Scalable Computation of Topological Abstractions for Scalar Data

Michael Will[1], Jonas Lukasczyk[1], Julien Tierny[2], Attila Gyulassy[3], Gunther Weber[4], Hamish Carr[5], and Christoph Garth[1]

[1] RPTU University Kaiserslautern-Landau
[2] CNRS and Sorbonne Université
[3] University of Utah School of Computing and the Scientific Computing and Imaging Institute
[4] Lawrence Berkeley National Laboratory
[5] University of Leeds

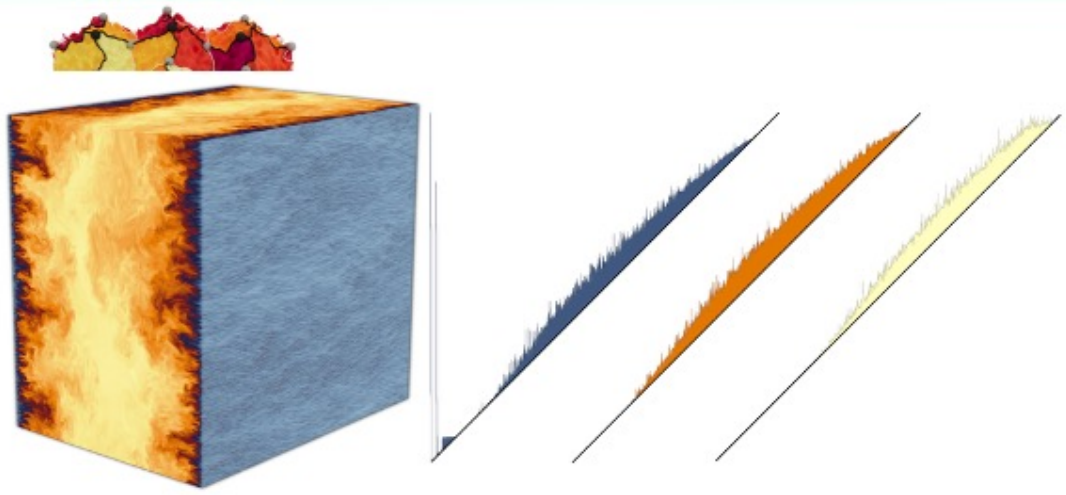
EuroVis 2026
The Eurographics Conference on Visualization

Motivation

- Topological Data Analysis uses concepts from Computational Topology and has become a popular tool for capturing the inherent structure and features of interest of scalar field data
- there are several topological abstractions of interest widely used for many applications
- growing dataset sizes makes computing them on single machines infeasible
- MS complex is a fundamental topological abstraction which segments the domain into areas of similar gradient flow

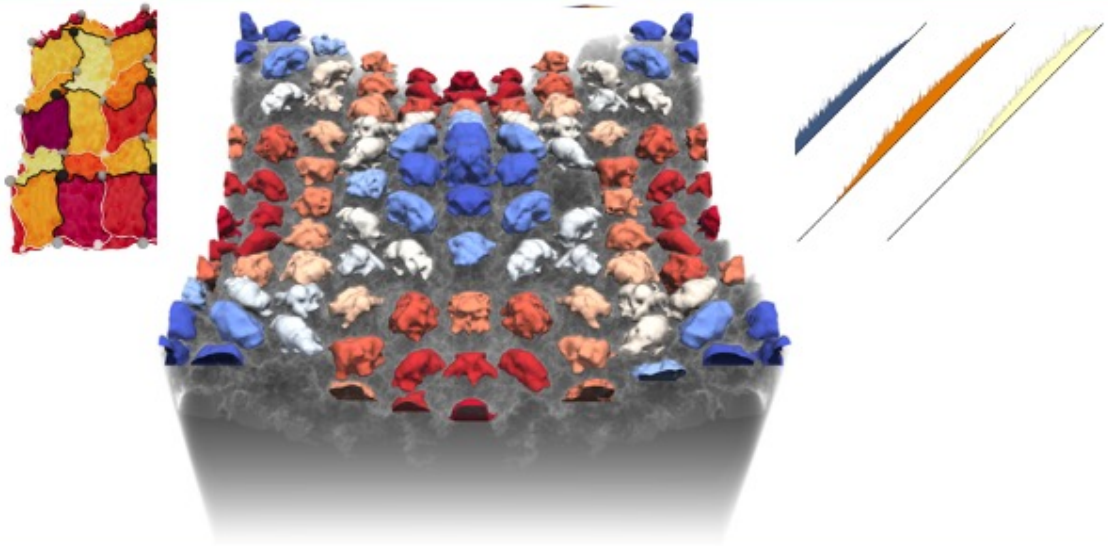
Motivation

LE GUILLOU E., FORTIN P., TIERNY J.:
Distributed Discrete Morse Sandwich: Efficient Computation of Persistence Diagrams for Massive Scalar Data, IEEE Transactions on Parallel & Distributed Systems



- Persistence diagram on a subset of Turbulent Channel Flow with 6 billion vertices, computed with over 4TB of RAM

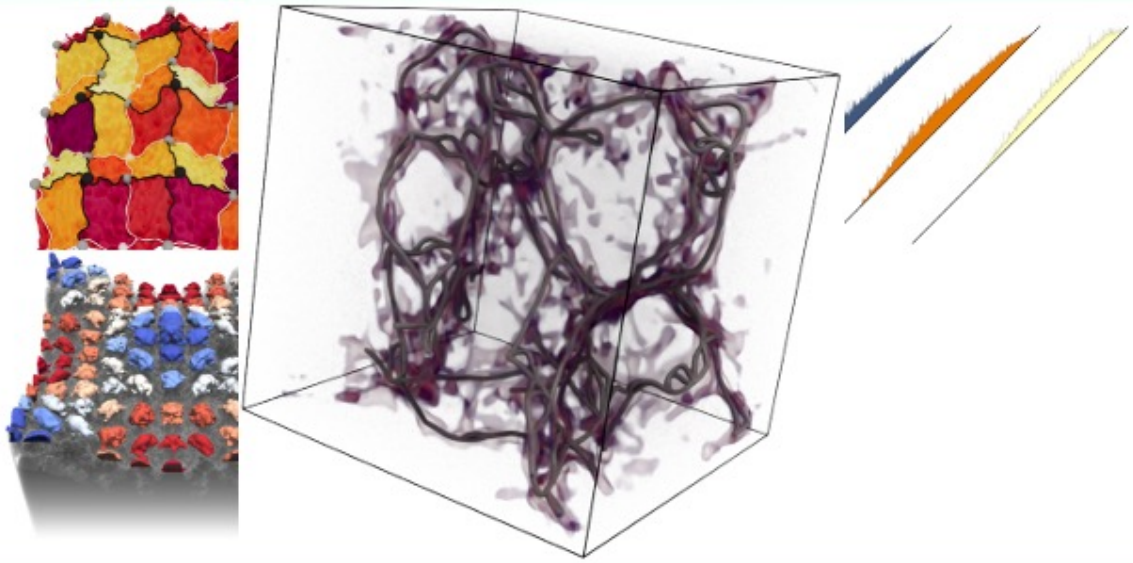
Motivation



M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 4

- segmented merge tree on the entropy field of a Richtmyer-Meshkov instability simulation
- Fluid dynamics simulations can scale to hundreds of thousands of processors

Motivation

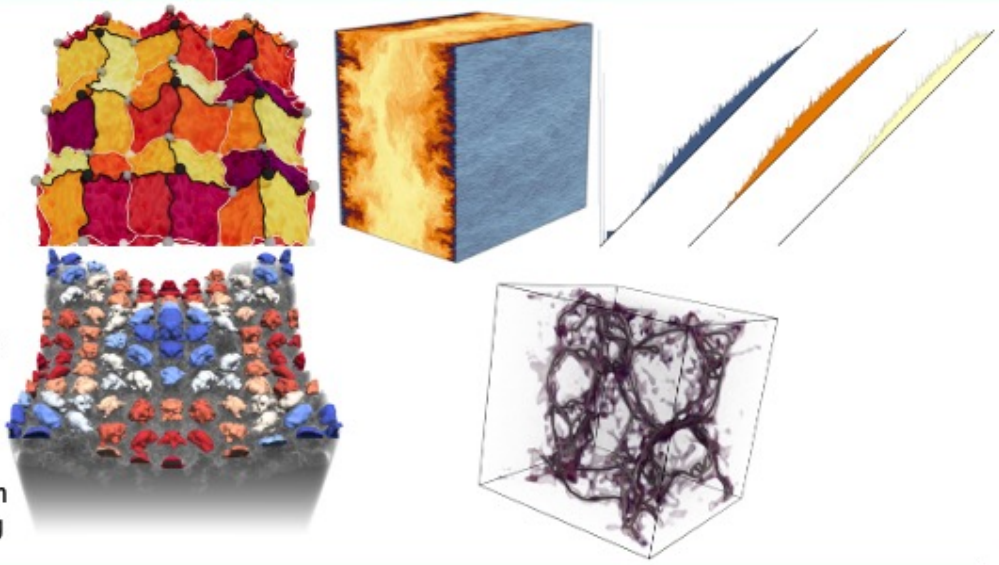


M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 5

- Extracting filaments and voids for cosmological density distributions
- The resulting topological abstractiosn may be simple, but the underlying data may be arbitrarily complex, Nyx cosmological hydrodynamics solver with terabytes of data

Motivation

- no general overview over different parallelization and distribution techniques exist
- Previous surveys (Heine et al. 2016, Yan et al. 2021) have focused on definitions and applications of TDA, but not the algorithmic scalability
- TDA captures *global* structure, but parallelism requires *local* processing



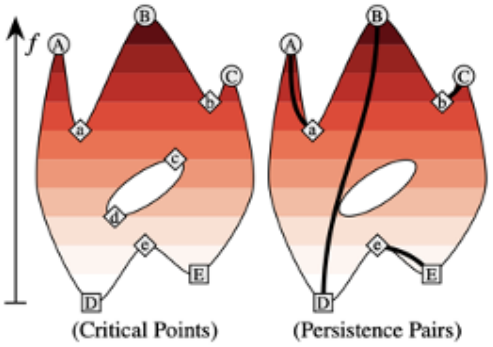
- no general overview over different parallelization and distribution techniques exist
- TDA captures *global* structure, but parallelism requires *local* processing

A very brief introduction to Computational Topology

...at least the parts relevant for our STAR

A very brief introduction to Computational Topology

- A scalar field assigns a value to every point in a domain, e.g. elevation in a mountain range
- **Critical points** are where the topology changes: minima (valleys), maxima (peaks), saddles (passes)
- As we sweep a threshold up through the data, connected components appear, merge, and disappear

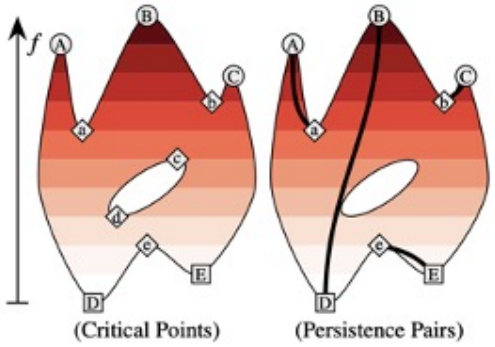


M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 8

- Data: piecewise linear scalar fields with data at vertices of d-dimensional connected simplicial complex. Values at edges and faces are interpolated

A very brief introduction to Computational Topology

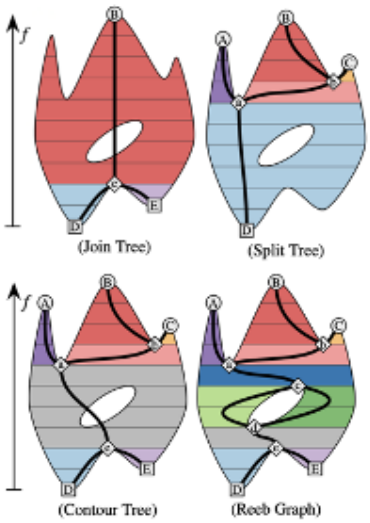
- Each feature is born at a critical point and dies when it merges with an older feature
- **Persistence** is how long a feature lives, measuring its importance
- Persistence Pairs and diagrams show the birth-death pairs



Elder rule

A very brief introduction to Computational Topology

- **Merge Trees**: track when connected components of sub/superlevel sets appear and merge
- **Contour trees**: track connectivity of level set contours on simply-connected domains, combination of upper and lower merge tree
- **Reeb graphs**: generalize contour trees to domains with holes and handles

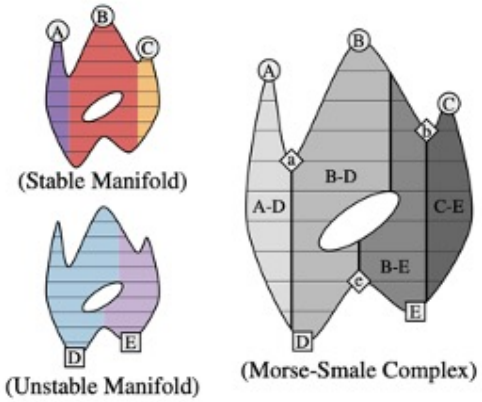


M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 10

- Sublevelset: all vertices which have a lower function value than an isovalue
- Reeb graphs: non simply connected

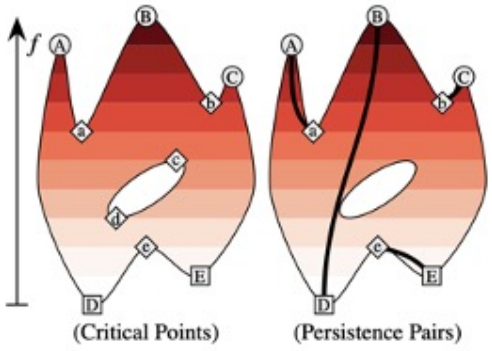
A very brief introduction to Computational Topology

- **Stable and unstable manifold**: partition the domain by gradient flow behavior, combine into the **Morse-Smale Complex**



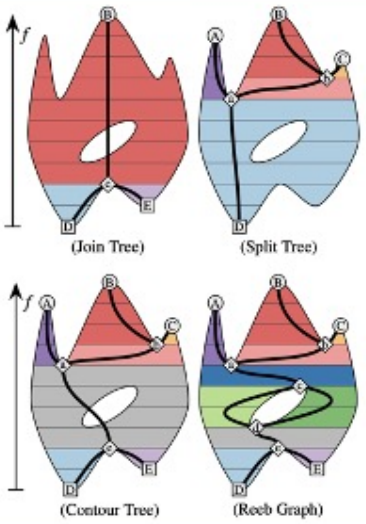
Overview

- survey paper about scalable topological data analysis
- 158 papers surveyed
- scalable computation models:
  - serial
  - shared memory parallel
  - distributed memory parallel
- relevant abstractions:
  - Set-based (Persistence Pairs / Diagrams)



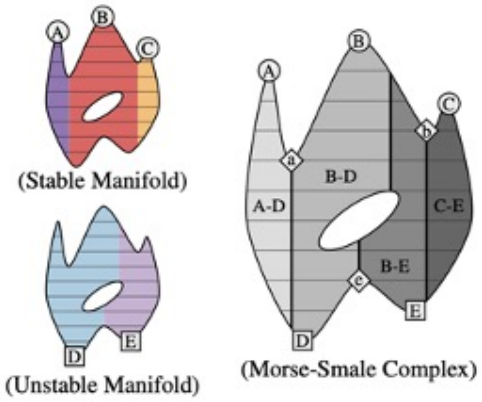
Overview

- survey paper about scalable topological data analysis
- 158 papers surveyed
- scalable computation models:
  - serial
  - shared memory parallel
  - distributed memory parallel
- relevant abstractions:
  - Set-based (Persistence Pairs / Diagrams)
  - Graph-based (Merge Tree, Contour Tree, Reeb Graph)



Overview

- survey paper about scalable topological data analysis
- 158 papers surveyed
- scalable computation models:
  - serial
  - shared memory parallel
  - distributed memory parallel
- relevant abstractions:
  - Set-based (Persistence Pairs / Diagrams)
  - Graph-based (Merge Tree, Contour Tree, Reeb Graph)
  - Complex-based (Morse-Smale Complexes)



Common building blocks

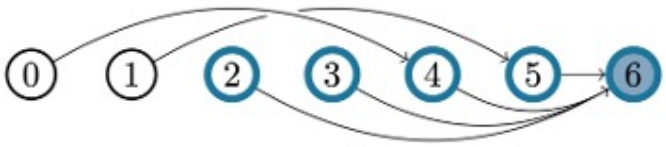
- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions

Common building blocks

- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions

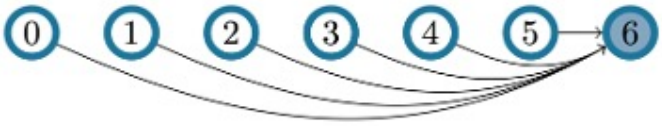
Common building blocks

- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions



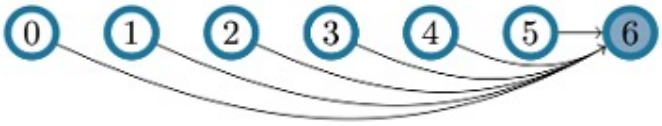
Common building blocks

- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions



Common building blocks

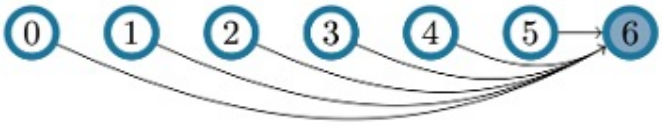
- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions
  - **Data-parallel primitives**: scan (prefix sum), sort, MapReduce; form the backbone of data-parallel algorithms like Parallel Peak Pruning



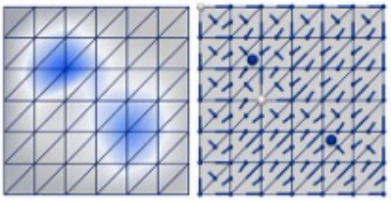
Input	1	2	3	4	5
Prefix Sum	1	3	6	10	15

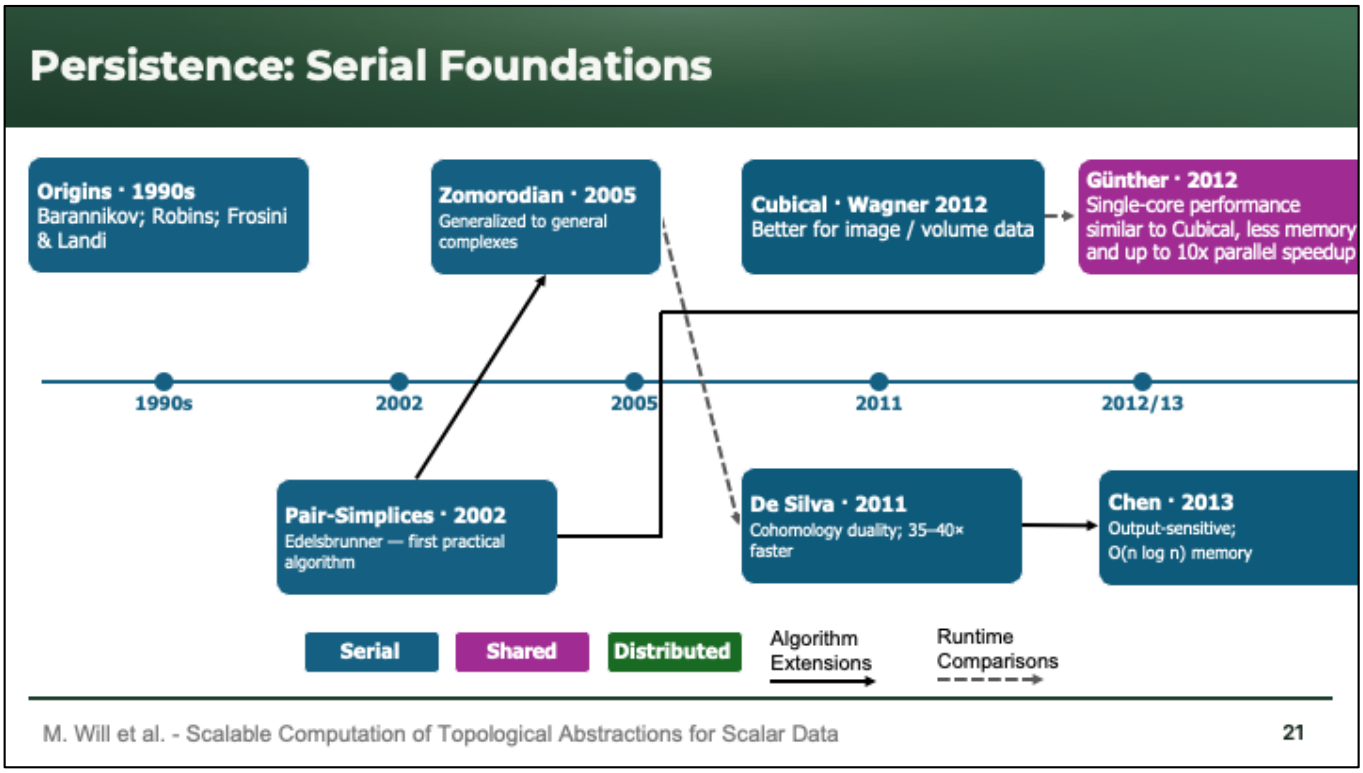
Common building blocks

- Many algorithms share the same low-level components, a "toolkit" for parallel TDA:
  - **Union-Find**: maintains connected components; parallelized via path compression and pointer jumping; used in some algorithms for all abstractions
  - **Data-parallel primitives**: scan (prefix sum), sort, MapReduce; form the backbone of data-parallel algorithms like Parallel Peak Pruning
  - **Discrete gradient**: embarrassingly parallel to compute; each vertex processed independently; crucial for Discrete Morse Theory-based approaches

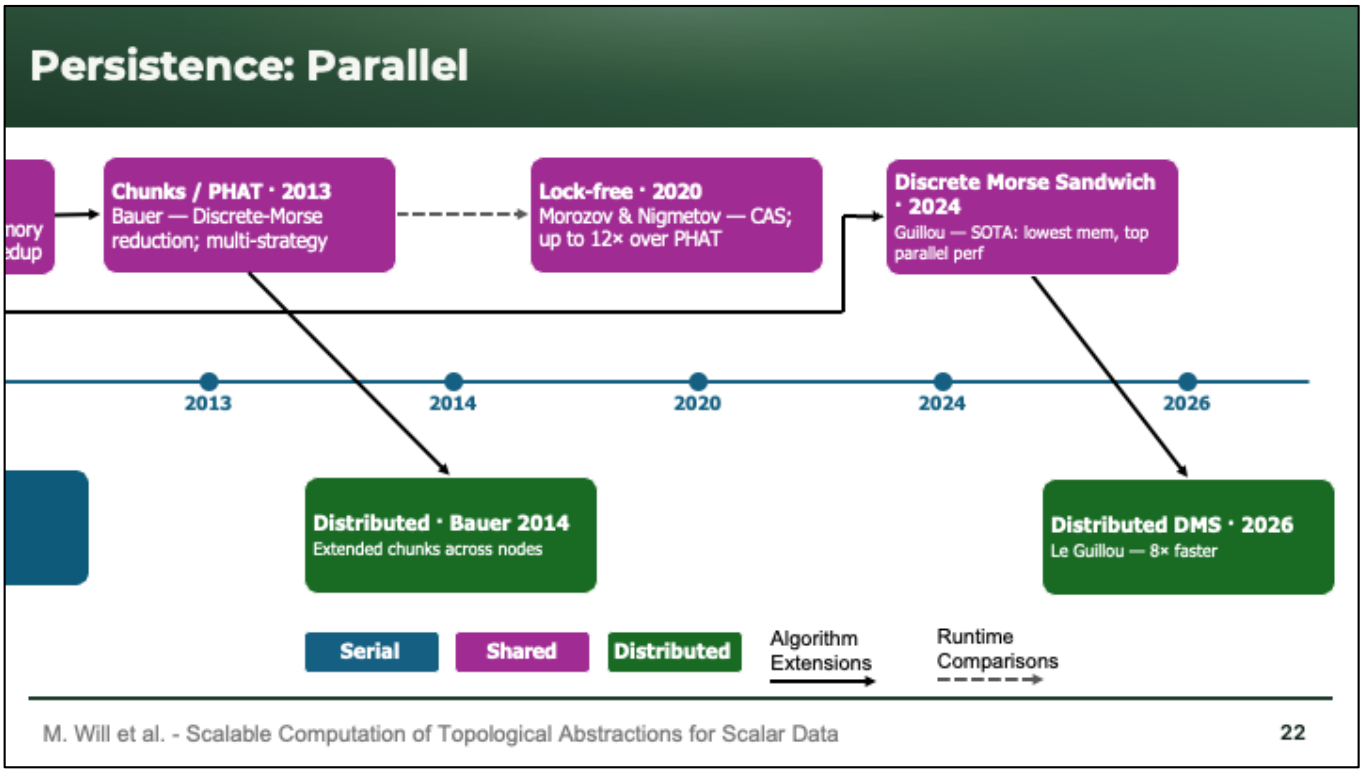


Input	1	2	3	4	5
Prefix Sum	1	3	6	10	15





- Persistence independently introduced by barannikov, robins and frosini&landi which partition critical values into birth death pairs by upper triangular matrix reduction
- Pair simplices captures and quantifies topological features across different filtration levels in a simplicial complex. also formally introduced the persistence diagram
- While the worst-case complexity of this algorithm is cubic, in practice it is significantly faster
- Zomorodian generalized it for higher dimensional data
- One branch: De Silva et al have presented a method exploiting dualities between homology and cohomology to accelerate persistence diagram computation
- This led to the first output sensitive algorithm by Chen et al. which solves previous reduction on rank computation on submatrices. Do not improve worst case complexity, but strong results, especially in memory
- Other branch: cubical complexes instead of triangular complexes



- Morozov: compare and swap
- DMS: combining Pair-Simplices with discrete morse theory and including aggressive specializations for 3D. strong memory and time improvements

Merge Trees: Serial Foundations

- Originally used for radar landscape simulation (Boyell & Ruston 1963)
- Intuition: imagine flooding a landscape: islands appear at maxima, merge at saddles
- Carr et al. (2003): standard algorithm using Kruskal + Union-Find in O(n log n + tα(t)) time
- Merge trees are a lightweight alternative to the full contour tree, especially useful for in-situ analysis

- Kruskals minimum spanning tree
- n vertices, t simplices, α is the inverse Ackermann function which grows very very slowly

Merge Trees: Shared Memory

- **FTM-Tree** (Gueunet et al. 2017): task-parallel Kruskal; propagations start at all maxima in parallel, synchronize at saddles; up to 8× faster than serial; memory-heavy for large sweep fronts

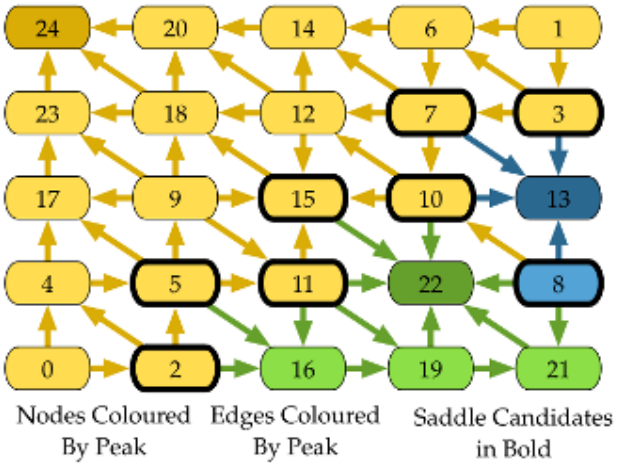
M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 24

- major drawback of this approach is maintenance of the entire sweep front using Fibonacci heaps, which require a lot of memory for large sweep fronts in large datasets
- Another unintended consequence of starting the tasks at the maxima is that the tasks are not evenly distributed, which can lead to load imbalances and lower parallel efficiency when increasing the size of the dataset but not the amount of the actual work done e.g. when resampling an initial dataset to a higher resolution for scaling experiments. While the merge tree may not increase in size, FTM-Tree will still require substantially more memory and time to compute the merge tree on the higher resolution dataset.

Merge Trees: Shared Memory

CARR H. A., WEBER G. H., SEWELL C. M., AHRENS J. P.: Parallel peak pruning for scalable SMP contour tree computation. LDAV 2016

- **FTM-Tree** (Gueunet et al. 2017): task-parallel Kruskal; propagations start at all maxima in parallel, synchronize at saddles; up to 8× faster than serial; memory-heavy for large sweep fronts
- **Parallel Peak Pruning** (Carr et al. 2016): data-parallel using sort and scan primitives; iteratively prunes leaf branches; up to 20× faster than FTM-Tree in 3D, Parallel Peak Pruning has various extensions for shared and distributed memory



- Instead of vector based approach, PPP uses data parallel primitives, identifying saddle candidates in the data by checking which monotone paths reach multiple peaks

Merge Trees: Shared Memory

CARR H. A., WEBER G. H., SEWELL C. M., AHRENS J. P.: Parallel peak pruning for scalable SMP contour tree computation. LDAV 2016

From Saddle Candidate	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Peak for Upper End	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	22	22	22	22	22	22	22	22	22	13	13	13	13
Lower End	15	11	10	10	8	7	7	7	5	5	5	3	3	2	2	15	11	11	11	10	8	8	5	2	10	8	7	3

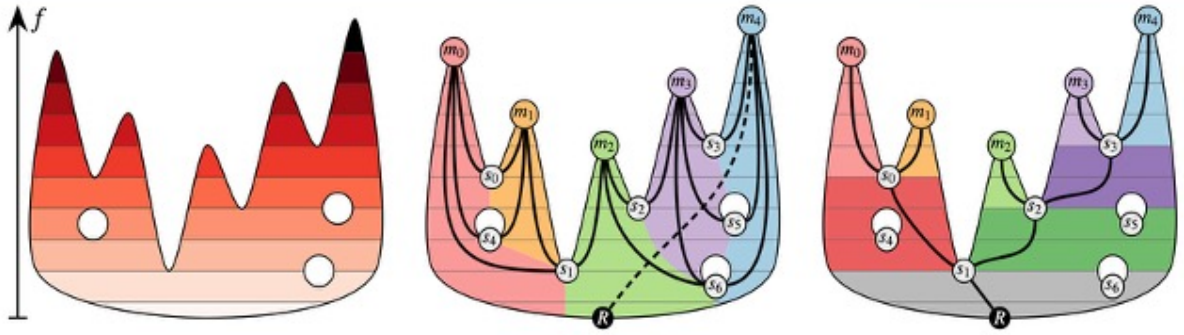
From Saddle Candidate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Peak for Upper End	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	24	22	22	22	22
Lower End	23	20	18	18	18	17	14	12	12	12	12	9	9	9	9	9	6	6	4	4	1	1	0	0	21	19	19	16

- **Parallel Peak Pruning** (Carr et al. 2016): data-parallel using sort and scan primitives; iteratively prunes leaf branches; up to 20× faster than FTM-Tree in 3D, Parallel Peak Pruning has various extensions for shared and distributed memory

- gov saddle g of a peak p is the highest saddle from which a monotone path to p exists
- And then pairing saddles and peaks in an sorted array
- Then all peaks higher than governing saddles are pruned and gov saddles either become regular points or peaks, and process is reiterated without pruned peaks

Merge Trees: Shared Memory

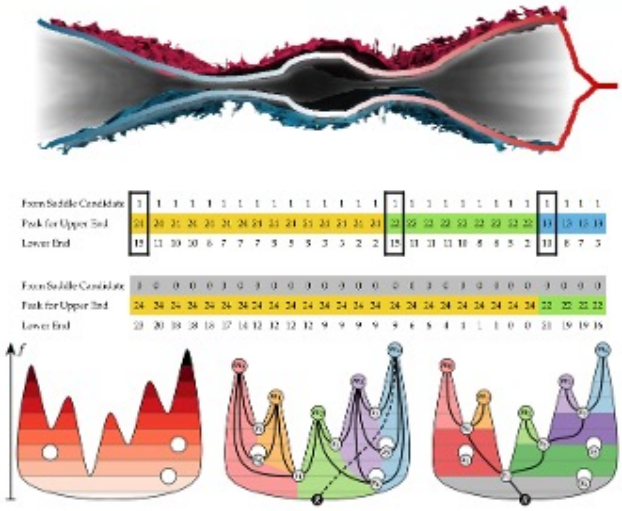
LUKASCZYK J., WILL M., WETZELS F., WEBER G. H., GARTH C.: ExTreeM: Scalable Augmented Merge Tree Computation via Extremum Graphs. IEEE TVCG 2023



- **ExTreeM** (Lukasczyk et al. 2023): first derives the extremum graph (orders of magnitude smaller than the domain), then runs merge tree computation on that; up to 2 orders of magnitude faster than FTM-Tree and up to 11x faster than PPP

Merge Trees: Shared Memory

- **FTM-Tree** (Gueunet et al. 2017): task-parallel Kruskal; propagations start at all maxima in parallel, synchronize at saddles; up to 8× faster than serial; memory-heavy for large sweep fronts
- **Parallel Peak Pruning** (Carr et al. 2016): data-parallel using sort and scan primitives; iteratively prunes leaf branches; up to 20× faster than FTM-Tree in 3D, Parallel Peak Pruning has various extensions for shared and distributed memory
- **ExTreeM** (Lukasczyk et al. 2023): first derives the extremum graph (orders of magnitude smaller than the domain), then runs merge tree computation on that; up to 2 orders of magnitude faster than FTM-Tree and up to 11x faster than PPP



M. Will et al. - Scalable Computation of Topological Abstractions for Scalar Data 28

- Note that PPP does not explicitly compute saddles, just saddle candidates. Those are vertices whose upper link is part of at least two different regions in the descending manifold, while ExTreeM‘s extremum graphs contains only the saddles and allows us to replace the multiple sortings in PPP with a simpler and more efficient maximum polling procedure

Merge Trees: Distributed Memory

- **Morozov & Weber** (2013): local-global representation where each processor stores a sparse version of the global tree; up to 14× faster than parallel contour tree computation
- **Werner & Garth** (2021): task-parallel distributed algorithm growing regions around critical points; up to 5× faster than Morozov & Weber with better scaling
- **Distributed merge forest** (Huang et al. 2021): localized trees connected by bridge sets; up to 14.6× faster workflows for large-scale queries

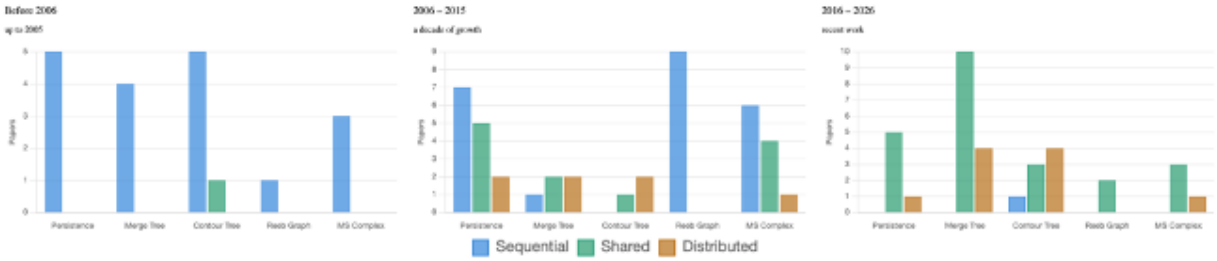
-Key use case: in-situ analysis of combustion simulations at volume size 2025×1600×400 on 30,000 cores

Contour Trees

- **Serial**:
  - Carr et al. (2003): established the standard O(n log n) algorithm
  - Raichel & Comandur (2022): first algorithm without a global sort
- **Shared memory**: Contour Forests (Gueunet 2016): partition domain by value, compute local trees, stitch; up to 4× faster than serial
- **Distributed**:
  - Li et al. (2024): first fully distributed augmented contour tree; up to 2 orders of magnitude faster than serial;
  - Li et al. (2025) adds pre-simplification for even larger datasets at 2.58× speedup and 6.67× lower memory

- Idea of Carr: merge the join tree and split tree (two merge trees) in a postprocessing step

Research Gaps



Research Gaps

Research Gaps

- **Reeb graphs**: burst of serial papers 2006–2015 bringing serial computation to maturity; only 1 paper after 2015; zero dedicated distributed algorithms
- **Morse-Smale complex**: heavy 2006–2015 activity; only 4 papers after 2015; distributed work can only handle simplified (not full) complexes
- **No unified benchmark**: surveyed works span decades and different hardware, a controlled reimplementation-based benchmark remains an open and valuable contribution

Open Problems and Research Gaps

- Distributed Reeb graph algorithms: contour tree algorithms partially fill the gap on simply-connected domains, but the general case has no solution
- Distributed Morse-Smale complex: current approaches (Gyulassy et al. 2012; Will et al. 2024) require local simplification or only compute segmentations, not the full complex
- Potential path forward: extend local-global representations (used successfully for merge/contour trees) to Reeb graphs and MS complexes
- ML-based TDA methods are evolving rapidly but currently lack exactness guarantees, possibly a future survey topic once the area matures

Conclusion

- TDA is powerful, but only useful at scale if we can compute it at scale
- Three-tier taxonomy: serial → shared memory → distributed memory, across five major abstractions
- Common building blocks (Union-Find, data-parallel primitives, discrete gradient) underpin many scalable algorithms
- Strong recent progress in persistence and merge/contour trees; significant gaps remain for Reeb graphs and MS complexes in distributed settings
- A unified benchmark across reimplemented methods on standardized hardware remains the most needed next contribution

- we hope that our work can guide future work by highlighting the state of the art and the gaps needing to be filled

Further information in the paper

- Applications

Further information in the paper

- Applications
- Performance comparisons

Shared Parallel			
Abstraction	Authors	Papers	Performance
Contour Trees	Pascucci et al.	Parallel Computation of the Topology of Level Sets [PCS04]	parallel speedup scales almost linearly
	Acharya et al.	A parallel and memory efficient algorithm for constructing the contour tree [AB15]	better performance and better parallel speedup and using only one-fifth of memory compared to [MB09,12]
	Gueunet et al.	Contour forests: Fast multi-threaded augmented contour trees [GFT16]	up to 6x faster than [CSA03], higher speedup than [MSN15]
		Task-based Augmented Contour Trees with Fibonacci Heaps [GFT19a]	extended version of [GFT17], up to 1.6x faster than it, up to 9x faster than a parallel variant of [CSA03], up to 4x times faster than [GFT16]
	Sharma et al.	On-Demand Augmentation of Contour Trees [SN20]	comparable to [GFT17], can selectively augment only regions of CT
Reeb Graph	Gueunet et al.	Task-based Augmented Reeb Graphs with Dynamic ST-Trees [GFT19b]	up to 3x faster than [Par11] using one thread, 4-38x faster using 32 threads
	Hajij et al.	An Efficient Data Retrieval Parallel Reeb Graph Algorithm [HB20]	no serial improvement over [DS12], parallel speedup of 2.8-4.3 on 6 cores
MS Complex	Günther et al.	Memory-Efficient Computation of Persistent Homology for 3D Images Using Discrete Morse Theory [GRSW11]	Coni with 1 critical points, scales only linearly with number of vertices, especially faster on noisy data than [BW11]
		Efficient computation of 3D Morse–Smale complexes and persistent homology using discrete Morse theory [GRW10,12]	improved [GRSW11], up to 20x faster than [BWS11]
	Shivashankar et al.	Parallel Computation of 2D Morse-Smale Complexes [SM12]	near linear parallel scaling
		Parallel Computation of 3D Morse-Smale Complexes [SN12]	orders of magnitude faster than [GGH09,10] and [GBP12], and less memory used
	Gyulassy et al.	Shared-Memory Parallel Computation of Morse-Smale Complexes with Improved Accuracy [GBP19]	30.77x faster than [GBP12], slower than [BW11] and [TFL+16], but more accurate
	Subhash et al.	GPU Parallel Computation of Morse-Smale Complexes [SPN20]	slower than [SN12] on small datasets due to GPU overhead, but up to 5x faster on larger datasets
	Maack et al.	Parallel Computation of Piecewise Linear Morse-Smale Segmentations [MLT+21]	more than an order of magnitude faster than [TFL+16] and 6.7x faster than [GBP19]

Distributed Parallel			
Abstraction	Authors	Papers	Performance
Persistence	Bauer et al.	Distributed Computation of Persistent Homology [BKR13]	comparable in time to [BKR14], but well scaling distributed memory usage
	Lewis et al.	Parallel Computation of Persistent Homology using the Blowup Complex [LM15]	well performance scaling, but high memory usage where distribution does not help
	Le Guillou et al.	Distributed Discrete Morse Sandwich: Efficient Computation of Persistence Diagrams for Massive Scalar Data [LGPT20]	comparable in time to [GAT21a], but well scaling distributed memory usage, on average 8x faster than [BKR13]
Merge Trees	Morozov et al.	Distributed merge trees [MW13]	up to 14x faster than [PCM04]
	Landge et al.	In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees [LPG+14]	comparable to [MW13], but better scaling to higher core counts
	Kicinsky et al.	Toward Localized Topological Data Structures: Querying the Forest for the Tree [KGBP20]	one order of magnitude faster than [GFTT19a] and [TFL+16], with well scaling distributed memory usage
	Wetzels et al.	Unordered Task-Parallel Augmented Merge Tree Construction [WG21]	up to 5x faster than [MW13] and [LPG+14], with better scaling
	Huang et al.	Distributed merge forest: a new fast and scalable approach for topological analysis at scale [HKP+21]	not computing the whole tree, but faster queries when contour trees are large, up to 11.6x faster workflows than [NM20]
Contour Trees	Jankar et al.	Iso-Contour Queries and Gradient Descent with Guaranteed Delivery in Sensor Networks [SZG+08]	no comparisons
	Morozov et al.	Distributed Contour Trees [MW14]	not computing the whole tree, but faster queries, good distributed scaling
	Nath et al.	Massively parallel algorithms for computing TIN DEMs and contour trees for large terrains [WSAM14]	distributed MPC model version using [CS03]
	Kintes et al.	W-Structures in Contour Trees [HC21]	abstract theory on W-structures in parallel contours trees
	Carr et al.	Distributed Hierarchical Contour Trees [CBW22]	significantly faster than [PCM04]
	Li et al.	Distributed Augmentation, Hypersweeps, and Branch Decomposition of Contour Trees for Scientific Exploration [LCB+24]	up to 2 orders of magnitude faster than serial [CWS+21]
		Extremely Scalable Distributed Computation of Contour Trees via Pre-Simplification [LCB+25]	better scaling and up to 2.16x faster than [LCB+24] with up to 6.67x lower memory usage

Further information in the paper

- Applications
- Performance comparisons
- Existing software packages

Software Package	Abstraction	Parallelization Types	Publications	Link	Language
TTK	Persistent Homology Merge Trees Contour Trees Reeb Graphs MS Complex	Shared Memory Distributed Memory (Persistent Homology)	[TFL*18, MBP*21, LGAWG*24]	https://topology-tool-kit.github.io/	C++ Python Bindings
Vaktoro	Merge Trees Contour Trees	Shared Memory Distributed Memory	[MSU*16, MMP*20, BMP*23]	https://vtk.org/	C++
PersistenceCycles	Persistent Homology	Shared Memory	[Jac22]	https://github.com/JacobCrf/PersistenceCycles	TTK Plugin C++ Python Bindings
Gudhi	Persistent Homology	Shared Memory	[MBGY16]	https://gudhi.inria.fr/	C++ Python Bindings
Dionysus	Persistent Homology	Serial	[Mor17]	https://www.mrzv.org/software/dionysus2/	C++ Python Bindings
Phat	Persistent Homology	Shared Memory	[BKRW17]	https://github.com/blazs/phat	C++
DIPHA	Persistent Homology	Shared Memory Distributed Memory	[BKR13]	https://github.com/DIPHA/dipha	C++
Javaplex	Persistent Homology	Serial	[AFV14]	https://appliedtopology.github.io/javaplex/	Java Matlab
Eirene	Persistent Homology	Serial	[Hen17]	https://github.com/Gnimuc/Eirene.jl	Julia
dionysus	Persistent Homology	Serial	[OPRSW24]	https://github.com/AppliedMathematicsSNU/dionysus	C++
Perseus	Persistent Homology	Serial	[MN13, Nan21]	http://people.maths.ox.ac.uk/nanda/perseus	C++
Oineus	Persistent Homology	Shared Memory	[MN20]	https://github.com/anigmetov/oineus/	C++ Python Bindings
Reeber	Merge Trees	Shared Memory Distributed memory	[NMO]	https://github.com/mrzv/reeber	C++
libcontre	Merge Trees	Serial	[CSA03], implements [CSA03]	https://github.com/wodiil/libcontre	C++
Cubicle	Persistent Homology	Shared Memory Distributed Memory	[Wag20, Wag18]	https://bitbucket.org/bmwag/cubicle/src/master/	C++
MSCEER	MS Complex	Shared memory	[GBP19]	https://github.com/sci-visus/MSCEER	C++
MScomplexes	MS Complex	Shared memory	[SN12, SMS12]	https://vgl.cse.iitc.ac.in/mscomplex/software.html	C++

Further information in the paper

- Applications
- Performance comparisons
- Existing software packages
- More details on presented papers