Ripser++: GPU-Accelerated Computation of Vietoris-Rips Persistence Barcodes

Simon Zhang, Mengbai Xiao and Hao Wang

The Ohio State University, USA

Student Presentation

What is a Vietoris-Rips Filtration?

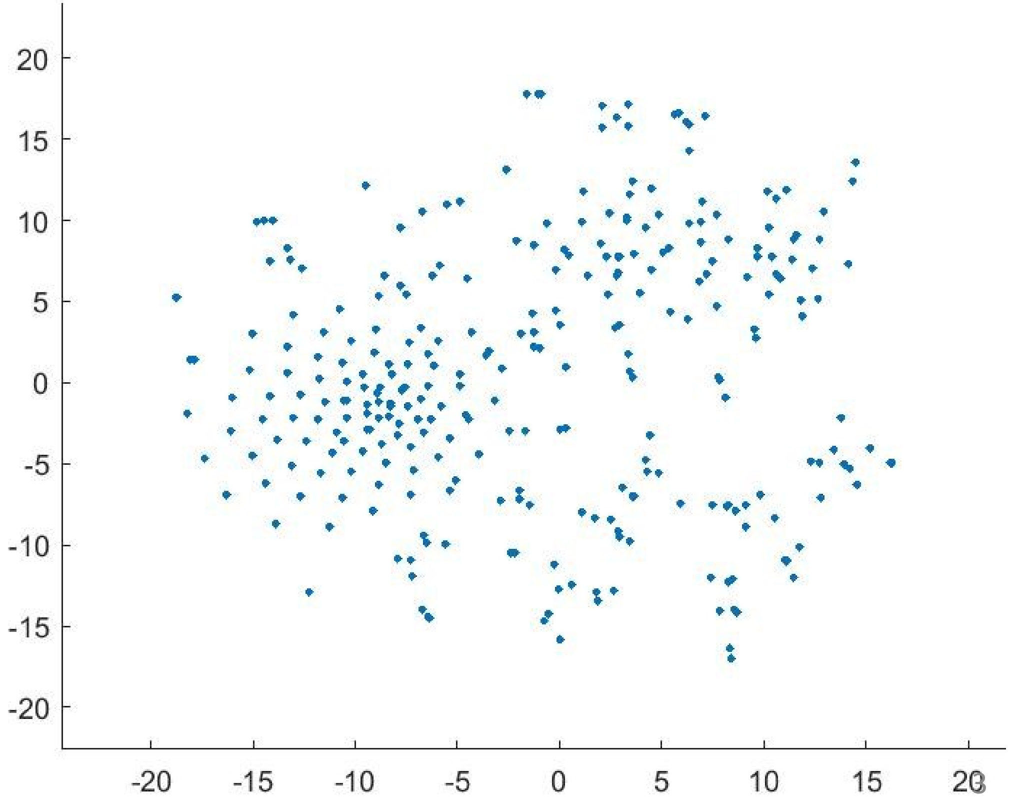
- Let X be a set of points with an underlying metric
- For every t (real), define a Vietoris-Rips complex by:

$$Rips_t(X) = \{\emptyset \neq s \subset X \mid diam(s) \leq t\}$$

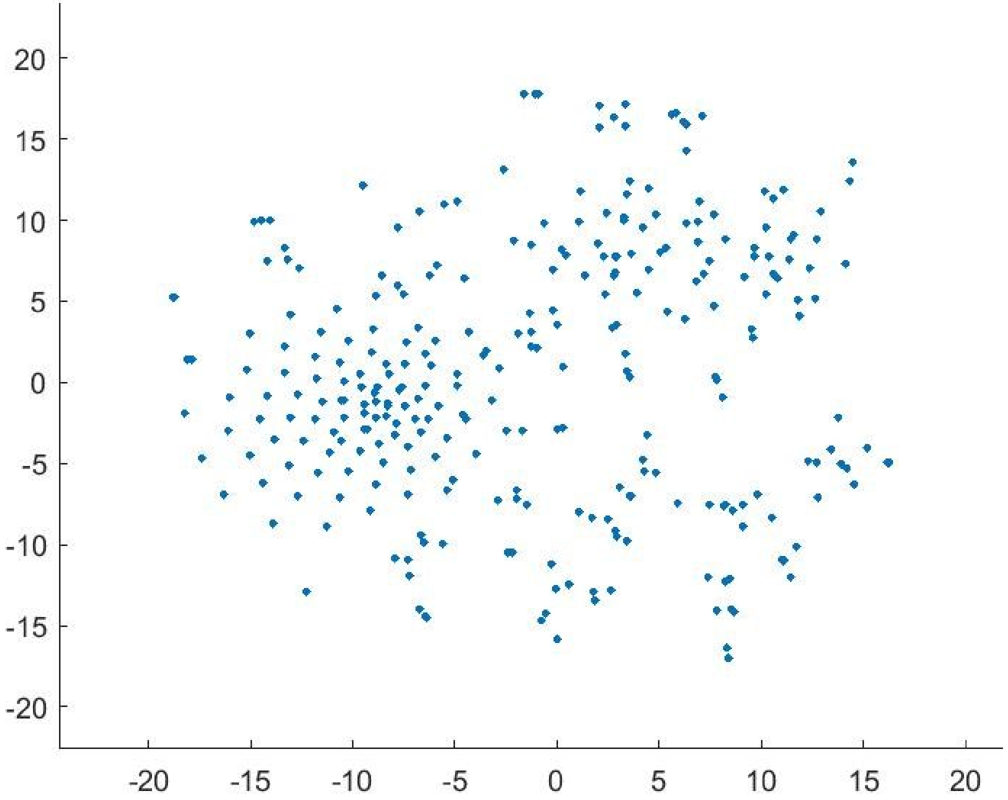
- Where the s are also known as (abstract) simplices on X
- The increasing sequence of such Vietoris-Rips complexes indexed by t and ordered by inclusions form a Vietoris-Rips filtration

An Illustration of a Vietoris-Rips Filtration

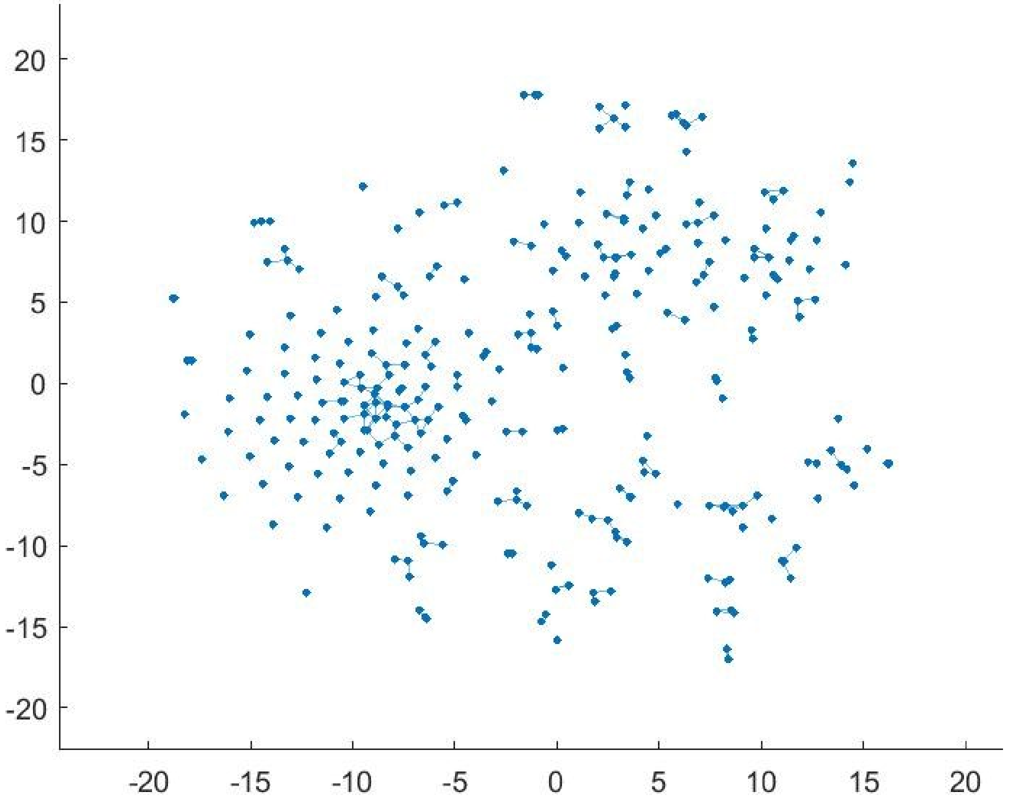
- Real-World Data: the C. elegans neuronal network X
  - Each node is a neuron and edges are synapses or gap junctions between neurons
  - one of the simplest connectomes in living organisms
- With dimensionality reduction from 202 dimensions down to the Euclidean plane by the t-SNE algorithm



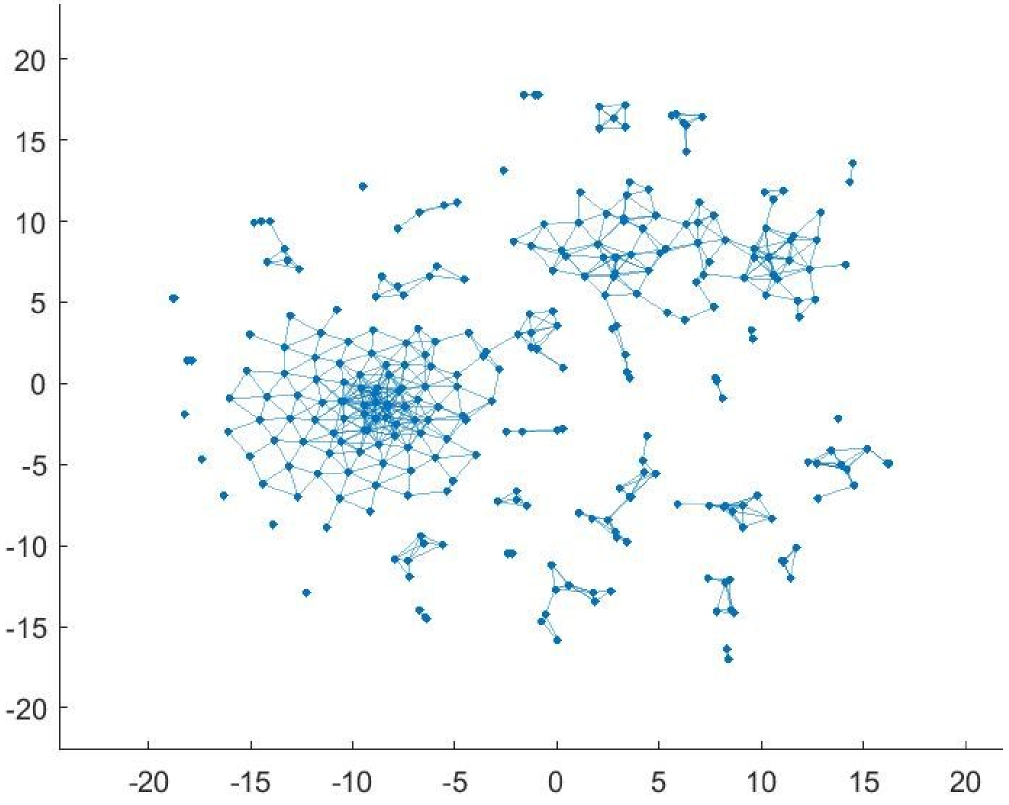
A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 0.0 (the original point cloud)



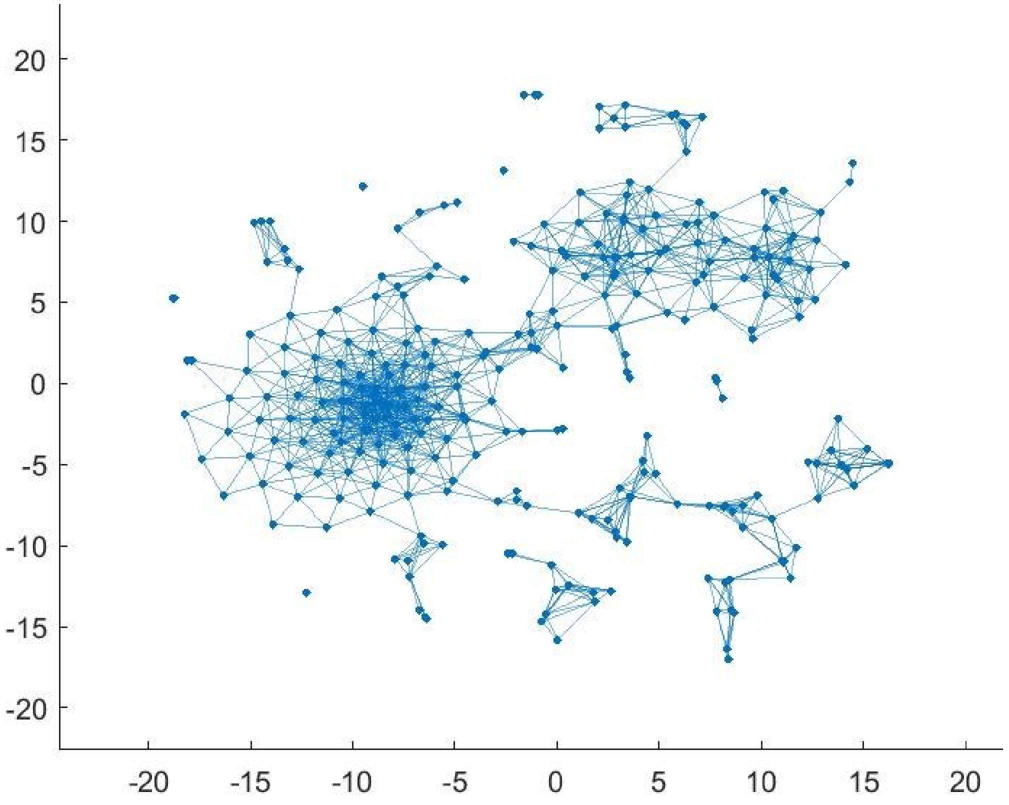
A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 1.0



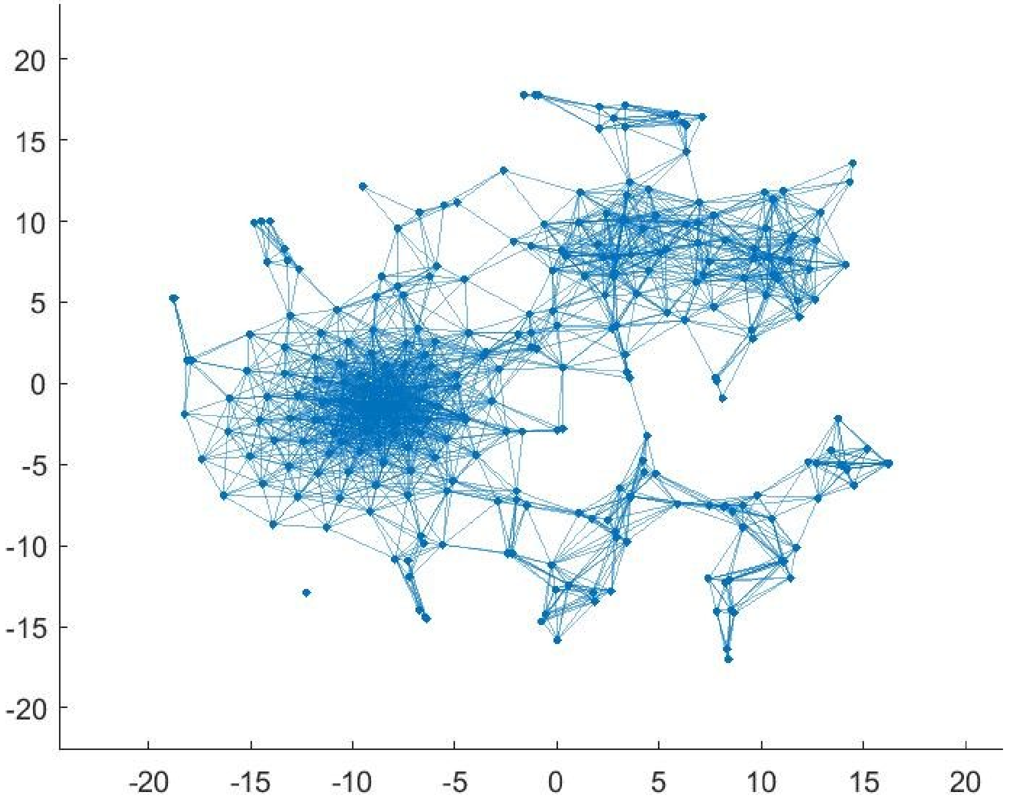
A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 2.0



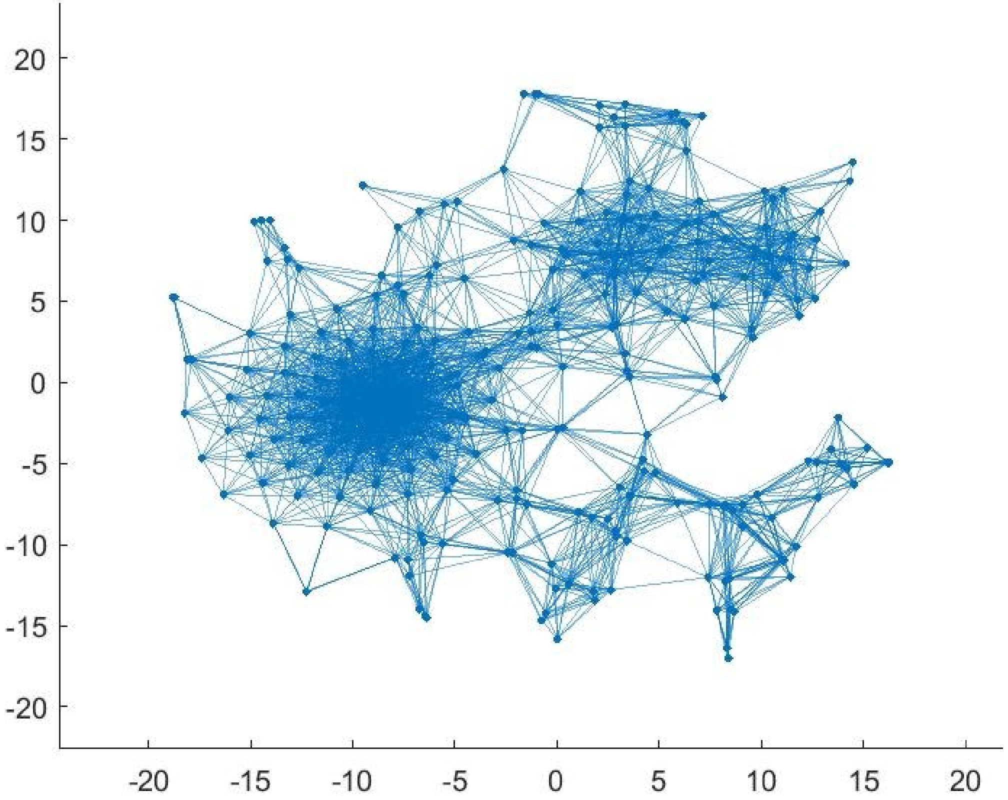
A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 3.0



A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 4.0

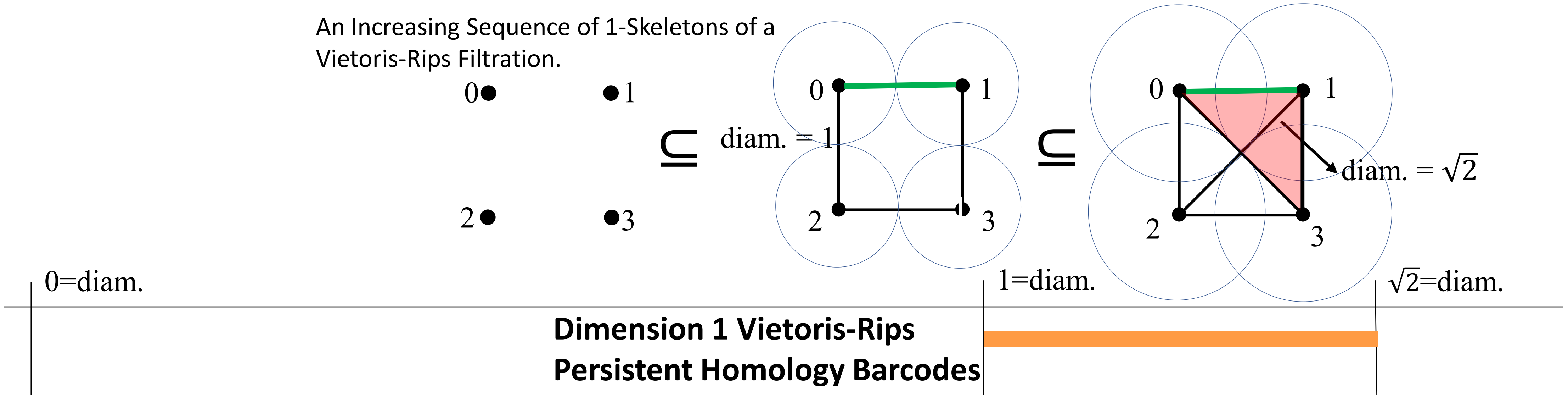


A illustration of the 1-skeleton of the Vietoris-Rips Complex up to diameter= 5.0



Persistent Homology: Persistence Barcodes

- Persistence Barcodes:
  - Consider a multiset of pairs (b,d) of simplex diameters where a “birth” and “death”, respectively of homological features occur in the Vietoris-Rips filtration.
    - e.g.$(1, \sqrt{(2)})$ is a birth-death pair
  - The multiset of half open intervals {[b,d)} represent the persistence barcodes

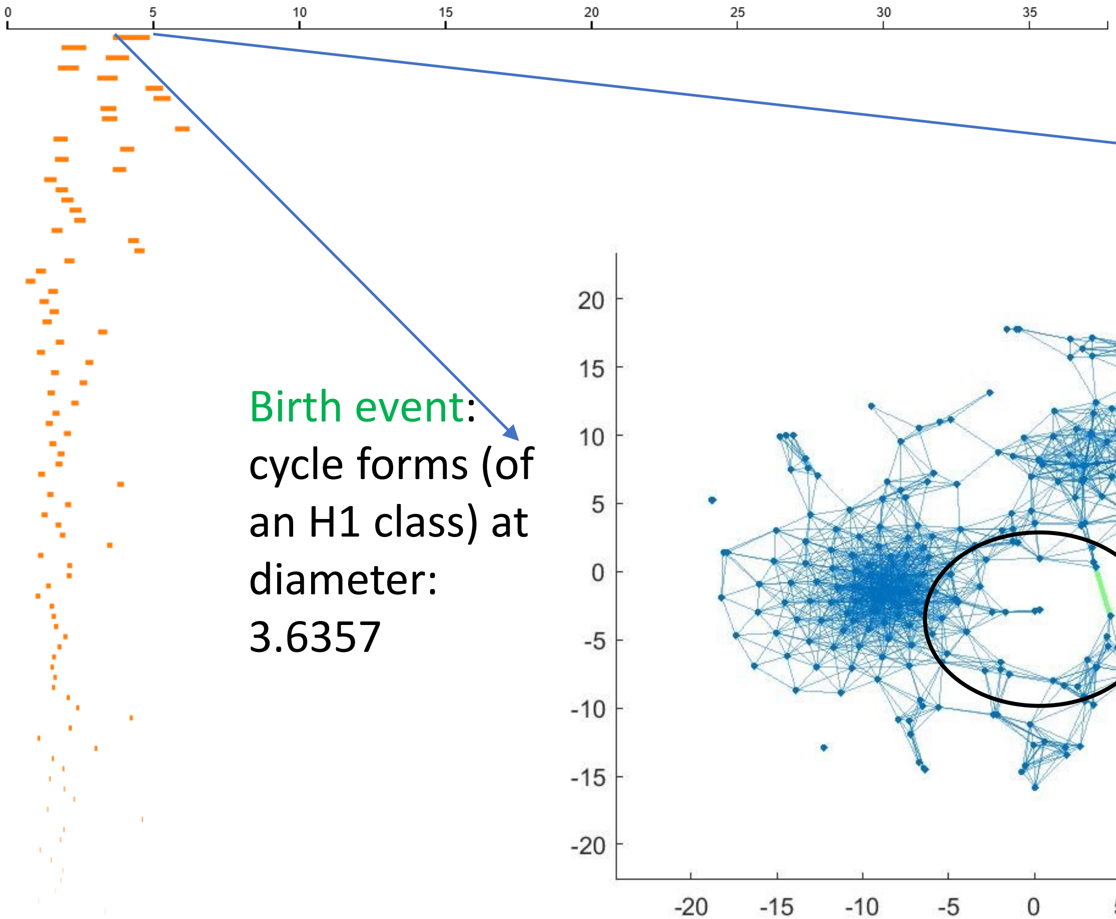


An Increasing Sequence of 1-Skeletons of a Vietoris-Rips Filtration.

Dimension 1 Vietoris-Rips Persistent Homology Barcodes

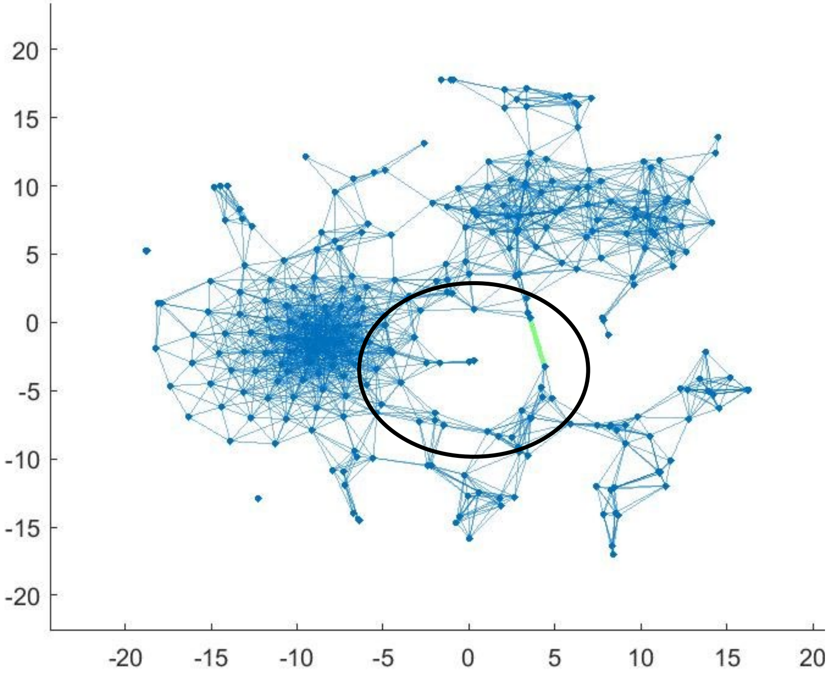
Persistent Homology: Birth and Death for H1 of the C. elegans Dataset

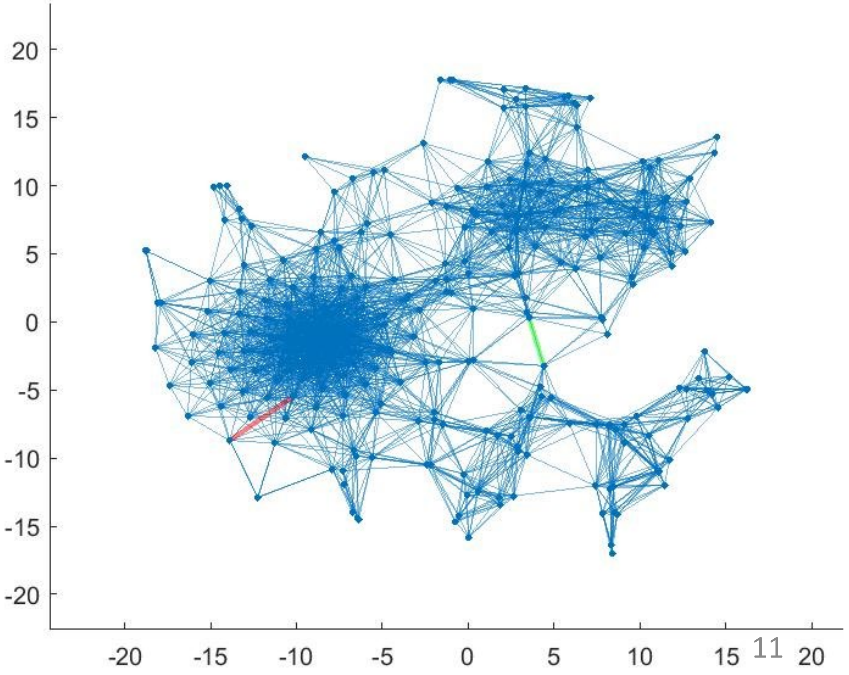
Persistence Barcodes:



Death event: (merge or zeroing of H1 class due to triangles (only the longest edge of the triangle is shown) added into the flag complex) at diameter: 4.8984

Birth event: cycle forms (of an H1 class) at diameter: 3.6357

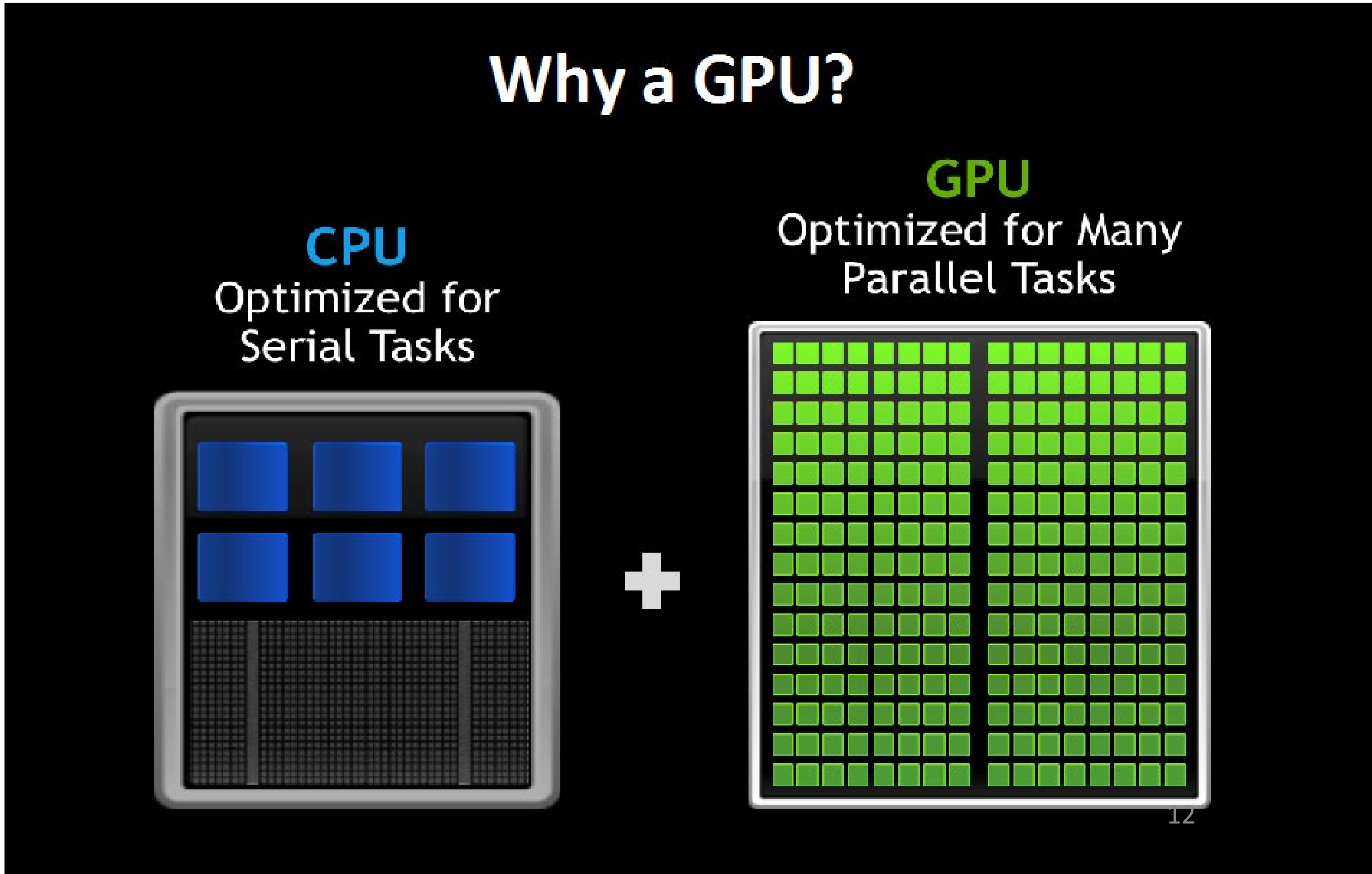




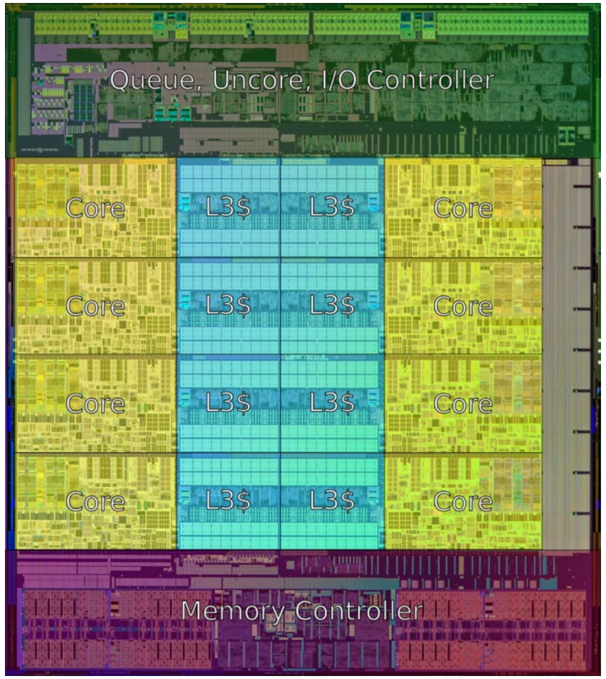
How does GPU offer Massive Parallelism?

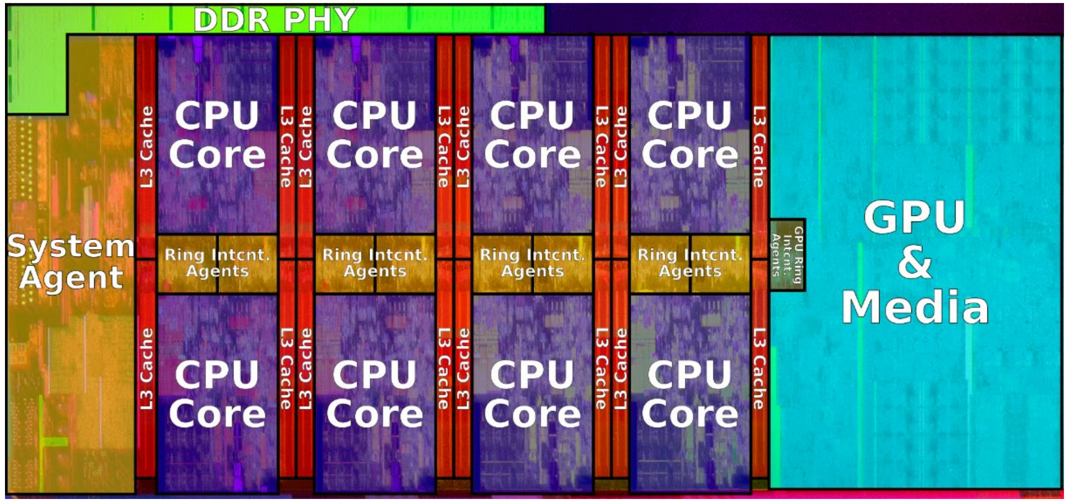
- A GPU (or graphical processing unit) is a processor designed for massively parallel algorithms executing in SIMT (single instruction multiple thread) mode

- If massive parallelism can be utilized then there can be tremendous speedup



GPU Acceleration is a Part of General Computing





2014 Q3 launched Intel Core i7-5960X (Haswell-E)

Large shared L3 cache, no GPU.

Eight 3.0 GHz cores (16 ops per cycles).

2018 Q4 launched Intel Core i7-9700K (Coffee Lake)

The die area is also used for GPU.

Eight 3.6 GHz cores (16 ops per cycles).

- **2014 Intel i7 CPU performance = 3.0 * 16 * 8 = 384 Gflops**
- **2018 Intel i7 CPU performance = 3.6 * 16 * 8 = 460.8 Gflops**
- As the area of **CPU cores** is shrinking, CPU performance doesn’t significantly improve in the past five years. Overall performance must be accelerated by **GPU.**

Performance of Ripser++ at a Glance

- Example dataset:
  - 192 points on $\mathbb{S}^2$ (embedded in $\mathbb{R}^3$)
  - Persistent homology barcodes up to dimension 3
  - Over 2.1 billion simplices in the 4-skeleton flag complex

Performance of Ripser++ at a Glance

- Example dataset:
  - 192 points on $\mathbb{S}^2$ (embedded in $\mathbb{R}^3$)
  - Persistent homology barcodes up to dimension 3
  - Over 2.1 billion simplices in the 4-skeleton flag complex
- Comparison with existing software:

  Super computer node: 28 x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.4GHz, 100 GB DRAM

  - Eirene: 769.50 seconds, 168.00 GB for CPU (no generators recorded)
  - Ripser: 36.96 seconds, 4.32 GB for CPU
  - **Ripser++:** 2.43 seconds **(15x+)**, 2.92 GB for GPU and 2.03 GB for CPU
    - Super computing GPU: NVIDIA Tesla V100, 32 GB Device Memory

  On my $900 laptop: 6 x Intel(R) Core(TM) i7-9750H CPU @ 2.6 GHz, 16 GB DRAM

  - **Ripser++:** 5.0 seconds **(7x+)**, 2.92 GB for GPU and 2.03 GB for CPU
    - Laptop GPU : NVIDIA GTX 1660 Ti, 6 GB Device Memory
- **Ripser++ is fastest in Vietoris-Rips persistence barcode computation**

Computation of Vietoris-Rips Persistence Barcodes

for standard matrix reduction algorithm, *see [Edelsbrunner, Letscher, Zomordian 2002]*

Let $\boldsymbol{K}$ be the largest complex of $Rips_\bullet(X)$

Let $\boldsymbol{F}: \mathbb{R} \rightarrow \boldsymbol{K}, \boldsymbol{S}: \mathbb{N} \rightarrow \boldsymbol{K}$ and $r: \mathbb{R} \rightarrow \mathbb{N}$

Algorithm 1 : Standard Vietoris-Rips Persistent Homology Computation

```
Require: data $X$ such as a point cloud, threshold $t$ ,and computation dim. $d$
Ensure: $\boldsymbol{P}$ persistence barcodes
1: $\boldsymbol{F} \leftarrow Rips_\bullet(X)$   ▷ Let $\boldsymbol{F}$ be the Rips filtration of $X$ for a given threshold $t$ and dim. of computation $d$
2: $\boldsymbol{S} \leftarrow$simplex-wise-refinement($\boldsymbol{F}$)   ▷ $\boldsymbol{F} = \boldsymbol{S} \circ r$ where $r$ is injective
3: $R \leftarrow \partial(\boldsymbol{S})$
4: for every column j in $R$ do   ▷ the standard matrix reduction algorithm
5:     while $\exists\ k < j$ s.t. $low_R(j)$=$low_R(k)$ do
6:         column $j \leftarrow$ column $k$ + column $j$
7:     if $low(j) \neq -1$ then
8:         $\boldsymbol{P} \leftarrow \boldsymbol{P} \cup r^{-1}([low(j), j))$ ▷ we call the pair $(low(j), j)$ a pivot in the matrix $R$.
```

What are the Challenges for Parallelization?

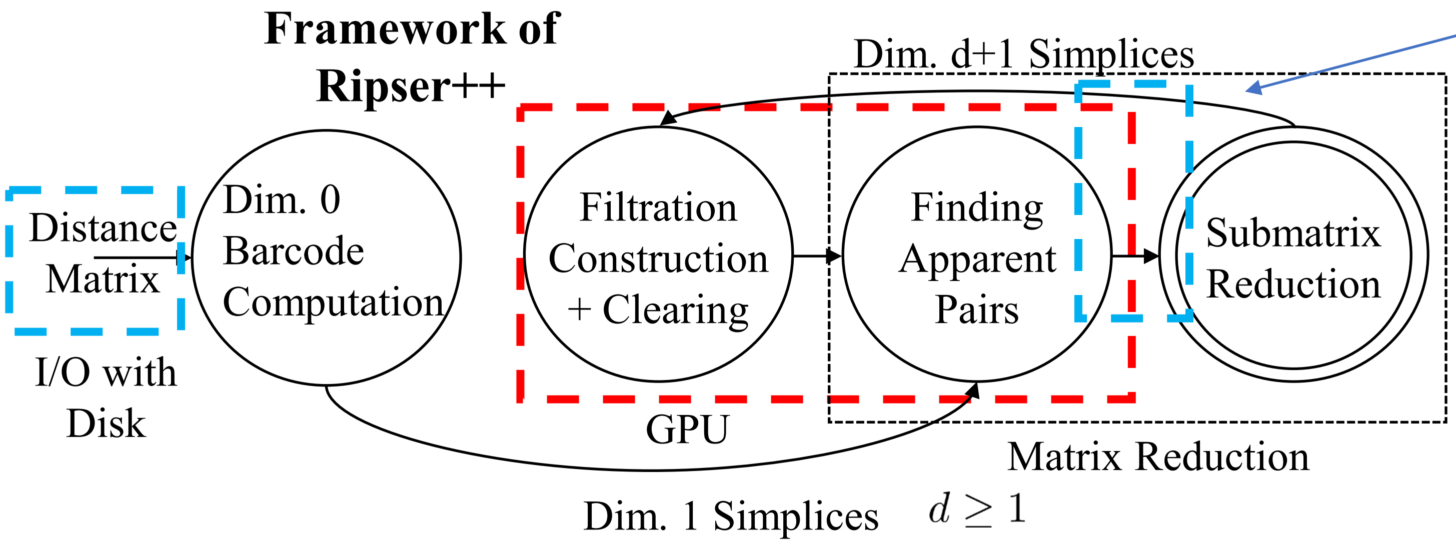
- Exponentially growing filtration size in dim. d of computation (lines 1 and 2)
- Sequential memory accesses (lines 1 and 2)
- Indefinite O(filt. size) col. additions (line 5)
- Heavy data movement during col. addition (lines 6)
- Extremely sparse computation!

- Identifying hidden parallelism

- Our goal is to develop GPU-accelerated parallel computation of this algorithm

Design Goals for High Performance

- Build upon the computational foundations of Ripser
- Parallelization of persistent homology barcode computation
- Eliminate as much I/O as possible
- Potential for memory performance through implementation



Efficient data structures to store persistence pairs and coboundary matrix columns

The Four Components of Ripser++ for Accelerated Performance

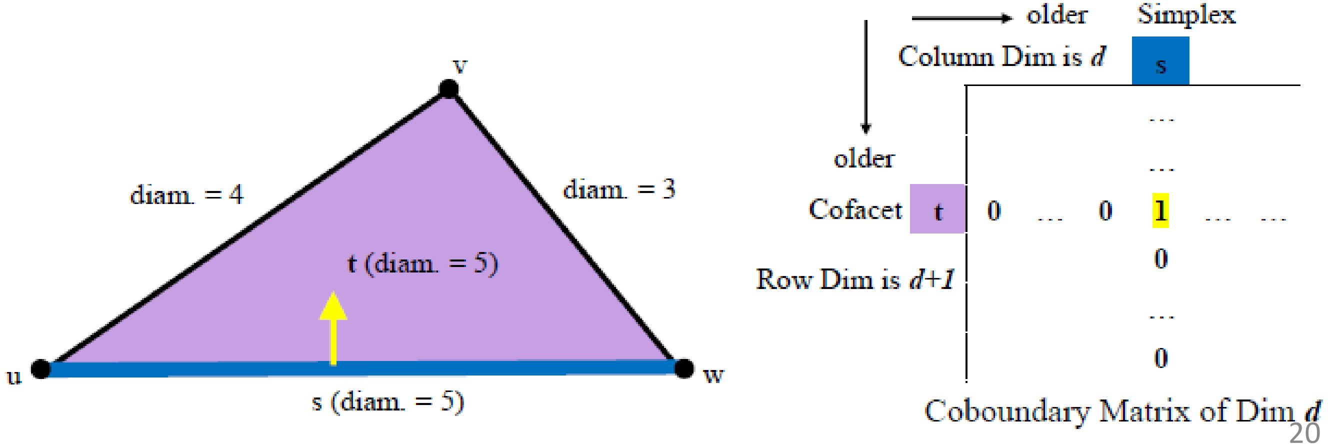
- Finding and Using Apparent Pairs
- A CPU-GPU Hybrid
- Efficient Filtration Construction with Clearing
- Efficient Hashmap

What is an Apparent Pair? (preliminaries)

- Given data (e.g. a point cloud X), form the Rips filtration $Rips_t(X)$ indexed by diameter thresholds t (up to some max threshold and dimension of computation)
- Define a simplex-wise filtration refinement on $Rips_t(X)$ via the ordering on simplices:
  - Increasing simplex diameters, followed by
  - Increasing simplex dimension, followed by
  - Decreasing simplex combinatorial indices
- Where the diameter of a simplex is the maximum length edge in the clique associated with a simplex
- Where the combinatorial index is a bijective encoding of simplices to the natural numbers [Knuth 1997] (most originally known to Pascal in 1887)
- If s<s' in the ordering, then s is **older** than s' and s' is **younger** than s

What is an Apparent Pair?

- A facet s of a simplex t is defined as the codimension 1 simplex in the boundary of t.
  - e.g. simplex (210) (having vertices 0, 1, and 2) has facets (10), (21), and (20)
- A cofacet t of simplex s is defined as a simplex containing s as a facet
  - E.g. simplex (10) could have cofacets (210) and (310)
- A pair of simplices (s,t) is an apparent pair [Bauer 2019] iff
  - s is the **youngest** facet of t
  - t is the **oldest** cofacet of s



(a) (b)

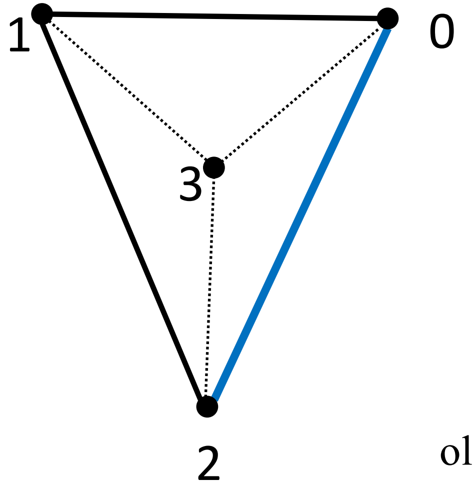
Finding Apparent Pairs

- The **Apparent Pairs Lemma** from this paper:
- Given a simplex s and its cofacet t
  1. t is the lexicographically greatest cofacet of s with diam(s)=diam(t) and
  2. no facet s’ of t is strictly lexicographically smaller than s with diam(s’)=diam(s)

  iff (s,t) is an apparent pair
- **Corollary**: apparent pairs can be found massively in parallel
- Checking this lemma for a given simplex is memory efficient
- Facets and cofacets can be efficiently enumerated by computation of combinatorial indices

Finding Apparent Pairs Algorithm, a Simple Case for a Single Column

- Consider edge (20) (assign a thread to this column)



Dim 1 Coboundary Matrix

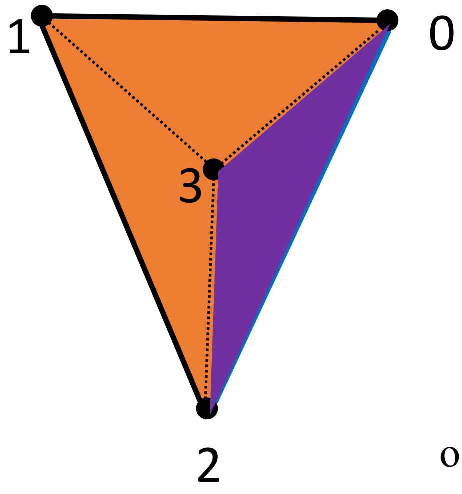
→ older

↓ older

(diam., simplex)	(6, (10))	(5, (20))	(4, (21))	(3, (30))	(2, (31))	(1, (32))
(6, (210))	**1**	**1**	**1**			
(6, (310))	**1**			**1**	**1**	
(5, (320))		**1**		**1**		**1**
(4, (321))			**1**		**1**	**1**

Finding Apparent Pairs Algorithm, a Simple Case for a Single Column

- Consider edge (20) (assign a thread to this column)
  - Check condition 1 of lemma: search in decreasing lexicographic order the cofacets of (20) for a triangle of diam((20))=5. Find (320)



Dim 1 Coboundary Matrix

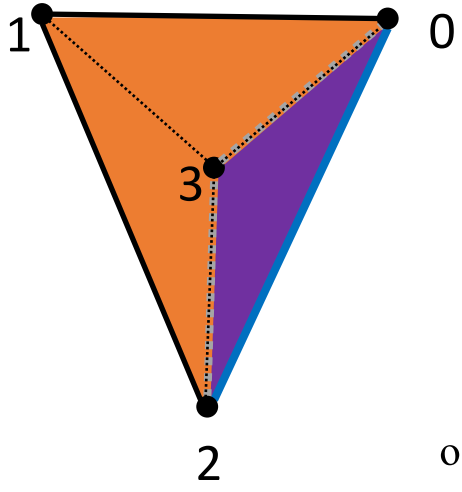
→ older

↓ older

(diam., simplex)	(6, (10))	(5, (20))	(4, (21))	(3, (30))	(2, (31))	(1, (32))
(6, (210))	1	1	1			
(6, (310))	1			1	1	
(5, (320))		1		1		1
(4, (321))			1		1	1

Finding Apparent Pairs Algorithm, a Simple Case for a Single Column

- Consider edge (20) (assign a thread to this column)
  - Check condition 1 of lemma: search in decreasing lexicographic order the cofacets of (20) for a triangle of diam((20))=5. Find (320)
  - Check condition 2 of lemma: search in increasing lexicographic order the facets of (320) for a facet s′ with diam(s′)=5 and cidx(s′)<cidx((20))



Dim 1 Coboundary Matrix

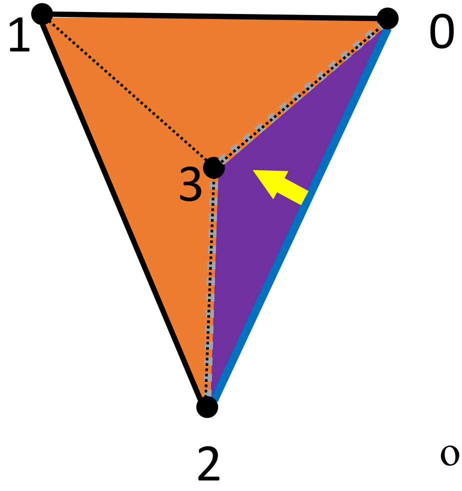
→ older

(diam., simplex)	(6, (10))	(5, (20))	(4, (21))	(3, (30))	(2, (31))	(1, (32))
(6, (210))	1	1	1			
(6, (310))	1			1	1	
(5, (320))		1		1		1
(4, (321))			1		1	1

↓ older

Finding Apparent Pairs Algorithm, a Simple Case for a Single Column

- Consider edge (20) (assign a thread to this column)
  - Check condition 1 of lemma: search in decreasing lexicographic order the cofacets of (20) for a triangle of diam((20))=5. Find (320)
  - Check condition 2 of lemma: search in increasing lexicographic order the facets of (320) for a facet s' with diam(s')=5 and cidx(s')<cidx((20))



Dim 1 Coboundary Matrix

→ older

older ↓

(diam., simplex)	(6, (10))	(5, (20))	(4, (21))	(3, (30))	(2, (31))	(1, (32))
(6, (210))	1	1	1			
(6, (310))	1			1	1	
(5, (320))		1		1		1
(4, (321))			1		1	1

Apparent Pairs Dominate Vietoris-Rips Persistence Pairs

- Empirically on real world and synthetic datasets, up to 99.9% of persistence pairs are apparent

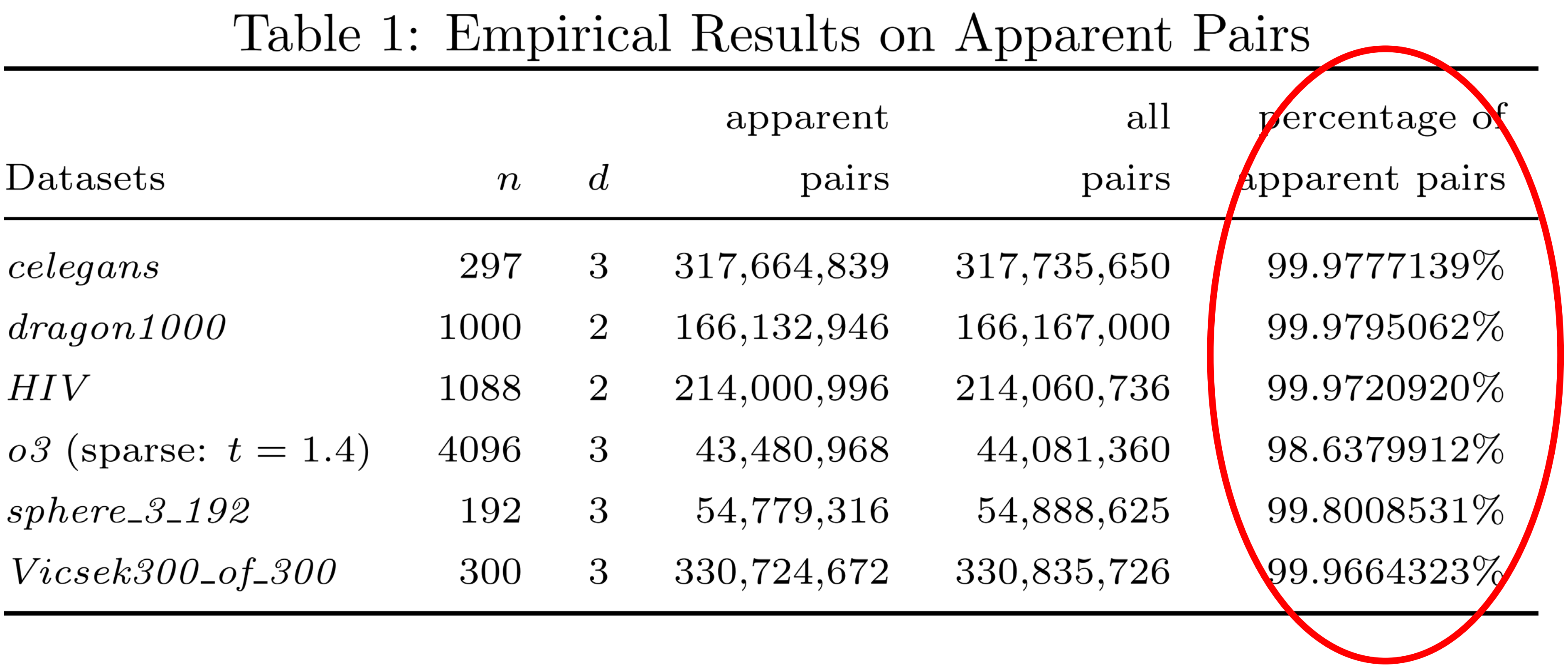
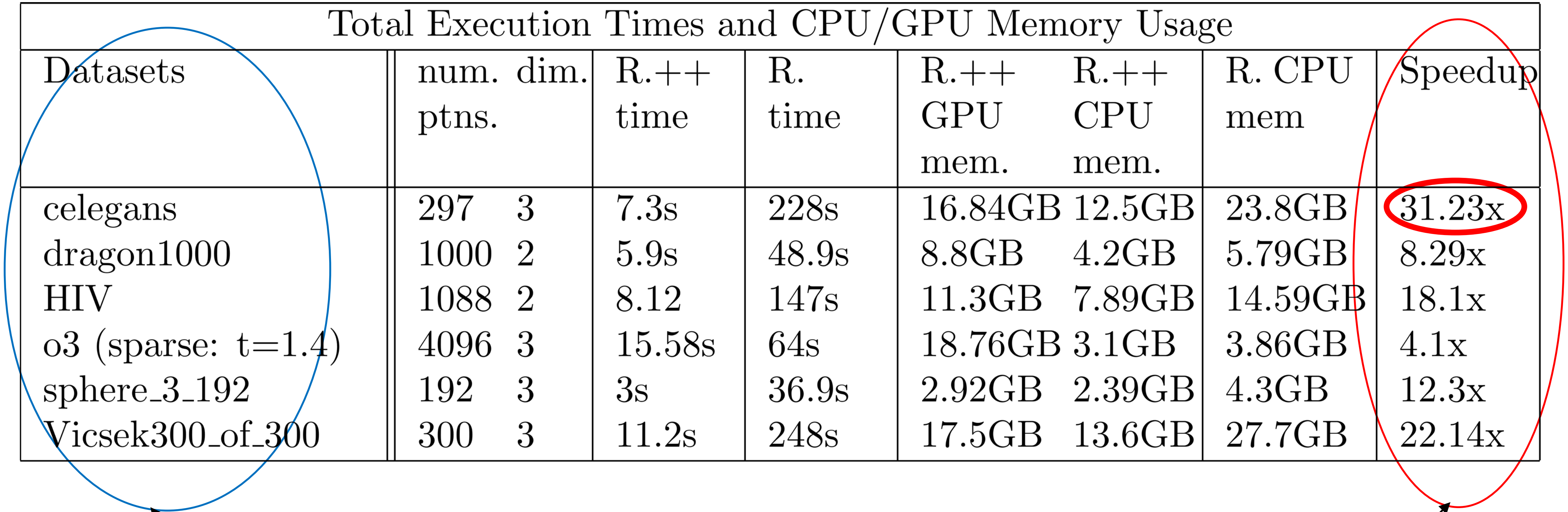


Table 1: Empirical Results on Apparent Pairs

Datasets	n	d	apparent pairs	all pairs	percentage of apparent pairs
celegans	297	3	317,664,839	317,735,650	99.9777139%
dragon1000	1000	2	166,132,946	166,167,000	99.9795062%
HIV	1088	2	214,000,996	214,060,736	99.9720920%
o3 (sparse: $t = 1.4$)	4096	3	43,480,968	44,081,360	98.6379912%
sphere_3_192	192	3	54,779,316	54,888,625	99.8008531%
Vicsek300_of_300	300	3	330,724,672	330,835,726	99.9664323%

Time and Memory Performance of Ripser++



Total Execution Times and CPU/GPU Memory Usage

Datasets	num. ptns.	dim.	R.++ time	R. time	R.++ GPU mem.	R.++ CPU mem.	R. CPU mem	Speedup
celegans	297	3	7.3s	228s	16.84GB	12.5GB	23.8GB	31.23x
dragon1000	1000	2	5.9s	48.9s	8.8GB	4.2GB	5.79GB	8.29x
HIV	1088	2	8.12	147s	11.3GB	7.89GB	14.59GB	18.1x
o3 (sparse: t=1.4)	4096	3	15.58s	64s	18.76GB	3.1GB	3.86GB	4.1x
sphere_3_192	192	3	3s	36.9s	2.92GB	2.39GB	4.3GB	12.3x
Vicsek300_of_300	300	3	11.2s	248s	17.5GB	13.6GB	27.7GB	22.14x

A diverse set of real-world and synthetic data sets

Speedup on these datasets

Summary

- Ripser++ is software with GPU-acceleration for computation of Vietoris-Rips persistent barcodes with up to 30x speedup over Ripser
- Apparent pairs are explored and studied
  - Utilized in a massively parallel way
  - Foundations for their dominant appearance in Vietoris-Rips filtrations
- Future work based on Ripser++
  - Accelerating persistent homology computation with lower-star filtrations or other filtrations types in a similar manner
  - Applications requiring high speed computations of persistent homology
  - Ripser++ on a cluster of GPUs (for even larger datasets)

Use Ripser++!

- Code is available at
  - https://github.com/simonzhang00/ripser-plusplus
- Read the full version paper at:
  - https://arxiv.org/abs/2003.07989
  - More theoretical results and details on implementation/optimizations

Thank You!