Hardware Acceleration of Persistent Homology Computation

Fan Wang[1(✉)], Chunhua Deng[2], Bo Yuan[2], and Chao Chen[1]

[1] Stony Brook University, Stony Brook, NY 11794, USA
fan.wang.1@stonybrook.edu
[2] Rutgers University, New Brunswick, NJ 08901, USA

Abstract. As a powerful tool for topological data analysis, persistent homology captures topological structures of data in a robust manner. Its pertinent information is summarized in a persistence diagram, which records topological structures, as well as their saliency. Recent years have witnessed an increased interest of persistent homology in various domains. In biomedical image analysis, persistent homology has been applied to brain images, neuron images, cardiac images and cancer pathology images. Meanwhile, the computation of persistent homology could be time-consuming due to column operations over a large matrix, called the boundary matrix. This paper seeks to accelerate persistent homology computation with a hardware implementation of the column operations of the boundary matrix. By designing a dedicated hardware to process fast matrix reduction, the proposed hardware accelerator could potentially achieve up to 20k–30k times speed-up.

Keywords: Topology data analysis · Persistent homology · Matrix operation · Hardware acceleration

1 Introduction

Topological Data Analysis studies topological structures such as connected components, handles and voids, which characterize data in a global, intuitive, and robust manner. In particular, the theory of persistent homology [8,10] captures topological properties of data through the view of a *filter function*, i.e., a scalar function such as image intensity, density function, etc. One may threshold the domain with certain threshold and inspect the *sublevel set*, namely, regions whose filter value is below the threshold. Persistent homology inspects a series of nested sublevel sets induced by different thresholds, called a filtration, and tracks the birth and death of different topological structures. The information is summarized in a persistence diagram, which is a set of points on a 2D plane whose x and y coordinates are topological structures' birth and death time respectively. See Fig. 1 for an example in which persistent homology is computed on a sample image from the MNIST dataset [13]. The sublevel sets corresponding to different function values (t_0 to t_6) are displayed with black pixels in the top row.

L. Zhou et al. (Eds.): LABELS 2019/HAL-MICCAI 2019/CuRIOUS 2019, LNCS 11851, pp. 81–88, 2019.
https://doi.org/10.1007/978-3-030-33642-4_9

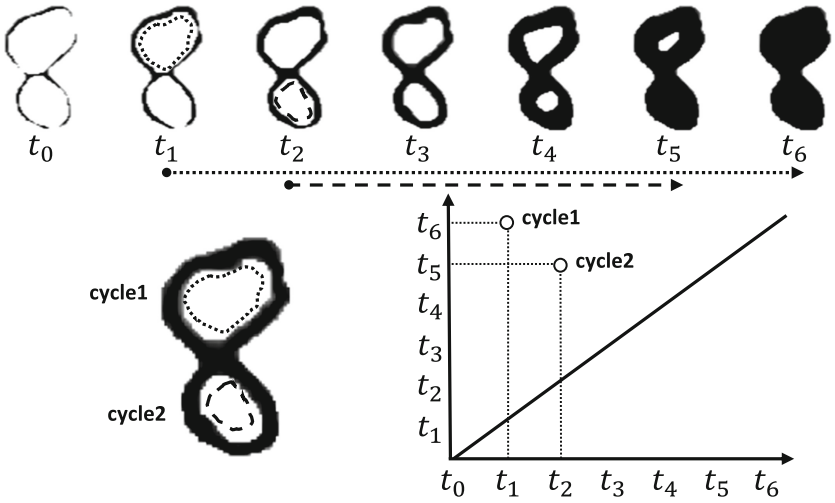

Fig. 1. Example of a persistence diagram (lower right) computed on an image (lower left) taken from the MNIST dataset. We use the inverse of the original image as input. The sublevel sets and the filtration are shown in the top row.

The upper cycle of digit 8 is created at $t1$ while the lower cycle forms at t_2. Both cycles are eventually filled in with black pixels (and thus disappear) at t_5 and t_6 respectively.

Numerous topology inspired methods have been proposed in recent years and they have been successfully applied to different problems, including molecular biology [4,12], signal analysis [18], sensor networks [11], robotics [19], shape recognition [15], graphics [7], geometric modeling [9] and many more. In biomedical image analysis, topological methods have been used but not limited to analyze global structures of sMRI and functional MRI images [1,2,14]. Topological invariant is by design robust to deformation and to noise. Without any tearing or gluing, topological structures will be preserved regardless of the deformations. A desirable property of persistence diagrams is that they are Lipschitz with respect to the underlying filter function [3].

An essential component involved in the computation of persistent homology is the reduction of a boundary matrix whose columns and rows represent elements of a discretization of a domain. Simplicies of zero, one, two and three dimensions are vertices, edges, triangles and tetrahedra (Fig. 2). A boundary matrix ∂ is a binary matrix with entry $\partial(u, v) = 1$ when simplex σ_u corresponding to row u belongs to boundary of σ_v corresponding to column v. A reduction of the boundary matrix reduces it into a canonical form through column addition operations over binary field. The number of column operations required by the reduction process is usually huge, because the boundary matrices can be prohibitively large even with input images of small sizes. As an example, digit images of resolution 28×28 from MNIST have 1D boundary matrix of size 784×1512 and 2D

boundary matrix of size 1512×729. Boundary matrix reduction has become a major bottleneck for persistent homology computation, and impedes its further applications. Therefore, it is imperative to accelerate the reduction process by dedicated hardware. Some hardware accelerators can speedup the computation process hundreds or even thousands of times compared to general purpose CPU and GPU [5,6]. By designing dedicated hardware to implement boundary matrix reduction, we can accelerate it up to 20000 times. To the best of our knowledge, this is the first paper proposing hardware accelerations for persistent homology computation.

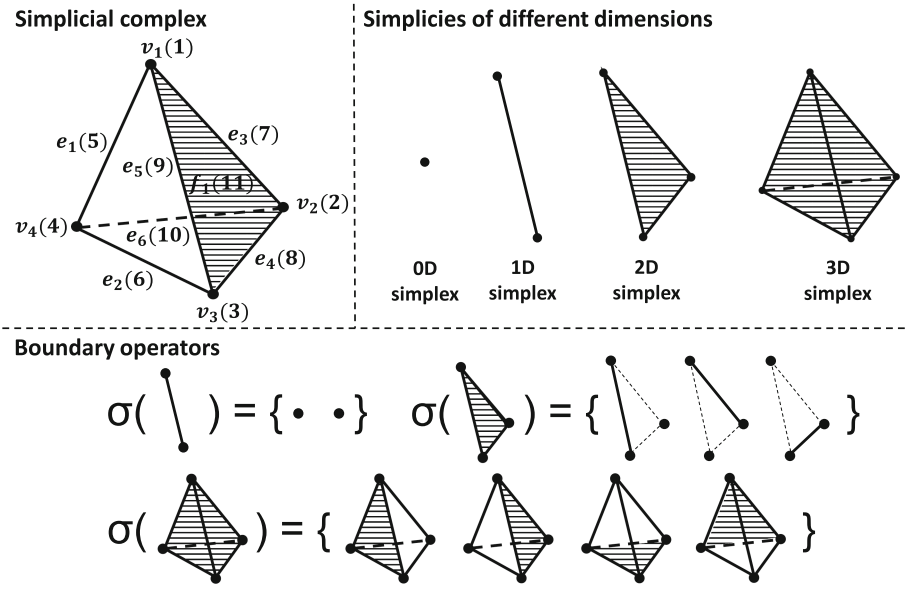

Fig. 2. Top left: a simplicial complex with filter function values marked in the parentheses beside corresponding vertices, edges, and faces; Top right: simplicies of dimension 0, 1, 2, and 3; Bottom row: examples of boundary operators on 1-, 2-, and 3-dimensional simplicies.

The remainder of the paper is organized as follows. We briefly explain the basics of the theory of persistent homology in Sect. 2. Details concerning boundary matrix and boundary matrix reduction are provided in Sect. 3. Lastly, Sect. 4 presents a hardware implementation which considerably accelerates the reduction process. Potential speedups from proposed hardware implementation are evaluated on two datasets, namely, MNIST and The Mammographic Image Analysis Society (MIAS) [20]. For illustration purpose, MIAS dataset is downsized from 1024×1024 to 32×32 with aspect ratio intact.

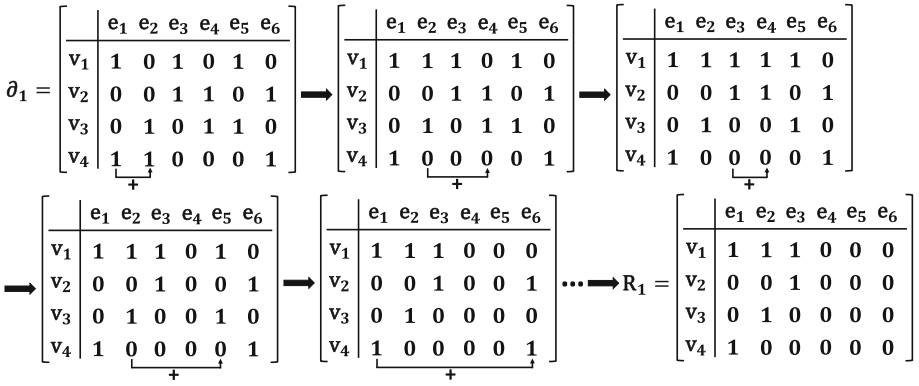

Fig. 3. ∂_1 is the 1−dimensional boundary matrix computed from the simplicial complex (with a filter function defined on it) illustrated in top left of Fig. 2. Rows and columns of ∂_1 correspond to vertices (0-simplicies) and edges (1-simplicies) respectively. First few steps of boundary matrix reduction are shown. The reduction process stops at R_1 which is the reduced result of ∂_1.

2 Persistent Homology

We review some of the basic concepts necessary to understand the idea of this paper, including simplex, simplicial complex, boundary operator, and filtration. Due to space limitations, some theories as important, such as cycle, chain group, homology group, etc., are intentionally skipped. Interested readers may refer to [10,16,17] for more details.

Simplicial Complex. A d-dimensional simplex σ is the convex hull of $d+1$ affinely independent vertices. In case of 3D data, the 0-, 1-, 2-, and 3-simplex are vertex, edge, triangle and tetrahedron respectively (top right of Fig. 2). A simplicial complex K is a finite set of simplicies satisfying two conditions: (1) any face of a simplex in K is also in K; (2) intersection of any two simplicies in K is either empty or is a face for both simplicies.

Boundary Operator. The boundary of a d-simplex is the formal sum of the $(d-1)$-simplicies which are faces of the d-simplex. In the second row of Fig. 2, the boundary of an edge (1-simplex) is the sum of its two endpoints (0-simplex). The edges constituting the triangle form the boundary of that triangle. And similarly, the formal sum of the four triangles is the boundary of the tetrahedron. The boundary operator is defined on individual simplicies as an operator that decomposes a d-simplex into its boundary comprising of a set of $(d-1)$-simplicies.

Filtration. Given a topology space X and a real-valued function f defined on X, we can construct a sublevel set $X_t = \{x \in X : f(x) \leq t\}$ where t is a

threshold controlling the "progress" of sublevel sets. As t increases from $-\infty$ to $+\infty$, a sequence of sublevel sets is produced in which the first is an empty set while the last covers the whole topology space X. This increasing sequence of sets is called a filtration induced by function f.

Algorithm 1. Boundary matrix reduction

```
1: procedure INITIALIZATION
2:     R ← boundary matrix ∂
3:     low_R() ← −1
4: for i = 1 to n do
5:     if column i has 1 then
6:         low_R(i) ← row index of the last 1 in column i of R
7:     endif
8: endfor
9: for i = 1 to n do
10:    while ∃i' < i with low_R(i') = low_R(i) do
11:        add column i' to column i
12:        update low_R(i)
13:    endwhile
14: endfor
```

3 Boundary Matrix Reduction

Computation of persistence diagram requires a filter function defined on simplicies. As can be seen from top left of Fig. 2, filtration function values are marked beside corresponding simplicies (vertices, edges, and faces). With the simplicies sorted usually in increasing order according to their function values, a boundary matrix ∂ can be computed by encoding the boundary operator in a binary matrix. An entry $\partial(u, v) = 1$ when simplex σ_u corresponding to column u is part of the boundary of simplex σ_v corresponding to column v.

Boundary matrix reduction reduces ∂ to another binary matrix R through column operations performed on ∂ from left to right. During each operation, a new column is reduced by addition with a potentially already reduced column from its left. The reduction process finishes when the rightmost column of R has index of nonzero entry as small as possible (or as high as possible in terms of matrix position) or the rightmost column is zero. To better explain the reduction, we define $low_R(i)$ to be the row index of the last 1 in column i of R or -1 in case that column i is zero. To reduce column i, we keep searching for another column j satisfying condition $low_R(i) = low_R(j), j < i$ and adding column j to column i until i is zero or no column j satisfying above condition can be found. It is important to note that these column additions use $\mathbb{Z}_2$ (i.e. mod 2) arithmetic so that $1 + 1 = 0$.

As an example, the $1-$dimensional boundary matrix ∂_1 computed from the simplicial complex defined in top left of Fig. 2 is reduced with first few

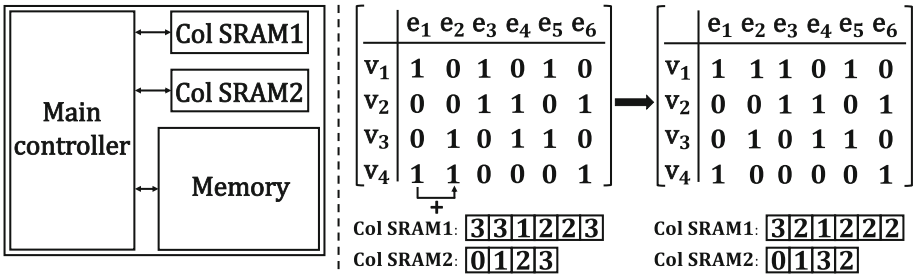

Fig. 4. Left: architecture of the proposed hardware implementation of boundary matrix reduction. Right: example Col SRAM updates of the first reduction step in Fig. 3.

steps of reduction process shown in Fig. 3. The reduced result is R_1 where $low_{R_1}(i) \neq low_{R_1}(j), i \neq j$ where column i and j specify two nonzero columns (see lower right of Fig. 3). Pseucodes for boundary matrix reduction is provided in Algorithm 1 where the boundary matrix is first scanned to initialize $low_R()$ with correct indices (the first for-loop) after which the algorithm follows what we have described previously.

4 Hardware Implementation

As described in Sect. 3, boundary matrix reduction entails a lot of column operations, which make it time-consuming to compute persistence diagram. Provided a large boundary matrix, the excessive number of cache misses caused by aforementioned column operations involved in reduction process inevitably become a major challenge in memory operations. Moreover, finding two columns i and j satisfying $low_R(i) = low_R(j)$ proves to be difficult due to the time and power consumption incurred from the perspective of both software and hardware. A novel hardware accelerator for boundary matrix reduction is proposed in this section, which can potentially achieve up to 20k–30k times speedups on MNIST and MIAS dataset.

A full description of the functionality for each hardware module illustrated in Fig. 4 is as follows:

1. Memory: the memory module stores boundary matrix. Currently, only on-chip SRAM is considered. The architecture can be easily extended to DRAM when applied to a larger dataset.
2. Col SRAM1: Col SRAM1 stores the index of the lowest 1 in each column (i.e. $low_R()$).
3. Col SRAM2: Col SRAM2 stores the number columns which share the same $low_R()$.
4. Main Controller: the main controller module is responsible for the control of the entire hardware including reading and writing of the SRAMs.

An example hardware flow of the first boundary matrix reduction step in Fig. 3 is shown in Fig. 4. Col SRAM1 indicates the index of the lowest 1 in column i (i.e. $low_R(i)$) while Col SRAM2 records the number of columns with the same $low_R()$. As the process of boundary matrix reduction progresses, the SRAMs are updated concurrently. With the information readily stored in SRAMs, the time to search for a new pair of qualifying columns can be greatly reduced.

Specifically, the Memory module is configured to have 24 SRAMs, each with depth of 1536 and width of 32−bit in our implementation. The circuit is synthesized with 28 nm CMOS technology. The circuit is designed to have an area of 0.5 mm^2 and to consume 20 mW power at 1 Ghz clock frequency.

10 samples were randomly drawn from both MNIST and MIAS dataset, and we measured their software and hardware reduction time on 1− and 2− dimensional boundary matrices separately for clarity purpose. Our software implementation of boundary matrix reduction (abbreviated as SW in Table 1 for clarity) was coded in C++ and compiled on a 64−bit Windows with Visual Studio 2015 as baseline approach. It took in filtration matrices as inputs and produced reduced boundary matrices as outputs. Additionally, software metrics reported in Table 1 were produced from a machine with an Intel Core i7-9700K 3.6 GHz CPU, and 8GB DDR4 memory. Table 1 gives averaged running time over 10 samples for both dataset, and we can observe considerable speedups from our proposed hardware accelerator especially for 1-dimensional boundary matrices.

Table 1. Comparisons of processing time between software and hardware implementations.

Dataset	Dimensions	SW runtime	HW runtime	HW/SW speedups
MNIST	2-dim	2224 ms	1.10 ms	2022x
	1-dim	2639 ms	0.13 ms	20300x
MIAS	2-dim	4087 ms	1.51 ms	2706x
	1-dim	4816 ms	0.22 ms	21891x

Acknowledgement. This work is partially supported by National Science Foundation Awards CCF-1854742, CCF-1815699, IIS-1855759 and CCF-1855760.

References

1. Chung, M.K., Bubenik, P., Kim, P.T.: Persistence diagrams of cortical surface data. In: Prince, J.L., Pham, D.L., Myers, K.J. (eds.) IPMI 2009. LNCS, vol. 5636, pp. 386–397. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02498-6_32
2. Chung, M., Hanson, J., Ye, J., Davidson, R., Pollak, S.: Persistent homology in sparse regression and its application to brain morphometry. IEEE Trans. Med. Imaging **34**(9), 1928–1939 (2015). https://doi.org/10.1109/TMI.2015.2416271
3. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. Discret. Comput. Geom. **37**(1), 103–120 (2007)

4. Cohen-Steiner, D., Edelsbrunner, H., Morozov, D.: Vines and vineyards by updating persistence in linear time. In: Proceedings of the Twenty-second Annual Symposium on Computational Geometry, SCG 2006, pp. 119–126. ACM, New York (2006). https://doi.org/10.1145/1137856.1137877
5. Deng, C., Liao, S., Xie, Y., Parhi, K.K., Qian, X., Yuan, B.: PermDNN: efficient compressed DNN architecture with permuted diagonal matrices. In: 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 189–202, October 2018. https://doi.org/10.1109/MICRO.2018.00024
6. Deng, C., Sun, F., Qian, X., Lin, J., Wang, Z., Yuan, B.: TIE: energy-efficient tensor train-based inference engine for deep neural network. In: Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, pp. 264–278. ACM, New York (2019). https://doi.org/10.1145/3307650.3322258
7. Dey, T.K., Li, K., Sun, J., Cohen-Steiner, D.: Computing geometry-aware handle and tunnel loops in 3D models. In: ACM SIGGRAPH 2008 Papers, pp. 45:1–45:9. ACM, New York (2008). https://doi.org/10.1145/1399504.1360644
8. Edelsbrunner, H., Letscher, D., Zomorodian, A.: Topological persistence and simplification. Discrete Comput. Geom. **28**(4), 511–533 (2002). https://doi.org/10.1007/s00454-002-2885-2
9. Edelsbrunner, H.: Surface tiling with differential topology. In: Desbrun, M., Pottmann, H. (eds.) Eurographics Symposium on Geometry Processing 2005. The Eurographics Association (2005). https://doi.org/10.2312/SGP/SGP05/009-011
10. Edelsbrunner, H., Harer, J.: Computational Topology: An Introduction. American Mathematical Society, Providence (2010)
11. Ghrist, R., Muhammad, A.: Coverage and hole-detection in sensor networks via homology. In: IPSN 2005: Fourth International Symposium on Information Processing in Sensor Networks, pp. 254–260, April 2005. https://doi.org/10.1109/IPSN.2005.1440933
12. Goodman, J.E., O'Rourke, J. (eds.): Handbook of Discrete and Computational Geometry. CRC Press, Inc., Boca Raton (1997)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
14. Lee, H., Kang, H., Chung, M.K., Lee, D.S.: Persistent brain network homology from the perspective of dendrogram. IEEE Trans. Med. Imaging **31**, 2267–2277 (2012)
15. Li, C., Ovsjanikov, M., Chazal, F.: Persistence-based structural recognition. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 2003–2010, June 2014. https://doi.org/10.1109/CVPR.2014.257
16. Munkres, J.R.: Elements of Algebraic Topology. Addison Wesley Publishing Company (1984). http://www.worldcat.org/isbn/0201045869
17. Oudot, S.: Persistence theory - from quiver representations to data analysis. In: Mathematical Surveys and Monographs (2015)
18. Perea, J.A., Harer, J.: Sliding windows and persistence: an application of topological methods to signal analysis. Found. Comput. Math. **15**, 799–838 (2015)
19. Silva, V.D., Ghrist, R.: Blind swarms for coverage in 2-D. In: Proceedings of Robotics: Science and Systems, p. 01 (2005)
20. Suckling, J.: The mammographic image analysis society digital mammogram database. Exerpta Medica. International Congress Series 1069, January 1994