GPU Parallel Computation of Morse-Smale Complexes

Varshini Subhash* Karran Pandey† Vijay Natarajan‡

Department of Computer Science and Automation
Indian Institute of Science, Bangalore

2020 IEEE Visualization Conference (VIS) | 978-1-7281-8014-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/VIS47514.2020.00014

Abstract

The Morse-Smale complex is a well studied topological structure that represents the gradient flow behavior of a scalar function. It supports multi-scale topological analysis and visualization of large scientific data. Its computation poses significant algorithmic challenges when considering large scale data and increased feature complexity. Several parallel algorithms have been proposed towards the fast computation of the 3D Morse-Smale complex. The non-trivial structure of the saddle-saddle connections are not amenable to parallel computation. This paper describes a fine grained parallel method for computing the Morse-Smale complex that is implemented on a GPU. The saddle-saddle reachability is first determined via a transformation into a sequence of vector operations followed by the path traversal, which is achieved via a sequence of matrix operations. Computational experiments show that the method achieves up to 7× speedup over current shared memory implementations.

Index Terms: Human-centered computing—Visualization—Visualization techniques—; Computing methodologies—Parallel computing methodologies—Parallel algorithms—Shared memory algorithms;

1 Introduction

The Morse-Smale (MS) Complex [8,9] is a topological structure that provides an abstract representation of the gradient flow of a scalar function. It represents a decomposition of the domain of the scalar field into regions with uniform gradient flow behavior. Applications to feature-driven analysis and visualization of data from a diverse set of application domains including material science [16,21], cosmology [25], and chemistry [6,11] have clearly demonstrated the utility of this topological structure. A sound theoretical framework for identification of features, a principled approach to measuring the size of features, controlled simplification, and support for noise removal are key reasons for the wide use of this topological structure.

Large data sizes and feature complexity of the data have necessitated the development of parallel algorithms for computing the MS complex. All methods proposed during the past decade employ parallel algorithms that typically execute on a multicore CPU architecture. In a few cases, the critical point computation executes on a GPU. In this paper, we present a fast parallel algorithm that computes the graph structure of the MS complex via a novel transformation to matrix and vector operations. It further leverages data parallel primitives resulting in an efficient GPU implementation.

1.1 Related Work

The development of effective workflows for the analysis of large scientific data based on the MS complex, coupled with increasing compute power of modern shared-memory and massively parallel architectures has generated a lot of interest in fast and memory efficient parallel algorithms for the computation of 3D MS complexes. Gyulassy et al. [13] introduced a memory efficient computation of 3D MS complexes, where they handled large datasets that do not fit in memory. Their method partitions the data into blocks, called parcels, that fit in memory. Next, it computes the MS complex for the individual parcels and uses a subsequent cancellation based merging of individual parcels to compute the MS complex of the union of the parcels. This framework was extended in the design of distributed memory parallel algorithms developed by Peterka et al. [20] and Gyulassy et al. [18], where they additionally leverage high performance computing clusters to process the parcels in parallel. Subsequent improvements to parallelization were based on novel locally independent definitions for gradient pairs by Robins et al. [22] and Shivashankar and Natarajan [23] that allowed for embarrassingly parallel approaches for gradient assignment. Further improvements in computation time were achieved by efficient traversal algorithms for the extraction of ascending and descending manifolds of the extrema and saddles [7,12,23].

Graph traversals for computing the ascending and descending manifolds of extrema, owing to their relatively simple structure, have been modeled as root finding operations in a tree and subsequently parallelized on the GPU [23]. However, the best known algorithms for the more complex saddle-saddle traversals are still variants of a fast serial breadth first search traversal. More recently, Gyulassy et al. [14, 15, 17] and Bhatia et al. [6] have presented methods that ensure accurate geometry while computing the 3D MS complex in parallel. The approaches described above, with the exception of Shivashankar and Natarajan [23], implement CPU based shared memory parallelization strategies. Shivashankar and Natarajan [23] describe a hybrid approach and demonstrate the advantage of leveraging the many core architecture of the GPU. Embarrassingly parallel tasks such as gradient assignment and extrema traversals were executed on the GPU, resulting in substantial speedup over CPU based approaches.

1.2 Contributions

In this paper, we describe a fast GPU parallel algorithm for computing the MS complex. The algorithm employs the discrete Morse theory based approach, where the gradient flow is discretized to elements of the input grid. Different from earlier methods including the GPU algorithm developed by Shivashankar and Natarajan [23], it utilizes fine grained parallelism for all steps and hence enables an efficient end-to-end GPU implementation. The superior performance of our algorithm is due to two novel ideas for transforming the graph traversal operations into operations that are amenable to parallel computation:

- Breadth first search (BFS) tree traversal for determining saddle-saddle reachability is transformed into a sequence of vector operations.
- The 1-saddle – 2-saddle path computation is transformed into a sequence of inherently parallel matrix multiplication operations that correspond to wavefront propagation.

*e-mail: varshinis@iisc.ac.in
†e-mail: karranpandey@iisc.ac.in
‡e-mail: vijayn@iisc.ac.in

The saddle-saddle path computation was a major computational bottleneck in earlier methods. Experiments indicate that our algorithm is up to 7 times faster than existing methods. Further, the implementation uses highly optimized data parallel primitives such as prefix scan and stream compaction extensively, thereby leading to high scalability and efficiency.

2 Preliminaries

In this section, we briefly introduce the necessary background on Morse functions and discrete Morse theory that is required to understand the algorithm for computing the 3D MS complex.

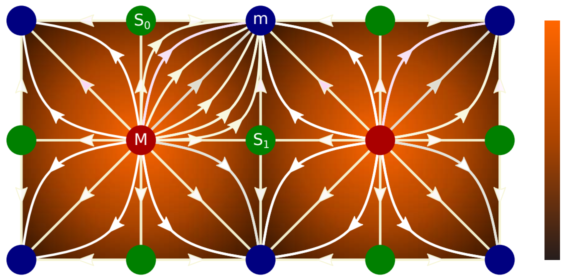

Figure 1: A 2D scalar function and its critical points (maxima - red, minima - blue, saddle - green) and reversed integral lines. A 2-cell MS_0mS_1 of the MS complex is a collection of all integral lines between m and M. A 1-cell (say, S_1m or MS_1) of the MS complex consists of the integral line between a minimum and a saddle or the integral line between a saddle and a maximum.

2.1 Morse-Smale Complex

Given a smooth scalar function $f : \mathbb{R}^3 \to \mathbb{R}$, the MS complex of f is a partition of $\mathbb{R}^3$ based on the induced gradient flow of f. A point p_c is called a *critical point* of f if the gradient of f at p_c vanishes, $\nabla f(p_c) = 0$. If the Hessian matrix of f is non-singular at its critical points, the critical points can be classified based on their *Morse index*, defined as the number of negative eigenvalues of the Hessian. Minima, 1-saddles, 2-saddles and maxima are critical points with index equal to 0,1,2, and 3 respectively. An *integral line* is a maximal curve in $\mathbb{R}^3$ whose tangent vector agrees with the gradient of f at each point in the curve. The origin and destination of an integral line are critical points of f. The set of all integral lines originating at a critical point p_c together with p_c is called the *ascending manifold* of p_c. Similarly, the set of all integral lines that share a common destination p_c together with p_c is called the *descending manifold* of p_c.

The *Morse-Smale (MS) complex* is a partition of the domain of f into cells formed by a collection of integral lines that share a common origin and destination. See Figure 1 for an example in 2D. The cells of the MS complex may also be described as the simply connected cells formed by the intersection of the ascending and descending manifolds. The 1-skeleton of the MS complex consists of nodes corresponding to the critical points of f together with the arcs that connect them. The 1-skeleton is often referred to as the combinatorial structure of the MS complex.

2.2 Discrete Morse Theory

Discrete Morse theory, introduced by Forman [10], is a combinatorial analogue of Morse theory that is based on the analysis of a discrete gradient vector field defined on the elements of a cell complex. Adopting an approach based on discrete Morse theory for computing the MS complex results in robust and computationally efficient methods with the added advantage of guarantees of topological consistency [13, 22, 23]. We focus on scalar functions defined on a 3D grid, a cell complex with cells of dimensions 0,1,2, or 3. If an i-cell α is incident on an $(i+1)$-cell β then α is called a *facet* of β and β is called a *cofacet* of α.

A *gradient pair* is a pairing of two cells $\langle \alpha^{(i)}, \beta^{(i+1)} \rangle$, where α is a facet of β. The gradient pair is a discrete vector directed from the lower dimensional cell to the higher dimensional cell. A *discrete vector field* defined on the grid is a set of gradient pairs where each cell of the grid appears in at most one pair. A *critical cell* with respect to a discrete vector field is one that does not appear in any gradient pair. The *index* of the critical point is equal to its dimension. A *V-path* in a given discrete vector field is a sequence of cells $\alpha_0^{(i)}, \beta_0^{(i+1)}, \alpha_1^{(i)}, \beta_1^{(i+1)} ...., \alpha_r^{(i)}, \beta_r^{(i+1)}, \alpha_{r+1}^{(i)}$ such that $\alpha_k^{(i)}$ and $\alpha_{k+1}^{(i)}$ are facets of $\beta_k^{(i+1)}$ and $\langle \alpha_k^{(i)}, \beta_k^{(i+1)} \rangle$ is a gradient pair for all $k = 0..r$. A V-path is called a *gradient path* if it has no cycles and a *discrete gradient field* is a discrete vector field which contains no non-trivial closed V-paths. Maximal gradient paths for a function defined on a grid correspond to the notion of integral lines in the smooth setting. We can thus similarly define ascending and descending manifolds for discrete scalar functions defined on a grid.

2.3 Parallel Primitives

Data parallel primitives serve as effective tools and building blocks in the design of parallel algorithms and we leverage them in all steps of our computational pipeline. Specifically, we use two primitives, *prefix scan* and *stream compaction*. Given an array of elements and a binary reduction operator, a prefix scan is defined as an operation where each array element is recomputed to be the reduction of all earlier elements [19]. If the reduction operator is addition, we call this the *prefix sum* operation. Given an array of elements, stream compaction produces a reduced array consisting of elements that satisfy a given criterion. We use Thrust [5], a popular library packaged with CUDA [1], for its prefix scan and stream compaction routines.

3 Algorithm

Discrete Morse theory based algorithms for computing the MS complex on a regular 3D grid consist of two main steps. The first step computes a well defined discrete gradient field and uses it to locate all critical cells. The second step computes ascending and descending manifolds as a collection of gradient paths that originate and terminate at critical cells. A combinatorial connection between two critical cells is established if there exists a gradient path between them. The collection of critical cells together with the connections is stored as the combinatorial structure of the MS complex.

We broadly follow the discrete Morse theory based approach employed by Shivashankar and Natarajan [23] for computing the MS complex. While the first step that identifies critical cells is similar to previous work [23], we introduce new methods for computing the gradient paths between saddles. Specifically, we introduce novel transformations that reduce the saddle-saddle gradient path traversal into highly parallelizable vector and matrix operations.

The difference in the structure of gradient paths that originate / terminate at extrema versus those originating / terminating at saddles necessitates different approaches for their traversals. Gradient paths that are incident on extrema can either split or merge but not both. So, traversing saddle-extrema gradient paths is analogous to a root finding operation in a tree. In this section, we restrict the description to the new methods and refer the reader to earlier work [23] for details on critical cell identification and saddle-extrema arc computation. We note here that our implementation of the critical cell and saddle-arc computation also incorporates improved methods for storage and transfer using data parallel primitives.

3.1 1-Saddle – 2-Saddle connections

Gradient paths between 1-saddles and 2-saddles can both merge and split. A sequence of merges and splits result in an exponential

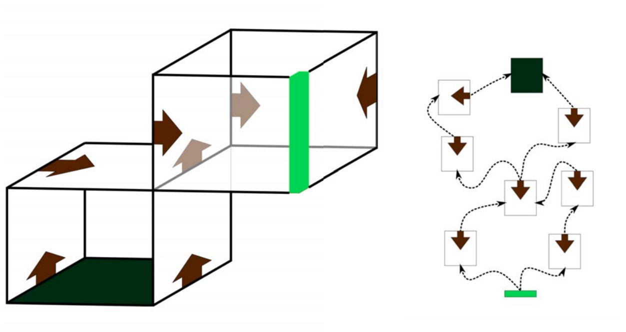

Figure 2: Gradient paths between 1-saddle (light green edge) and 2-saddle (dark green 2-cell) may both merge and split. Left: 1-2 gradient pairs (red arrows) constitute the gradient paths between the saddles. Right: A directed acyclic graph (DAG) representing the collection of gradient paths between the 1-saddle and 2-saddle. Dashed edges of the DAG represent incidence relationship between the 2-saddle or the 2-cell of a (1,2) gradient pair and the 1-cell of a (1,2) gradient pair or the 1-saddle. There are four possible gradient paths between the 1-saddle and 2-saddle due to the merge and split. A sequence of such configurations results in an exponential increase in the number of gradient paths. Figure adapted from [23], edges of the DAG are directed from the 1-saddle towards the 2-saddle.

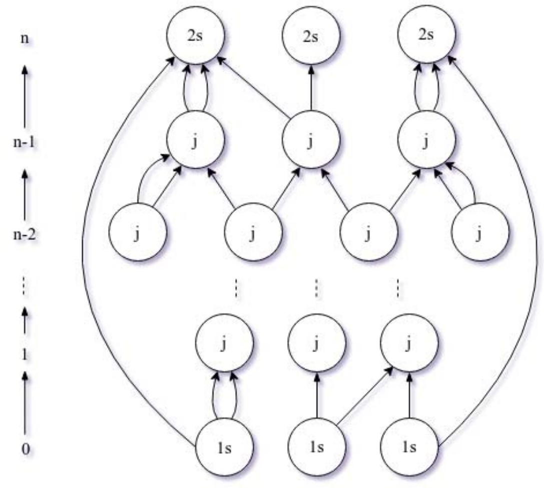

Figure 3: A minor G' of the DAG from Figure 2 computed by contracting all simple paths. Count paths between 1-saddles and 2-saddles by iteratively identifying paths of increasing length. The i^{th} iteration identifies all paths of length i, discovering the paths from nodes in level-0 to nodes in level-i. A node in level-i may be discovered again in a later iteration. In this case, the algorithm records both paths.

growth in the number of gradient paths, see Figure 2. Computing the saddle connections is the major computational bottleneck in current methods for computing the MS complex. The saddle-saddle arc computation is analogous to counting the paths between multiple sources and destinations in a directed acyclic graph (DAG). A trivial task based parallelization fails to work because of the storage required for each thread. Existing approaches tackle this problem by first marking reachable pairs of saddles using a serial BFS traversal. Next, they traverse paths between reachable pairs and count all paths using efficient serial traversal techniques.

We introduce transformations that enable efficient parallel computation of both steps, reachability computation and gradient path counting. The reachable pairs in the DAG are identified using a fine-grained parallel BFS algorithm [19] that is further optimized for graphs with bounded degree. In order to compute all gradient paths between reachable pairs, we first construct a minor of the DAG by contracting all simple paths to edges. We represent the minor using adjacency matrices and compute all possible paths between the reachable pairs via a sequence of matrix multiplication operations. These transformations enable scalable parallel implementations of both steps without the use of synchronization or locks.

3.2 Saddle-Saddle reachability

We mark reachable saddle-saddle pairs using parallel BFS traversals of the DAG starting at all 1-saddles. A BFS traversal iteratively computes and stores a frontier of nodes that are reachable from the source node. The frontier is initialized to the set of all source nodes, namely the 1-saddles. After the i^{th} iteration, the frontier consists of nodes reachable via a gradient path of length i. Note that the outdegree of a node in the DAG is at most 4, which imposes a bound on the size of the frontier in the next iteration. This bound allows us to allocate sufficient space to store the next frontier. All nodes in the current frontier are processed in parallel to update the frontier. Subsequently, a stream compaction is performed on the frontier to collect all nodes belonging to the next frontier. The traversal stops when it reaches a 2-saddle and the algorithm terminates when the frontier is empty.

3.3 Gradient path counting

We now restrict our attention to the subgraph of the 1-skeleton of the MS complex consisting of gradient paths between 1-saddles and 2-saddles. Edges belonging to the corresponding subgraph of the DAG are marked during the previous BFS traversal step. Let G denote this subgraph. Direct traversal and counting the number of paths between a pair of reachable saddles is again inherently serial. Instead, we first construct a minor G' of G by contracting all simple sub-paths in G. This transformation results in a graph G' whose nodes are either 1-saddles, 2-saddles, or *junction nodes*. A junction node is a node whose outdegree is greater than 1. Paths in G' are counted using a sequence of matrix multiplication operations, which are amenable to fine grained parallelism. We construct the minor in parallel and count the paths using matrix operations.

DAG minor construction. A path in the graph G is *simple* if none of its interior nodes is a junction node. We compute a minor G' of G by contracting all simple paths between 1-saddles, 2-saddles and junction nodes into edges between their source and destination nodes. We process all 1-saddles and junction nodes in parallel to trace all paths that originate at that node. The paths terminate when they reach a junction node or a 2-saddle. So, all paths are guaranteed to reach their destination without splits. Note that some of the paths may merge. However, such paths originate from different source nodes and are processed by different threads.

All path traversals share a common array to record the most recent visited node in each path. In each iteration, all paths advance by a single node. The paths terminate if they reach a 2-saddle or a junction node. We perform a stream compaction on the array at the end of the iteration to remove terminated paths and to ensure that all threads are doing useful work. The algorithm terminates when all paths have been marked as terminated. Again, the bound on the outdegree of the nodes in the DAG implies that at most four paths originate at a node. So, their adjacency lists have a fixed size and hence the size of the array maintained by the algorithm is bounded. We note that storing the adjacencies for each source node implicitly records paths of multiplicity greater than one.

Path counting via matrix operations. Nodes of the graph minor

Dataset	Size	Number of Critical Points	pyms3d [23] (*secs*)	Our Algorithm (*secs*)	MS Complex Speedup	Parallel BFS Speedup	Path Counting Speedup
Fuel	64×64×64	783	0.05	0.45	0.11	2.97	0.05
Neghip	64×64×64	6193	0.29	0.49	0.59	22.30	0.20
Silicium	98×34×34	1345	0.07	0.43	0.16	14.09	0.07
Hydrogen	128×128×128	26725	0.91	0.69	1.33	35.41	0.45
Engine	256×256×128	1541859	26.86	5.40	4.98	108.41	2.38
Bonsai	256×256×256	567133	39.58	5.61	7.05	79.10	4.35
Aneurysm	256×256×256	97319	8.00	1.30	6.14	72.86	4.51
Foot	256×256×256	2387205	45.10	8.52	5.29	129.13	2.82
Isabel-Precip	500×500×100	9528383	37.88	7.87	4.81	82.90	1.73

Table 1: MS Complex computation times for our GPU parallel algorithm compared with pyms3d [23,24]. We observe good speedups in runtime, particularly for the larger datasets. Both saddle-saddle reachability and gradient path counting algorithms contribute to the improved running times.

G' constructed via path contraction may be placed in levels as shown in Figure 3. We convert the adjacency list representation of G' into sparse matrices and count the number of paths between all 1-saddle – 2-saddle pairs using iterative matrix multiplication. Edges in G' may be classified into four different types: $1s-j$, $j-j$, $j-2s$ and $1s-2s$. Let A_{1s-j}, B_{j-j}, B^*_{j-2s}, and D_{1s-2s} represent the matrices that store the respective edges.

Consider nodes belonging to consecutive levels in Figure 3, in particular between level-0 and level-1. We first multiply $A_{1s-j} \times B_{j-j}$ and store the result in A_{1s-j}, which establishes connections between level-0 and level-2 nodes. Each successive multiplication of A_{1s-j} with B_{j-j} establishes connections between level-0 and the newly discovered level. A special case arises in the traversal when a node is revisited at multiple levels. Such a node appears at different depths from the source and each unique visit should be recorded. We update the sequence of matrix operations to ensure that these special cases are handled correctly by maintaining a storage matrix A^*_{1s-j} that collects all newly discovered paths after each iteration. We describe the procedure in detail below.

Initialize four matrices with their respective edges as follows: A_{1s-j}, B_{j-j}, B^*_{j-2s}, D_{1s-2s}. We maintain A^*_{1s-j}, which iteratively records newly discovered paths. Each iteration multiplies A_{1s-j} with B_{j-j} to yield a matrix C_{1s-j}, after which we add matrix A_{1s-j} to A^*_{1s-j}. The newly discovered paths in C_{1s-j} become the input frontier A_{1s-j} for the next iteration's multiplication.

After the final iteration, A^*_{1s-j} contains all possible paths from 1-saddles to junction nodes. The purpose of A^*_{1s-j} is two-fold: first, it records shorter paths that terminate during intermediate iterations. Next, nodes that are discovered multiple times add their incrementally discovered paths to it after each multiplication. This matrix is then multiplied by B^*_{j-2s} to obtain D^*_{1s-2s}, establishing connections between 1-saddles and 2-saddles. Edges between 1-saddles and 2-saddles in G' are stored in D_{1s-2s}. In the final step, D_{1s-2s} is added to D^*_{1s-2s} to record the set of paths with multiplicity between all 1-saddles and 2-saddles. Matrices are stored in their sparse forms, thereby reducing the memory footprint. We use the CUDA sparse matrix library cuSPARSE [2] for matrix multiplication operations.

4 Experiments

We perform all experiments on an Intel Xeon Gold 6130 CPU @ 2.10 GHz powered workstation with 16 cores (32 threads) and 32 GB RAM. GPU computations were performed on an Nvidia GeForce GTX 1080 Ti card with 3584 CUDA cores and 11 GB RAM. We use nine well known datasets for testing [4].

Our computational pipeline comprises of assigning gradient pairs based on the scalar field, identifying critical points, computing the 1–2 saddle connections and finally computing the extrema gradient paths. This gives us the combinatorial structure of the MS complex and the ascending and descending manifolds of critical points. Each step of the pipeline is performed on the GPU. We compare our results with pyms3d [3,23,24], which presently report the best runtimes for computing the MS complex on shared memory processors. We set the number of CUDA threads per block as 512, with 64 blocks in a grid. We ensure identical parameters when comparing our results with [23,24]. The results reported by Gyulassy et al. [14,15,17] and Bhatia et al. [6] focus on improving geometric accuracy, thus yielding larger runtimes.

5 Results

Table 1 compares the runtime and shows the speedups for individual stages of the pipeline. The parallel saddle-saddle reachability algorithm performs better on all datasets, achieving up to 129× speedup (Foot). The speedups increase with the number of critical points as expected. The path counting algorithm also performs better for larger datasets, achieving up to 4.51× speedup (Aneurysm). The number of saddle points and junction nodes in the smaller datasets are multiple orders of magnitude smaller than those in the larger datasets. The overheads of GPU data transfer time and construction of matrices dominate the runtime resulting in poor scaling. In contrast, pyms3d [23] processes each 2-saddle in parallel and each traversal originating at a 2-saddle terminates early given the fewer number of junction nodes. This is likely the reason for its superior performance on the smaller datasets. We observe 1.33× to 7× speedup for the overall MS complex computation for larger datasets. The path counting step dominates the runtime and hence affects the overall speedup.

6 Conclusions

We have introduced a novel parallel algorithm that computes the MS complexes of 3D scalar fields defined on a grid. It is the first completely GPU parallel algorithm developed for this task and results in superior performance with notable speedups for large datasets. Our approach is the first of its kind to leverage parallel data primitives coupled with matrix multiplication as a means of graph traversal. We also ensure a small memory footprint and a completely parallel approach devoid of locks or synchronization. We are presently developing a parallel algorithm for MS complex simplification.

Acknowledgments

This work is partially supported by a Swarnajayanti Fellowship from the Department of Science and Technology, India (DST/SJF/ETA-02/2015-16), a Mindtree Chair research grant, and an IoE grant from Indian Institute of Science, Bangalore.

References

[1] CUDA Zone. https://developer.nvidia.com/cuda-zone, 2020. [Online; accessed 03-July-2020].

[2] cusparse. `https://developer.nvidia.com/cusparse`, 2020. [Online; accessed 12-July-2020].

[3] MSComplex. `https://vgl.csa.iisc.ac.in/mscomplex`, 2020. [Online; accessed 20-August-2020].

[4] Open scientific visualization datasets. `https://klacansky.com/open-scivis-datasets/`, 2020. [Online; accessed 10-July-2020].

[5] Thrust. `https://developer.nvidia.com/thrust`, 2020. [Online; accessed 03-July-2020].

[6] H. Bhatia, A. G. Gyulassy, V. Lordi, J. E. Pask, V. Pascucci, and P.-T. Bremer. Topoms: Comprehensive topological exploration for molecular and condensed-matter systems. *Journal of computational chemistry*, 39(16):936–952, 2018.

[7] O. Delgado-Friedrichs, V. Robins, and A. Sheppard. Skeletonization and partitioning of digital images using discrete Morse theory. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):654–666, 2014.

[8] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. Symposium on Computational Geometry*, pp. 361–370, 2003.

[9] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. *Discrete and Computational Geometry*, 30(1):87–107, 2003.

[10] R. Forman. A user's guide to discrete Morse theory. *Sém. Lothar. Combin*, 48:35pp, 2002.

[11] D. Günther, R. A. Boto, J. Contreras-Garcia, J.-P. Piquemal, and J. Tierny. Characterizing molecular interactions in chemical systems. *IEEE Transactions on Visualization and Computer graphics*, 20(12):2476–2485, 2014.

[12] D. Günther, J. Reininghaus, H. Wagner, and I. Hotz. Efficient computation of 3d Morse–Smale complexes and persistent homology using discrete Morse theory. *The Visual Computer*, 28(10):959–969, 2012.

[13] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008.

[14] A. Gyulassy, P.-T. Bremer, and V. Pascucci. Computing Morse-Smale complexes with accurate geometry. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2014–2022, 2012.

[15] A. Gyulassy, P.-T. Bremer, and V. Pascucci. Shared-memory parallel computation of Morse-Smale complexes with improved accuracy. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):1183–1192, 2018.

[16] A. Gyulassy, M. Duchaineau, V. Natarajan, V. Pascucci, E. Bringa, A. Higginbotham, and B. Hamann. Topologically clean distance fields. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1432–1439, 2007.

[17] A. Gyulassy, D. Günther, J. A. Levine, J. Tierny, and V. Pascucci. Conforming Morse-Smale complexes. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2595–2603, 2014.

[18] A. Gyulassy, V. Pascucci, T. Peterka, and R. Ross. The parallel computation of Morse-Smale complexes. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pp. 484–495. IEEE, 2012.

[19] D. Merrill, M. Garland, and A. Grimshaw. Scalable gpu graph traversal. *Acm Sigplan Notices*, 47(8):117–128, 2012.

[20] T. Peterka, R. Ross, A. Gyulassy, V. Pascucci, W. Kendall, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In *2011 IEEE Symposium on Large Data Analysis and Visualization*, pp. 105–112. IEEE, 2011.

[21] S. Petruzza, A. Gyulassy, S. Leventhal, J. J. Baglino, M. Czabaj, A. D. Spear, and V. Pascucci. High-throughput feature extraction for measuring attributes of deforming open-cell foams. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):140–150, 2019.

[22] V. Robins, P. J. Wood, and A. P. Sheppard. Theory and algorithms for constructing discrete Morse complexes from grayscale digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1646–1658, 2011.

[23] N. Shivashankar and V. Natarajan. Parallel computation of 3D Morse-Smale complexes. *Computer Graphics Forum*, 31(3pt1):965–974, 2012.

[24] N. Shivashankar and V. Natarajan. Efficient software for programmable visual analysis using Morse-Smale complexes. pp. 317–331, 06 2017. doi: 10.1007/978-3-319-44684-4_19

[25] N. Shivashankar, P. Pranav, V. Natarajan, R. van de Weygaert, E. P. Bos, and S. Rieder. Felix: A topology based framework for visual exploration of cosmic filaments. *IEEE Transactions on Visualization and Computer Graphics*, 22(6):1745–1759, 2015.