Multicore Homology via Mayer-Vietoris

Ryan H. Lewis*

Institute for Computational Mathematics and Engineering, Huang Building, Stanford University

Afra Zomorodian

The D. E. Shaw Group, 1166 Avenue of the Americas, Ninth Floor, New York, New York

Abstract

In this work we investigate the parallel computation of homology using the Mayer-Vietoris principle. We present a two stage approach for parallelizing persistence. In the first stage, we produce a cover of the input cell complex by overlapping subspaces. In the second stage, 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. Next, we compute the homology of each subspace in the blowup complex in parallel and then glue these results together in serial. We show how to use the persistence algorithm to organize these computations. In the first stage, any algorithm can be used to produce a cover of the input complex. We describe an algorithm for producing a cover of a space with a simple structure and bounded overlap based on graph partitions. Additionally, we present a simplistic model for the problem of finding covers appropriate for parallel algorithms and show that finding such covers is NP-HARD. Finally, we present a second parallel homology algorithm. This algorithm avoids the explicit construction of the blowup complex saving space. We implement our algorithms for multicore computers, and compare them against each other as well as existing serial and parallel algorithms with a suite of experiments. We achieve roughly $8\times$ speedup of the homology computations on a 10-dimensional complex with about 46 million simplices using 11 cores.

Keywords: Computational Topology, Algorithms, Theory

1. Introduction

In this paper, we present fast multicore algorithms for computing the homology of arbitrary dimensional cell complexes over field coefficients. Figure (1) shows the speedup factor of our two algorithm for computing homology over $\mathbb{Z}_2$ coefficients of the data set M, described in Section 5. By decomposing the space into the 11 pieces visualized in Figure (1b), we are able to reduce the boundary matrix of the input space in .37 seconds, approximately eight times faster than the 3 seconds necessary for serial computation. All our timings are done on a 64-Bit GNU/Linux machine with dual, six core, 2.93Ghz Intel X5670 CPUs, and hyperthreading disabled.

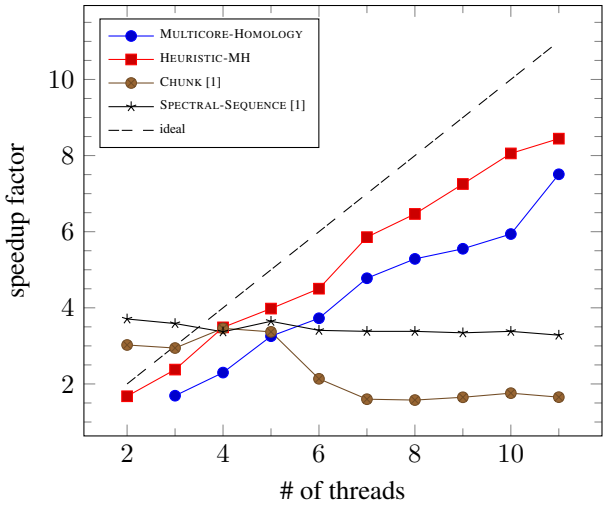
1.1. Motivation

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 [2]. The classic pipeline for topological data 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 [3, 4], captures multiscale topological structure. Computing field homology, especially over $\mathbb{Z}_2$ coefficients, is an integral part of topological data analysis.

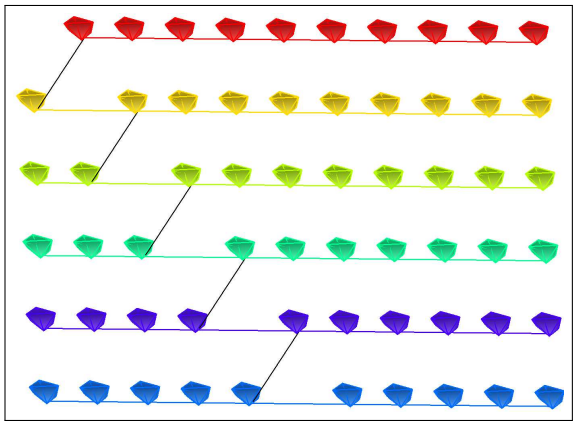
In this paper, we focus on developing a parallel algorithm to compute homology on multicore shared-memory machines. This algorithm is a first step toward a distributed-memory algorithm that will allow us to compute the persistent homology of massive structures on computer clusters.

*Principal Corresponding author

Email addresses: rhl@stanford.edu (Ryan H. Lewis), afra@cs.dartmouth.edu (Afra Zomorodian)



(a) We achieve ∼77% efficiency across 11 cores via HEURISTIC-MH and ∼64% via MULTICORE-HOMOLOGY. We also plot the speedup factor of the parallel algorithms CHUNK and SPECTRAL-SEQUENCE from Bauer et. al [1].



(b) Shown is a portion of the simply connected input space M. M contains ∼22K copies of a fully connected 10 dimensional complex on 11 vertices each connected to the next by a single edge. Each color represents a portion of 7 sets of a cover by 12 pieces.

Figure 1: On the left is the speedup in homology computation for the 10 dimensional complex with 45M simplices partially shown on the right.

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 [5]. Their software is available within the GAP software package [6]. Joswig surveys the computation of invariants, including homology for simplicial spaces with a focus on manifolds [7]. Kaczyński et al. develop heuristics to compute cubical homology [8]. Kaltofen et al. provides a theoretical investigation of randomized parallel algorithms for computing the Smith normal form [9, 10] over finite fields and $\mathbb{Q}$, however, these algorithms are not useful in practice [11].

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 this work, we decompose our input space using a *cover* so the pieces correspond to *subspaces* and their various intersections. The Mayer-Vietoris spectral sequence expresses the relationship between the homology of these subspaces to the homology of the space itself. This makes the Mayer-Vietoris spectral sequence a natural gadget to study when developing algorithms for parallel homology [12]. Merino et al. use the Mayer-Vietoris exact sequence to compute the homology of three-dimensional simplicial complexes [13]. Lipsky et al. use the Mayer-Vietoris spectral sequence in an attempt to derive a parallel algorithm [14]. Both works are theoretical in nature. The researchers do not address algorithmic issues of complexity, finding covers for input, implementations of their algorithms, or any empirical results. The Mayer-Vietoris blowup complex is the *total complex* of the Mayer-Vietoris spectral sequence. In other words the Mayer-Vietoris blowup complex is a topological space which encodes the data given as input to the spectral sequence. Its homology is equivalent to that of the original input space. Zomorodian and Carlsson show how computing homology of Mayer-Vietoris blowup complex localizes the homology basis [15].

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, et al. 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 can be computed in polynomial time and the persistence algorithm exhibits linear-time behavior in practice [3, 4]. Our framework relies on the

Mayer-Vietoris blowup complex, a spatial version of the Mayer-Vietoris spectral sequence [15]. The Mayer-Vietoris blowup complex is the *total complex* of the terms of the first page of the Mayer-Vietoris spectral sequence and its 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 its homology in parallel using the persistence algorithm. We note that while we restrict our attention to field homology our software could be modified to produce $\mathbb{Z}$-valued homology.

Our approach has two stages. In the first stage, we find a cover of the input space. In the second stage, we use this cover to build the blowup complex and compute its homology in parallel. The homology computation within the second stage may be viewed as the 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 computations are the same as their non-relative siblings that is, they do not need to be further reduced against each other.

Since the first stage of the 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 may avoid building the blowup complex by using the cover to generate a new filtration on the original space for carrying out parallel computations without the blowup complex.

In Section 5, we present the results of a suite of experiments using a multicore version of our parallel algorithms and provide experimental results. All of the techniques in this paper are deterministic. Our software and our datasets are publicly available.

2. Background

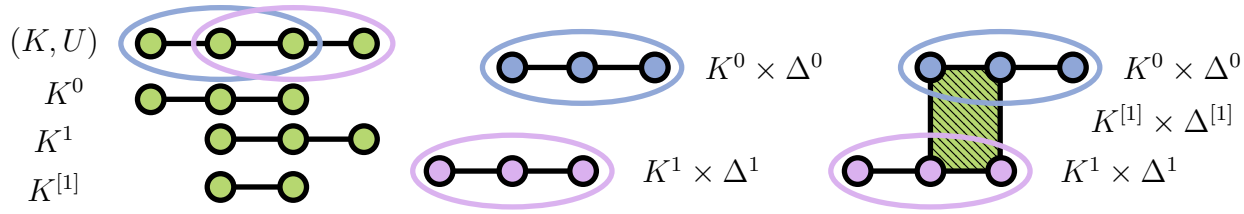
We begin with a review of simplicial complexes, homology, and blowup complexes. We refer the read to Hatcher for background material in algebraic topology [12]. and to Zomorodian [16, Chapter 13] for computational topology. In principle the methods outlined in this paper generalize to any type of cellular 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 [15]. 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 *vertex separator* of a graph G is a set of vertices I such that the removal of I from G results in a disconnected graph.

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 $\mathrm{Cl}\,(L) = \{\tau \mid \tau \subseteq \sigma \in L\}$ and is a simplicial complex. 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* [12], 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 when each cover set is closed we call the cover a *closed cover* U. 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 $N(\sigma)$ the simplex in $N(U)$ which lists the cover sets containing σ.



(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).

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 [17, 12, 8, 18]. 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 [12]. 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 $\operatorname{im}(.)$ 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 homology, we begin by defining the action of the boundary operator 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* [19]. In practice, we use the *persistence algorithm* [3, 4]. This algorithm can compute the homology of any *based persistence complex* [15], 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. The algorithm proceeds by determining if the addition of a cell into the complex creates a new homology class or annihilates a homology class previously created. The result is a pairing between cells which create homology and the corresponding cell which destroy's that homology. Except, if a homology class is never killed, in which case it is left unpaired. β_i is the number of unpaired i-cells.

We focus on characterizing the three inputs needed 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 [15], but in this paper we focus on blowups of simplicial complexes. For a longer exposition of the Mayer-Vietoris blowup complex we refer

the reader to Zomorodian & Carlsson [15]. Given a simplicial complex K and cover $U = \{U_i\}_i$ of n subcomplexes, let $K^J = \bigcap_{k\in J} U_j$. 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 [15] 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 (2a), 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 (2c). 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 [15, 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 $\sigma \times \tau$, where $\sigma \in K$ and $\tau \in N(U) \subseteq \Delta^{[n]}$. For example, the product of two edges, $bc \times 01$, gives us a quadrilateral cell in Example 2.3.1. While we may then triangulate the blowup complex to get a simplicial complex in order to compute its homology, this is computationally prohibitive, due to the need for triangulation. Luckily this approach is also not necessary. 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 $\dim \sigma + \dim \Delta^J = n$. The notation $\otimes$ denotes *tensor product*. Recall that the tensor product of two vector spaces is obtained by taking a quotient of the free vector space on the cartesian product [12, Page 218]. We define the boundary operator as [15, Lemma 4]:

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

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 $bc \otimes 01$: 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 of the blowup complex, 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 filtration of the blowup complex has two phases, the *local* and the *global* phase. In the local phase, the complex explodes into multiple pieces, representing the disjoint union of each set in the cover, as in Figure (2b). This means that we have potentially multiple versions of a simplex if it lies in an intersection of two sets in the cover. For example,

```
MULTICORE-HOMOLOGY(K, p)
1  U ← COVER(K, p)
2  K^U ← BUILD-BLOWUP-COMPLEX(K, U)
3  parallel for Δ^J ∈ N(U)
4       do PAIR-CELLS(K^J × Δ^J)[1]
5  for d > 0
6       do for Δ^J ∈ N(U) a d-cell.
7            do PAIR-CELLS(Cl(K^J × Δ^J))
```

```
BUILD-BLOWUP-COMPLEX(K, U)
1  K^U ← ∅
2  parallel for σ ∈ K
3       do for τ ⊆ U[σ]
4            do K^U ← K^U ∪ (σ × τ)
```

Figure 3: Psuedocode for computing the blowup complex and its homology in parallel. COVER can be any algorithm for generating a cover of K by p subspaces. The procedure PAIR-CELLS is defined in the Computational Topology section of the Algorithms and Theory of Computation Handbook [16, Page 3-17].

since edge bc falls within both sets in the cover in Figure (2a), 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 (2c), the cell $b \times 01$ connects $b \times 0$ and $b \times 1$.

To describe this 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 (2c) has the following filtration:

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

The Algorithm in Figure (3) shows how to build the blowup complex and compute its 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.

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.

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 lead to efficient computation. We state an optimization problem over covers which minimizes the size of the blowup of a complex. We then show that this optimization problem is 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 the 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.

[1] When the list of cells given as input to PAIR-CELLS is not a sub complex computation should be interpreted as relative homology computation by ignoring elements of the boundary which are not given in the input.

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 cover set in parallel, the number of cover sets should be the number p of available processors. Finally, each cover set 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 $\alpha \in (\frac{1}{p}, 1)$. Putting together all of the desired properties of blowups, we have the following optimization problem stated for $p = 2$ and $\alpha \in (\frac{1}{2}, 1)$:

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 its 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 be a cover K of size $p > 1$. Suppose that the intersection of any three sets in U vanishes. Then*

$$|K^U|/|K| = 1 + 2\frac{|I|}{|K|}.$$

where $I = \bigcup_{i \neq j} U_i \cap U_j$.

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 is 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. In other words given a vertex separator of a graph G we may view it as a cover of that graph and vice versa. The equivalent problem for vertex separators is 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:

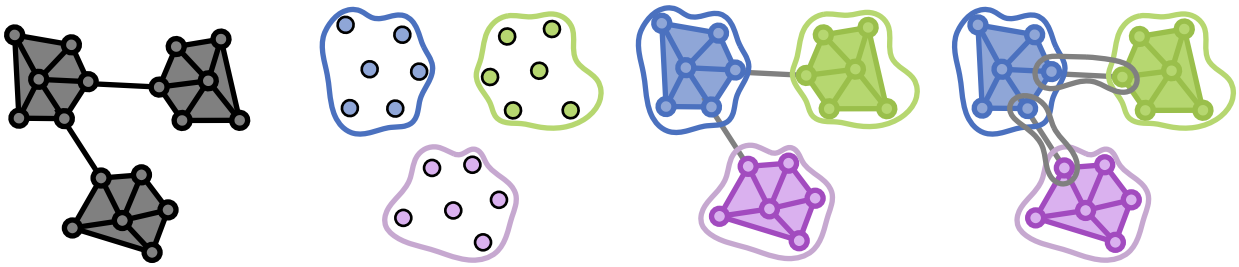
$$|I| \text{ is minimized subject to } \max_i (|V_i| + |E_i|) + |I| \leq \alpha(|V| + |E|)$$

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 its decision problem variant is NP-COMPLETE [20].

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

Proof. By restricting α-BALANCED-MINIMUM-BLOWUP and α-SUBGRAPH-BALANCED-VERTEX-SEPARATOR are equivalent when the former is restricted to graph instances. □

This procedure shows us that finding covers of graphs with bounded overlap also identifies partitions of that graph. In the next section we show how given a complex K and a partition of its 1-skeleton one can produce a cover of the entire complex with bounded overlap.



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

Figure 4: Our heuristic algorithm for 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).

4.2. Partition-Based Covers

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.

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 covers any procedure for generating covers suffices. In many situations there might 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 methods for cover enumeration, *random ϵ-balls* and *tilings* [15]. For complexes embedded in a low dimensional space one might consider algorithms based on *Voronoi diagrams* or when the data is available by level sets of *Morse functions*. However, in the general setting it is possible to generate 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 seem to fall into four major classes of algorithms: geometric, non-geometric, spectral, and hybrid methods [21]. Hybrid methods mix the techniques of the other three. In practice, we use METIS, a hybrid method, since it tends to produce balanced partitions quickly [22]. Of course any partitioning scheme will work. Next, we describe OPEN-COVER(K, P), which extends a partition of G to an open 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 its vertices lie in P_i and is added to $\tilde{U}_p$ otherwise.

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

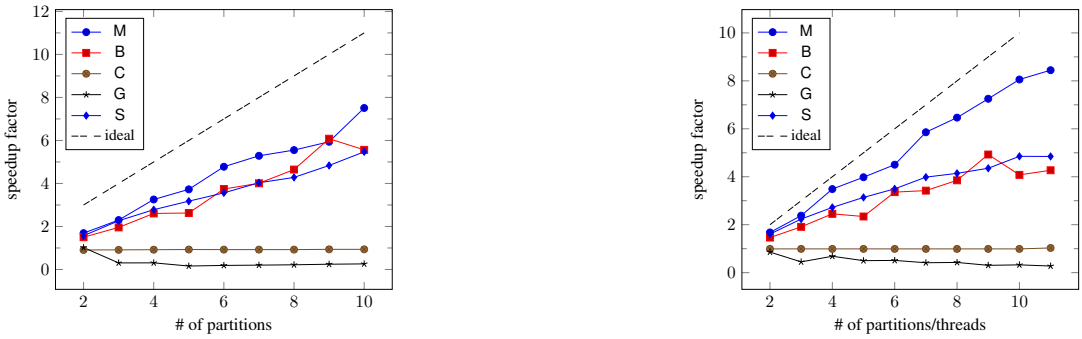
Lemma 3. *Given a complex K, $p \geq 2$,* PARTITION-BASED-COVER(K,p) *generates a cover U with $|K^U|/|K| < 3$.*

Proof. For a complex K and $p \geq 2$ let U be the cover of K by $p+1$ subcomplexes output by 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 intersections. It follows by Lemma 1 that $|K^U|/|K| < 3$. □

Since we are interested only in the homology of K and not it's persistent homology we may avoid the construction of the blowup complex and use the open cover generated to place a filtration on K. In particular, consider the filtration on K obtained by ordering $\tilde{U}_i < \tilde{U}_p$ for $i \in [p-1]$. It is clear that before including $\tilde{U}_p$ the complex is again disconnected and thus these columns of the matrix may be reduced in parallel. Finally, we reduce this last set of columns against the columns from the first p cover sets. We call this procedure HEURISTIC-MH.

In the next section we compare these two parallel algorithms against the standard serial algorithm as well as the algorithm CHUNK of Bauer et. al on a series of examples. The CHUNK algorithm is based on the spectral sequence of a filtration [1].

5. Experiments



(a) Speedup factor for reducing ∂_{K^U}

(b) Speedup factor for reducing ∂_K

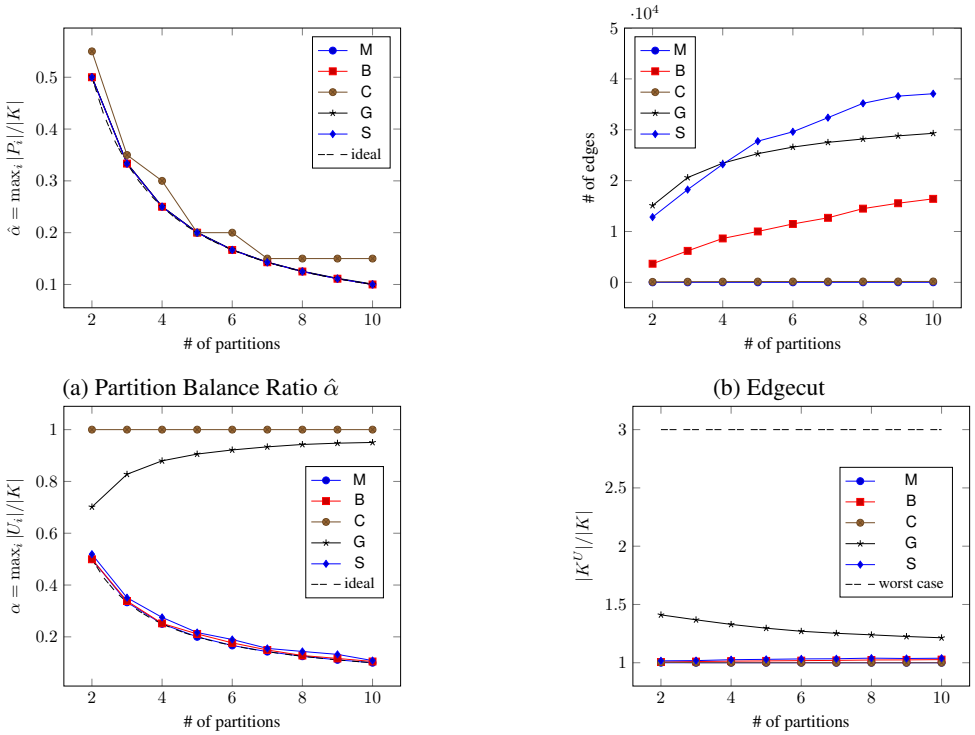
Figure 6: (Left) Speedup factor T_s/T_p where T_p is the time to reduce ∂_{K^U} in parallel on $p+1$ threads and T_s is the time to reduce ∂_K in serial. (Right) Speedup factor of T_s/T_p where T_p measures the time to reduce ∂_K in parallel.

In this section, we describe the implementation of our algorithms and explore their performance on real and synthetic data. We compare our performance against our existing serial software as well as the Persistent Homology Algorithm Toolbox (PHAT) [1]. Our implementation is in C++ using the generic programming paradigm. We rely on the METIS library for computing graph partitions [22], the Intel Threading Building Blocks Library [23] for parallelism, and our own library for homology computation. Our parallel implementation of MULTICORE-HOMOLOGY computes an initial filtration on K, a cover U, builds a blowup complex K^U with its associated filtration [in parallel], and then reduces ∂_{K^U}. For HEURISTIC-MH we reduces a permuted ∂_K, instead of building K^U. Unlike the psuedocode for MULTICORE-HOMOLOGY when reducing ∂_{K^U}, our implementation reduces the columns corresponding to cells of the form $\sigma \times \tau$ with $\dim(\tau) > 0$ in serial after the parallel reduction of all other cells. Preliminary experiments suggested that this added parallelism would not produce speedup. Our serial implementation only computes an identical initial filtration, and then reduces ∂_K.

We now provide details on how these experiments were carried out. As previously mentioned all of our experiments are done using 11 cores on a 2 CPU, 12 Core, x86-64 Linux Machine, with 2.93 GHz Intel Xeon X5670 Processors, 74 GB of RAM, and hyperthreading disabled. We time both parallel and serial programs in wall-clock time using the tbb::clock. We measure the total amount of memory requested by a process, its *resident set size*, via the *process filesystem*. This is an upper bound on the total memory used. Each time measured is the *makespan* or longest running thread time within a section of code. Time is always reported in seconds, and all reported measurements are averaged over 10 trials. We remind the reader that while we may spawn p threads we only ever have at most $p-1$ of the

Input Statistics					
D	$\lvert D\rvert$	ϵ, p	$\lvert E\rvert$	d	$\lvert K\rvert$
M	249,920	-	1,272,319	10	46,530,559
C	20	-	190	19	1,048,575
B	34,837	0.05	489,876	3	9,714,912
S	50,000	0.18	546,388	8	19,134,612
G	1250	0.047		4	73,309

Table 1: Input Statistics: The name D, and number of vertices of each data set $|D|$, as well as input parameter ϵ or p in the case of a random graph, embedding dimension $d = \dim D$, size $|K|$, and edge-set size $|E|$ of each complex K.



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

(b) Edgecut

(c) Balance Ratio for U

(d) Blowup Factor

Figure 7: Statistics for partitions and covers generated

total p cores in order to leave room for system processes. In this work we use at most one thread per available core. When running PHAT we used the latest stable version 1.4 and the "vector vector" option as this is the same basic data structure we use in our library. All software has compiled with gcc and optimizations enabled.

5.1. Data

We summarize each data set in Table 1. All complexes are skeleta of a Vietoris-Rips Complex [24]. Next, we describe the input space for each experiment. Recall that M is a collection of 22,720 copies of a fully connected 10 dimensional complex on 11 vertices, organized into 10 groups of 2,272, with each copy within a group connected to the next by a single edge, and each group connected to the next by a single edge as shown in Figure (1b). C is a fully connected complex on 19 vertices. Recall that $\Delta^{[n]}$ has $\Theta(2^n)$ faces. B is a 3-complex built on a set of points sampled from the *Stanford bunny*. We create S by using Muller's method [25] to sample uniformly on the unit 3-sphere and then use the diagonal map $x \to (x, x)$ to embed the points in $\mathbb{R}^8$ [12]. G is a 4-dimensional clique complex built on a sparse Erdős-Rényi graph $G(n, p)$ with $n = 1250$ and $p = 0.047$.

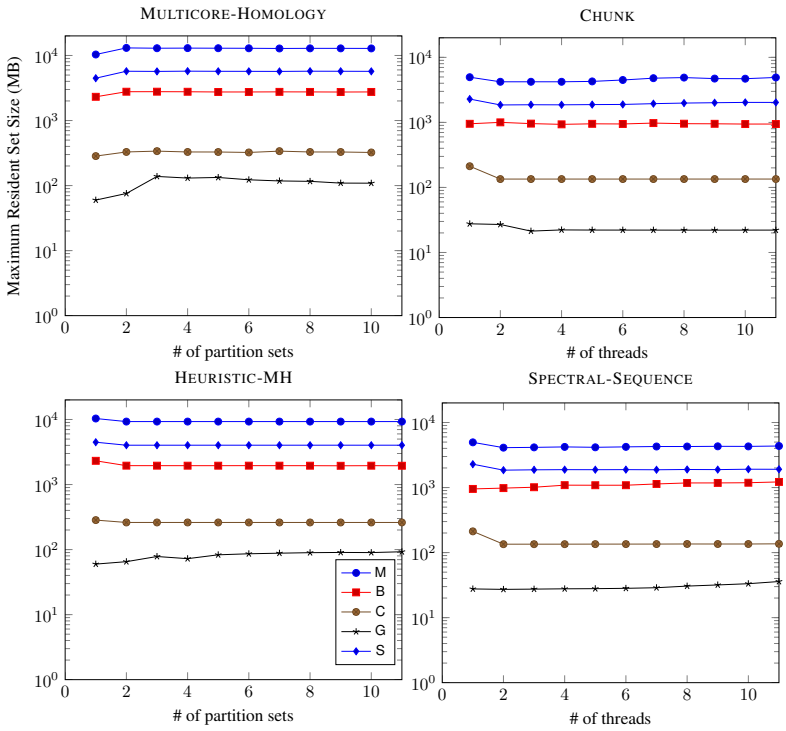


Figure 8: Total memory usage for each algorithm. Recall that PHAT takes as input a boundary matrix whereas the procedures outlined in this work take as input a simplicial complex and generates an identical boundary matrix before reducing it. At $x = 1$ on all plots we display the memory used for the standard algorithm from the appropriate software package.

5.2. Statistics

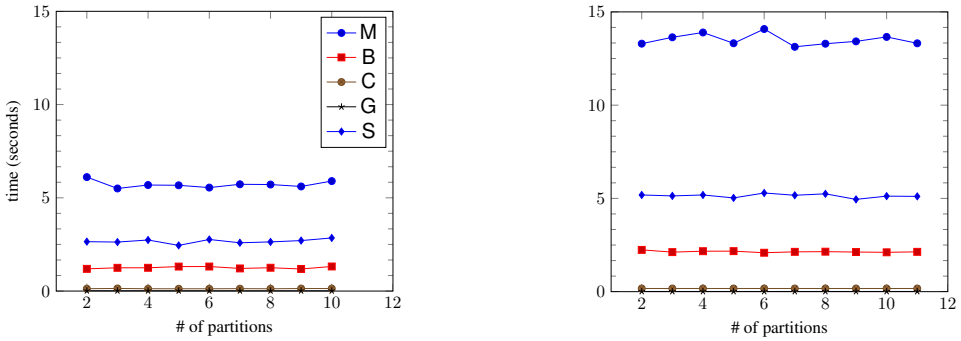
Recall from Section 4.2 that our input is a complex K and integer $p > 1$. Our goal is to build a balanced cover for which $|K^U|/|K|$ is as small as possible. First, we build a a graph partition of the one skeleton $G(K)$. To produce our graph partition we chose the unsupervised graph partitioning algorithm METIS because it tends to produce balanced graph partitions. In Figure (7a) we show the balance ratio $\hat{\alpha} = \max_i |V_i|/|V|$ for each partition produced by METIS. Next, we complete our graph partition into a cover. Figure (7c) shows the balance ratio $\alpha = \max_i |U_i|/|K|$ for covers produced by: PARTITION-BASED-COVER. Finally, the procedure BUILD-BLOWUP-COMPLEX computes the blowup complex along with its filtration. In Figure (7d) we plot $|K^U|/|K|$. Recall that covers produced by PARTITION-BASED-COVER have $|K^U|/|K| < 3$ and in general for n sets this ratio is at worst $O(2^n)$.

5.3. Timing & Measurements

For each of our data sets we present the speedup factor of our reduction algorithm versus serial persistence in Figure (6).

First, we can see that our techniques tend to scale the best on inputs in which all topological features are localized by the cover. For example, we see the best performance on M. This is not surprising since for any $p \in [2, 10]$ this complex exhibits a partition-based cover which balances its 46.5M simplices nearly perfectly while maintaining that the size of all intersections between all sets is exactly $p - 1$. Second, geometric inputs such as B and S have entirely global topology; These global topological features are resolved by reducing a handful of columns in the portion of the computation that is executed serially. However, these inputs still emit balanced covers, so we see speedup since overall the bulk of the work is roughly evenly divided across each core. Finally, we see that inputs which are flag complexes of cliques or expander graphs, such as C or G, emit no balanced cover and all covers seem to result in a large blowup complex. As expected our parallel algorithms exhibit no speedup on these inputs.

We observe that with the exception of G the parallel reduction of the boundary matrix for the blowup complex runs in time similar to the parallel reduction of the permuted boundary matrix. However there is overhead to each approach. Both algorithms require the computation of a cover. On one hand, to reduce ∂_{K^U} we must first build K^U and its associated filtration. However in HEURISTIC-MH we must construct a new filtration on K. Recall that



(a) Time to build K^U with p partition sets.

(b) Time to re-filter K

Figure 9: Comparison of the time to build a blowup complex in $O(\frac{m}{p} + p)$ time versus re-filter the base complex in $O(\frac{m}{p} \log m)$.

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. The procedure BUILD-BLOWUP-COMPLEX is implemented as a variant of the PREFIX-SUM algorithm [26]. In particular this means that BUILD-BLOWUP-COMPLEX produces the filtration of the blowup complex along with the complex itself. Aside from its output BUILD-BLOWUP-COMPLEX only uses $O(p)$ extra space. When avoiding the blowup complex we do so by creating a new filtration in $O(\frac{m}{p} \log m)$ where $m = |K|$ and p is the total number of available threads.

Figure (9) compares the running time of BUILD-BLOWUP-COMPLEX against the time to re-filter K. From the standpoint of memory consumption it is clear that the blowup avoiding algorithm is a better choice. However, when the resulting blowup complex is similar in size to the original space, It may be possible to significantly improve overall running time by building the blowup complex simply because the process of sorting may end up being slower than building the blowup.

We end this section by comparing the Mayer-Vietoris algorithm to CHUNK and SPECTRAL-SEQUENCE algorithms available in PHAT. SPECTRAL-SEQUENCE and CHUNK are parallel implementations of the spectral sequence algorithm based on the spectral sequence of a filtration [1]. We plot the time to reduce ∂_K and ∂_{K^U} with p threads versus the time for the each algorithm from PHAT to reduce ∂_K in Figure (10). Figure (8) compares the total memory usage for these algorithms. Recall that PHAT takes as input a description of ∂_K whereas for our experiments we read in as input K and then build and reduce ∂_{K^U}. While the implementation of the chunk algorithm in PHAT can be significantly faster than its implementation of the standard algorithm, their algorithms do not always seem to scale with the number of available threads. Our experiments suggest that the algorithms provided in PHAT attain speedup mainly due to the out of order nature of their reductions. The two optimizations used in these algorithms significantly reduces the total work required as compared to the serial algorithm, but these optimizations do not seem to help scalability. Practically, this software is still in the early stages of development, so we expect future versions to be more competitive.

6. Conclusion & Future Work

In this paper we presented two methods for computing homology in parallel. We describe each step of both methods, 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.

There are many avenues for future research. The nerve of the covers generated in this paper have are a star graph. It would be useful to be able to generate covers whose nerve has higher topological features. For example, if the nerve was a cycle then we could take advantage of added parallelism when reducing the corresponding cells in the blowup complex. The partition based covers are akin to a bottom up approach to cover generation. A top down algorithm which operates by partitioning the maximal cells might have better performance on datasets where a small separator is non existent or difficult to find. It would be of clear interest to have an approximation algorithm to the problem discussed in this work or to a variant thereof. It would also be of interest to combine the algorithms outlined in this works with the ones from PHAT. In particular, each piece of the boundary matrix produced by a Mayer-Vietoris style algorithm could be further reduced via these alternative approaches.

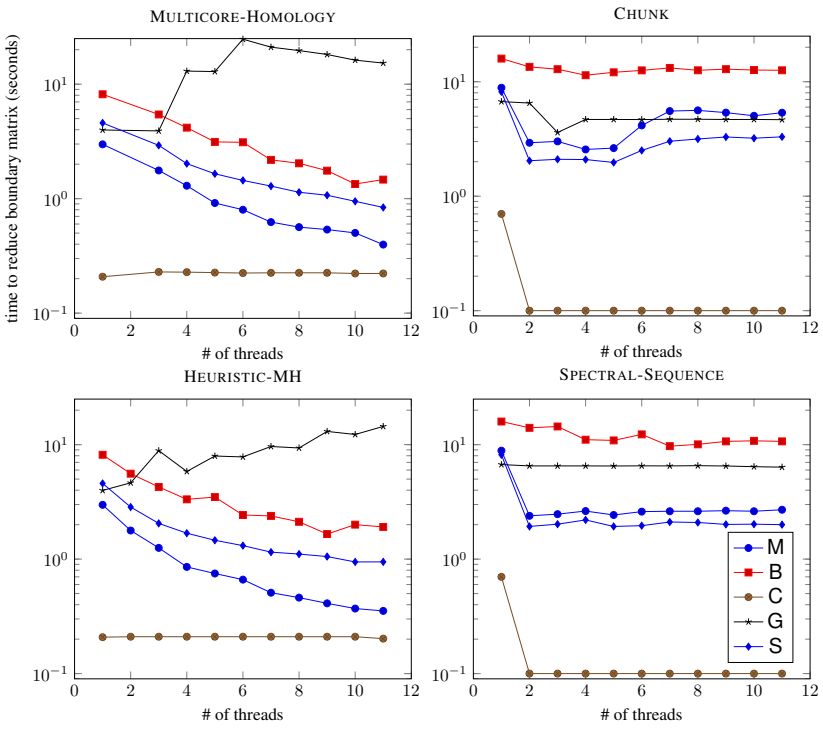


Figure 10: Time to reduce the boundary matrix for each algorithm. At $x = 1$ on all plots we display the running time for reducing ∂_K using the standard algorithm from the appropriate software package.

It is possible to filter the blowup complex to have identical persistent homology to that of a filtration $\leq_K$ of an input complex K. Given a filtration $\leq_K$ on K and a cover U one can construct a filtration on K^U by restriction of the cover to the each subspace in the filtration. One can now use this data to construct a filtration of blowup complexes. The resulting filtration produces identical persistent homology to that of $\leq_K$ on K. At a chain level, this amounts to ordering product cells first by their factor in K, breaking ties using the second factor. Recall that in this work we ordered product cells first by the second factor, breaking ties using the first factor. While, it is no longer straightforward to carry out the persistence algorithm in parallel as described in this work, it is possible to compute the persistent homology of this filtration in parallel. We leave the details to a followup paper.

Acknowledgments & Bibliography

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

Bibliography

[1] U. Bauer, M. Kerber, J. Reininghaus, Clear and compress: Computing persistent homology in chunks, in: P.-T. Bremer, I. Hotz, V. Pascucci, R. Peikert (Eds.), Topological Methods in Data Analysis and Visualization III, Mathematics and Visualization, Springer International Publishing, 2014, pp. 103–117. doi:10.1007/978-3-319-04099-8_7.
URL http://dx.doi.org/10.1007/978-3-319-04099-8_7

[2] G. Carlsson, Topology and Data, Bulletin of the American Mathematical Society 46 (2009) 255–308.
URL http://www.ams.org/bull/2009-46-02/S0273-0979-09-01249-X/home.html

[3] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete & Computational Geometry 28 (4) (2002) 511–533. doi:10.1007/s00454-002-2885-2.

[4] A. Zomorodian, G. Carlsson, Computing persistent homology, Discrete & Computational Geometry 33 (2) (2005) 249–274. doi:10.1007/s00454-004-1146-y.

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

[6] The GAP Group, GAP – Groups, Algorithms, and Programming, Version 4.5.4 (2012).
URL http://www.gap-system.org

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

[8] T. Kaczyński, K. Mischaikow, M. Mrozek, Computational homology, Vol. 157, Springer Verlag, 2004.

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

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

[11] B. D. S. E. Kaltofen, Personal Communication (2012).

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

[13] D. Boltcheva, S. Merino, J.-C. Léon, F. Hétroy, Constructive Mayer-Vietoris Algorithm: Computing the Homology of Unions of Simplicial Complexes, Rapport de recherche RR-7471, INRIA (Dec 2010).
URL http://hal.inria.fr/inria-00542717/en/

[14] D. Lipsky, P. Skraba, M. Vejdemo-Johansson, A spectral sequence for parallelized persistence, CoRR abs/1112.1245.
URL http://arxiv.org/abs/1112.1245

[15] A. Zomorodian, G. Carlsson, Localized homology, Computational Geometry: Theory & Applications 41 (3) (2008) 126–148. doi:10.1016/j.comgeo.2008.02.003.

[16] A. Zomorodian, Computational topology, in: M. Atallah, M. Blanton (Eds.), Algorithms and Theory of Computation Handbook, 2nd Edition, Vol. 2, Chapman & Hall/CRC Press, Boca Raton, FL, 2010, Ch. 3.

[17] S. Eilenberg, J. A. Zilber, Semi-simplicial complexes and singular homology, The Annals of Mathematics 51 (3) (1950) pp. 499–513.
URL http://www.jstor.org/stable/1969364

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

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

[20] R. H. Lewis, Yet another graph partitioning problem is NP-hard, CoRR abs/1403.5544. arXiv:1403.5544.
URL http://arxiv.org/abs/1403.5544

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

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

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

[24] A. Zomorodian, Fast construction of the Vietoris-Rips complex, Computers & Graphics 34 (3) (2010) 263 – 271. doi:10.1016/j.cag.2010.03.007.

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

[26] C. Breshears, The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel, Applications, O'Reilly Media, 2009.