is *Phase II*. For the sake of reproducibility, we now provide details on our experiments.

We time both parallel and serial programs with wall-clock time using the tbb::clock. Each time reported is the average makespan, measured in seconds, over 10 runs of the program. As previously mentioned all of our experiments are done using 11 cores on a 2 CPU, 12 Core, x86-64 Linux Machine, with 3.4 MHz Intel Xeon X5690 Processors, 49 GB of RAM, SSE 4.2, and we disable hyperthreading to prevent unnecessary contention between threads. We remind the reader that we only have available $p-1$ of the total p cores in order to leave room for system processes, and that in this work we use at most one thread per available core.

5.1 Data

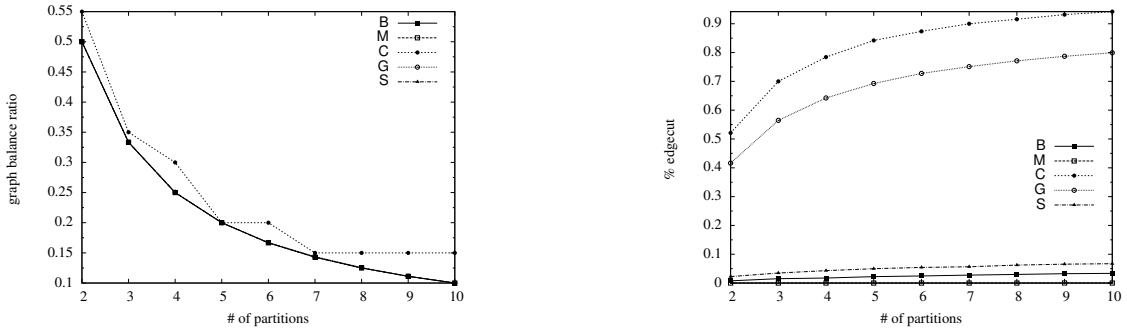
All data sets are listed in Table 1. Within Table 1 we also summarize the results of the serial and parallel algorithms within each of the two phases. All complexes are skeleta of a Vietoris-Rips Complex $\mathcal{V}_\epsilon$ [22]. Next we describe the input space for each experiment.

M is a collection of 22,720 copies of a fully connected 10 dimensional complex on 11 vertices, each copy connected to the next by a single edge. C is a fully connected complex on 19 vertices. Recall that the n-simplex has $\Theta(2^n)$ faces. B is a 3-complex built on a set of points sampled from the *Stanford bunny*. We create S, using Muller's method to sample uniformly on the unit 3-sphere [18] and then use the diagonal map $x \to (x, x)$ to embed the points in $\mathbb{R}^8$ [10]. G is a 4-dimensional clique complex built on a sparse Erdős-Rényi graph $G(n, p)$ with $n = 1250$ and $p = 0.05$.

5.2 Statistics

Recall from Section 4.2 our input is a complex K and integer $p > 1$. Our goal is to build a balanced cover for which $|K^U|/|K|$ is as small as possible. First, we build a a graph partition of the one skeleton $G(K)$. To produce our graph partition we chose the unsupervised graph partitioning algorithm METIS because it tends to produce balanced graph partitions. Figure 6(a) shows the balance ratio $\hat{\alpha} = \max_i |V^i|/|V|$ for each computed partition on each data set.

Second, given a partition we next build a cover using PARTITION-BASED-COVER. Finally, the procedure BUILD-BLOWUP-COMPLEX computes the blowup complex along with its filtration. Figure 7(a) shows $|K^U|/|K|$ for the covers computed on each data set, and Figure 7(b) shows the balance ratio of the corresponding blowup complex: $\alpha = \max_i |U^i|/|K|$.



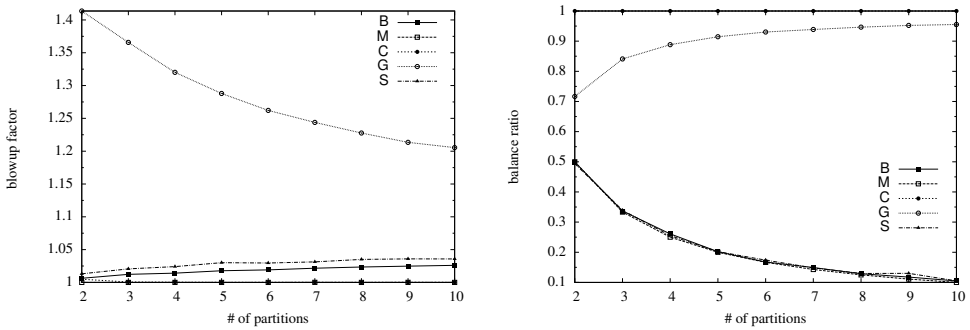
(a) Partition Balance Ratio $\hat{\alpha}$

(b) Edgecut

Figure 6: Edgecut and balance ratios computed on the 1-skeleton of each dataset.

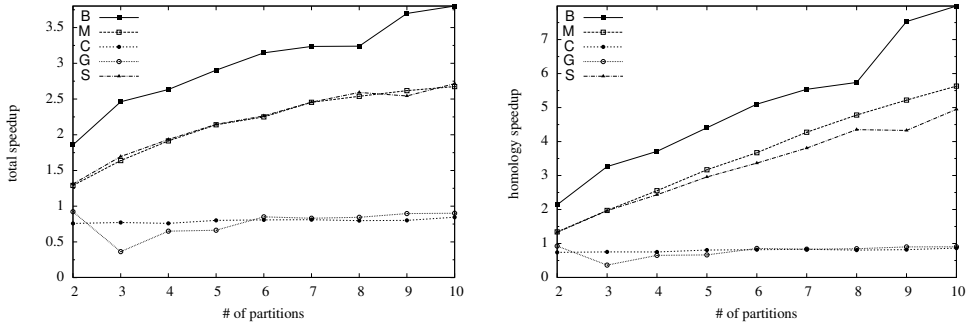
5.3 Timing

For each of our data sets we present the speedup factor of our algorithm versus serial persistence. Figure 8(a) shows the speedup factor of our entire algorithm from start to finish as compared to a



(a) Blowup Factor $|K^U|/|K|$ (b) Balance Ratio α for each covering

Figure 7: Blowup factor and cover balance ratio for each complex.



(a) Overall speedup factor (b) Speedup factor for homology computation

Figure 8: Speedup factor plots for the parallel algorithm as compared to the serial algorithm.

persistence run in serial on the same input, from start to finish. In Figure 8(b) we also compare the speedup for homology computations between both algorithms.

First, we can see that our techniques tend to scale the best on inputs in which all topological features are localized by the cover. However we see significantly better performance on B versus M. Due to the massive size of each M, we believe that the variation between these two experiments is demonstrating how memory bandwidth does not scale with the number of cores used by an algorithm. One might consider mitigating the effect of limited memory bandwith by prefetching or other architecture specific parallelization techniques, but our goal is to demonstrate proof of concept, not optimization for all cases.

Second, geometric inputs such as a S have entirely global topology; these features are necessarily resolved by a long serial computation. However, they still emit a balanced cover, so the bulk of the work is still evenly divided across each core, and we still see a linear speedup on this input.

Finally, we see that inputs such as C or G which are flag complexes of cliques or expander complexes, have entirely global topological structure, emit no balanced cover. and produce a small blowup factor. Our algorithm is slower than the equivalent serial computations which is expected.

6 Conclusion & Future Work

In this paper we present a method for computing the homology of a cellular complex in parallel. We describe each step of the method, implement all algorithms, and present preliminary experimental results. While our main goal is to compute the persistent homology of larger complexes in distributed memory we have demonstrated the ability for parallel computations based on spatial decompositions of the input to outperform serial computations.

We mentioned in Section 3 that there is room for more parallelism in MULTICORE-HOMOLOGY. For example, Let M be any matching of the d-cells of $N(U)$. One can first carry out these iterations of the for loop in parallel. Additionally, given a filtration $\leq_K$ on K and a cover U one can construct a filtration on K^U by projection. The resulting filtration produces identical persistent homology to that of $\leq_K$ on K. This filtration orders cells in the blowup first by their factor in K then breaks ties by ordering based on the factor in $N(U)$. We plan a followup paper to demonstrate parallelism using this approach in distributed memory.

ACKNOWLEDGMENTS

The authors would like to thank Gunnar Carlsson, Don Sheehy, Steve Canon, and Milka Doktorova, for discussions and support.

References

[1] U. Bauer, K. M., and J. Reininghaus. Clear and compress: Computing persistent homology in chunks. *arxiv*, 2013.

[2] D. Boltcheva, S. Merino, J.-C. Léon, and F. Hétroy. Constructive Mayer-Vietoris Algorithm: Computing the Homology of Unions of Simplicial Complexes. Rapport de recherche RR-7471, INRIA, Dec 2010.

[3] G. Carlsson. Topology and Data. *Bulletin of the American Mathematical Society*, 46:255–308, 2009.

[4] J. Dumas, F. Heckenbach, D. Saunders, and V. Welker. Computing simplicial homology based on efficient smith normal form algorithms. *Algebra, geometry, and software systems*, 177:207, 2003.

[5] B. D. S. E. Kaltofen. Personal Communication, 2012.

[6] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[7] S. Eilenberg and J. A. Zilber. Semi-simplicial complexes and singular homology. *The Annals of Mathematics*, 51(3):pp. 499–513, 1950.

[8] P. Fjallstrom. Algorithms for graph partitioning: A survey. *Computer and Information Science*, 3(10), 1998.

[9] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.5.4*, 2012.

[10] A. Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, UK, 2002.

[11] M. Joswig. Computing invariants of simplicial manifolds. *Arxiv preprint math/0401176*, 2004.

[12] T. Kaczyński, K. Mischaikow, and M. Mrozek. *Computational homology*, volume 157. Springer Verlag, 2004.

[13] E. Kaltofen, M. Krishnamoorthy, and D. Saunders. Fast parallel computation of hermite and smith forms of polynomial matrices. *SIAM. J. on Algebraic and Discrete Methods*, 8:683–690, 1987.

[14] E. Kaltofen, M. Krishnamoorthy, and D. Saunders. Parallel algorithms for matrix normal forms. *Linear Algebra and Applications*, 136:189–208, 1989.

[15] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359, 1999.

[16] D. Lipsky, P. Skraba, and M. Vejdemo-Johansson. A spectral sequence for parallelized persistence. *CoRR*, abs/1112.1245, 2011.

[17] J. May. *Simplicial objects in algebraic topology*. D. Van Nostrand Inc., Princeton, NJ, 1967.

[18] M. E. Muller. A note on a method for generating points uniformly on n-dimensional spheres. *Commun. ACM*, 2(4):19–20, 1959.

[19] C. Pheatt. Intel® threading building blocks. *Journal of Computing Sciences in Colleges*, 23(4):298–298, 2008.

[20] F. Uhlig. *Transform linear algebra*. Prentice Hall, Upper Saddle River, NJ, 2002.

[21] A. Zomorodian. Computational topology. In M. Atallah and M. Blanton, editors, *Algorithms and Theory of Computation Handbook*, volume 2, chapter 3. Chapman & Hall/CRC Press, Boca Raton, FL, second edition, 2010.

[22] A. Zomorodian. Fast construction of the Vietoris-Rips complex. *Computers & Graphics*, 34(3):263 – 271, 2010.

[23] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

[24] A. Zomorodian and G. Carlsson. Localized homology. *Computational Geometry: Theory & Applications*, 41(3):126–148, 2008.