2012 IEEE 26th International Parallel and Distributed Processing Symposium

The Parallel Computation of Morse-Smale Complexes

Attila Gyulassy, Valerio Pascucci
Scientific Computing and Imaging Institute
Dept. of Computer Science, University of Utah
Salt Lake City, United States of America
jediati@sci.utah.edu, pascucci@sci.utah.edu

Tom Peterka, Robert Ross
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, United States of America
tpeterka@mcs.anl.gov, rross@mcs.anl.gov

***Abstract*—Topology-based techniques are useful for multi-scale exploration of the feature space of scalar-valued functions, such as those derived from the output of large-scale simulations. The Morse-Smale (MS) complex, in particular, allows robust identification of gradient-based features, and therefore is suitable for analysis tasks in a wide range of application domains. In this paper, we develop a two-stage algorithm to construct the Morse-Smale complex in parallel, the first stage independently computing local features per block and the second stage merging to resolve global features. Our implementation is based on MPI and a distributed-memory architecture. Through a set of scalability studies on the IBM Blue Gene/P supercomputer, we characterize the performance of the algorithm as block sizes, process counts, merging strategy, and levels of topological simplification are varied, for datasets that vary in feature composition and size. We conclude with a strong scaling study using scientific datasets computed by combustion and hydrodynamics simulations.**

***Keywords*-Morse-Smale complex; Parallel topological analysis**

I. Introduction

The expanding computational power of modern supercomputers enables simulations to generate data with greater resolution and complexity than ever before. Furthermore, as sensors gain resolution and the size of commodity storage increases, the same trend is observed for captured data, for example, for CT scans and confocal microscopy mosaics. Sophisticated analysis techniques that scale with the explosion in data size and complexity are necessary for the effective analysis of such data. Topology-based graph structures are a promising approach because they enable a multi-resolution representation that summarizes important features and can be explored interactively.

The Morse-Smale (MS) complex, a segmentation of a scalar field into regions of uniform gradient flow behavior, is one such topological structure. It is an unstructured graph with edges representing the topological connectivity of features and nodes storing their attributes and geometric embedding. As a cellular decomposition of the domain, it can provide a map to the feature space defined by gradient flow properties that can be orders of magnitude smaller than the input data, depending on the original complexity. Furthermore, the process of defining features is reduced to designing interactive queries on the graph structure. This allows scientists to conduct parameter studies without the need to rerun analyses on the original data.

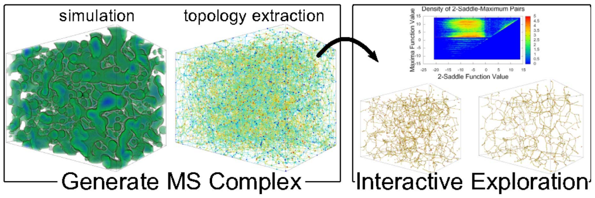


Figure 1. The analysis pipeline using MS complexes starts with acquisition of simulation data and computation of a fine-scale complex (left). All subsequent analysis queries this structure. A scientist may interactively visualize statistics about the topological structure of the data or select different threshold values to define features. Such exploration provides immediate feedback to the scientist with visualization of the extracted features as well as statistics generated on-the-fly (right).

To motivate our work, Figure 1 contains a small example of how the MS complex can answer underlying science questions. The remainder of this paper then demonstrates how to generate similar complexes as in the left side of this figure, in parallel and at very large scale. In Figure 1 the MS complex is used to find the filament structure of a simulated porous material [12], represented as a signed volumetric distance field from an uncertain interface demarcating the interior and exterior of the material. The MS complex traces the potential locations of the filament structures, or three-dimensional ridge-lines.

After the MS complex is computed from the data in the left side of Figure 1, it is explored interactively on the right. This interactive exploration enables viewing the filament structures for multiple threshold values and at multiple topological scales, making possible a parameter study of the impact and stability of the choice of threshold values. As an embedded graph, the filaments can be analyzed using graph algorithms, extracting statistics such as length, cycle count, and the minimum cut. This kind of approach has similar applications in a wide variety of disciplines including physics, computational chemistry, combustion, biomedical imaging, astronomy, and oil exploration.

The bottleneck until now in the analysis pipeline just described has been initial computation of the complex,

1530-2075/12 $26.00 © 2012 IEEE
DOI 10.1109/IPDPS.2012.52

requiring both significant memory resources and processing time. Given the size and complexity of many datasets, computing the MS complex requires HPC resources, often the same supercomputers used to compute the original data. Two main challenges must be addressed to compute the MS complex in parallel.

First, features often have global scope, and the spatial decomposition imposed by a parallel approach must be overcome to resolve them. When a data set is divided into spatial blocks to be processed in parallel, gradient flow features identified locally may in fact be part of much larger ones spanning several blocks. A globally consistent resolution of these features is necessary for robust analysis, and the process of resolving the features inevitably requires potentially expensive interprocess communication.

Second, the MS complex encoding the segmentation at the finest topological resolution may have a prohibitively large memory footprint due to an abundance of insignificant features or noise. A multi-resolution representation is needed to simplify the fine-scale complex, where typically only the coarsest levels of the hierarchy may represent features of interest. However, a complete simplification requires knowledge about the global connectivity of the complex, limiting the simplification that can be performed independently.

In this paper, we designed, implemented, and tested a novel parallel algorithm to construct the 1-skeleton of the MS complex. Our solution features tunable parameters that allow trade offs in use of resources, output size, and accuracy. These parameters include blocking strategy, merging strategy, and simplification level of the topology. The key to the scalability of our approach is to balance the degree of simplification with communication load, based on understanding how the parallel performance of topology-based techniques is impacted by the distribution and size of features in the original data.

To characterize our algorithm's performance, we conducted extensive parameter studies using artificial data as well as actual scientific data. The synthetic datasets are designed to test best-case and worst-case performance, where feature size and count are varied in a range of data sizes and processor counts. The scientific datasets demonstrate strong scaling on results of large-scale simulations of combustion and hydrodynamic simulations. These results indicate guiding principles that can further help design the next generation of scalable topology-based analysis algorithms with the goal of deploying them in situ with simulations.

This is the first parallel algorithm for computing the MS complex that scales efficiently to tens of thousands of nodes of distributed-memory HPC supercomputers such as the IBM Blue Gene/P. Enabling the large-scale parallel computation of the MS complex opens up new possibilities for scientists who can use the MS complex for further simplification, filtering, feature extraction, and quantitative analysis of their data.

II. Related Work

The MS complex is a topological data structure that provides an abstract representation of the gradient flow behavior of a scalar field [29], [28] that is beginning to make an impact in analysis of large-scale scientific data. For example, Laney et al. [20] used the descending 2-manifolds of a two-dimensional MS complex to segment an interface surface and count bubbles in a simulated Rayleigh-Taylor instability. Bremer et al. [2] used a similar technique to count the number of burning regions in a lean premixed hydrogen flame simulation. Gyulassy et al. [12] used carefully selected arcs from the 1-skeleton of the three-dimensional MS complex to analyze the core structure of a porous solid.

Efficient computation of the MS complex for large volumetric data is still an open challenge. The first algorithm for two-dimensional piecewise-linear (PL) data was presented by Edelsbrunner et al. [6]. Bremer et al. [1] improved this by following gradients more faithfully and described a multi-resolution representation of the scalar field. Several algorithms have been proposed to compute the complex for volumetric data [5], [13], [14], [15], however, these techniques are limited by computational overhead to small, simple data sets.

A discrete interpretation of Morse theory, as presented by Forman [8], simplifies the construction of the complex by discretizing gradient flow. In this approach, a discrete gradient vector field is computed that uniquely determines the combinatorial structure of the MS complex. The main computational aspect in using a discrete approach is generating a discrete gradient vector field. Lewiner et al. [21] showed how a discrete gradient field can be constructed and used to identify the MS complex, however, this construction required an explicit graph-based representation of gradient paths, prohibitively expensive for large volumetric data. King et al. [19] presented a method for constructing a discrete gradient field that agrees with the large-scale flow behavior of the data defined at vertices of the input mesh. In our approach, we use the parallel algorithm presented by Gyulassy et at. [10] for its simplicity of implementation and its dynamic simulation of simplicity, that greatly reduces the number of zero-persistence critical points found. Although this algorithm was presented as parallelizable, no implementation had been achieved.

One challenge to realizing a parallel implementation is managing communication in a data-intensive algorithm. Parallel analysis algorithms containing global features in a distributed architecture need to communicate this information between processes. The computational time of these algorithms often scales well, making parallel analysis bound by data movement, as [25] demonstrated for volume rendering. The dominant communication patterns seen in analysis algorithms are local neighborhood communication, as in particle tracing [24], [26] and global reduction, as in image compositing [30]. Elements of our configurable merge algorithm are motivated by a configurable image compositing algorithm called Radix-k, shown to have good scalability at the full scale of HPC machines [22].

The second stage of our algorithm, merging, is an example of graph simplification. Graph simplification approaches are used by the information visualization community in order to view and interact with complex graph structures, such as the sequential simplification algorithm presented by Hennessey et al. [17]. Parallel graph simplification is a component of multilevel graph partitioning algorithms. This stage is usually called coarsening and can be found in the ParMeTiS [27] and PT-Scotch libraries [3]. These libraries, however, do not provide the exact graph simplification that we need, and they have been shown to scale to only 128 processes. The same is true of the Parallel BGL [9].

III. Background

Discretization is the fundamental tool used in the topological analysis of scalar functions available as samples on a grid. In the following, we review concepts from Morse theory, and present their discrete analogue that is the basis for practical algorithms. Finally, we review topology-based simplification.

A. Morse Functions and the MS Complex

Let f be real-valued smooth map $f : \mathbb{M} \to \mathbb{R}$ defined over a compact d-manifold $\mathbb{M}$. A point $p \in \mathbb{M}$ is critical when $\nabla f(p) = \mathbf{0}$, that is, the gradient is zero, and is non-degenerate when its Hessian (matrix of second partial derivatives) is non-singular. The function f is a *Morse function* if all its critical points are non-degenerate and no two critical points have the same function value. Morse functions are *dense* in the space of functions, that is, any function can be closely approximated by a Morse function, a fact that makes them useful in the analysis of real-world data. The *Morse Lemma* states that there exist local coordinates in a neighborhood around p such that f has the following *standard form*: $f_p = c \pm x_1^2 \pm x_2^2 \cdots \pm x_n^2$. The number of minus signs in this equation gives the *index* of critical point p. In three-dimensional functions, minima are index-0, 1-saddles are index-1, 2-saddles are index-2, and maxima are index-3.

An integral line in f is a path in $\mathbb{M}$ whose tangent vectors agree with the gradient of f at every point along the path. The upper and lower limits of an integral line are its *destination* and *origin*, respectively. *Ascending* and *descending* manifolds are obtained as clusters of integral lines having common origin and destination, respectively. A Morse function f is a *Morse-Smale function* if its ascending manifolds intersect descending manifolds only transversally. The intersection of the ascending and descending manifolds of a Morse-Smale function forms the *Morse-Smale (MS) complex*. Of key importance in the analysis of scalar functions is the 1-skeleton of the MS complex, formed by the 0- and 1-dimensional elements. The 0-dimensional elements are critical points, and are called *nodes* in the complex, and the 1-dimensional elements are integral lines connecting critical points differing in dimension by 1, and are called *arcs* of the complex. Figure 2(b) illustrates the 1-skeleton of the complex on a simple height field.

B. Discrete Morse Theory

Simulation data is most often available as samples at the vertices of an underlying mesh. The data samples assign values to vertices, and a continuous function is recovered on the interior of cells through interpolation. Recently, however, new approaches embrace the mesh structure, and apply topological results directly to a purely discrete, combinatorial representation of the function. Discrete Morse theory, introduced by Forman [8], reproduces results from smooth Morse theory in this discrete domain.

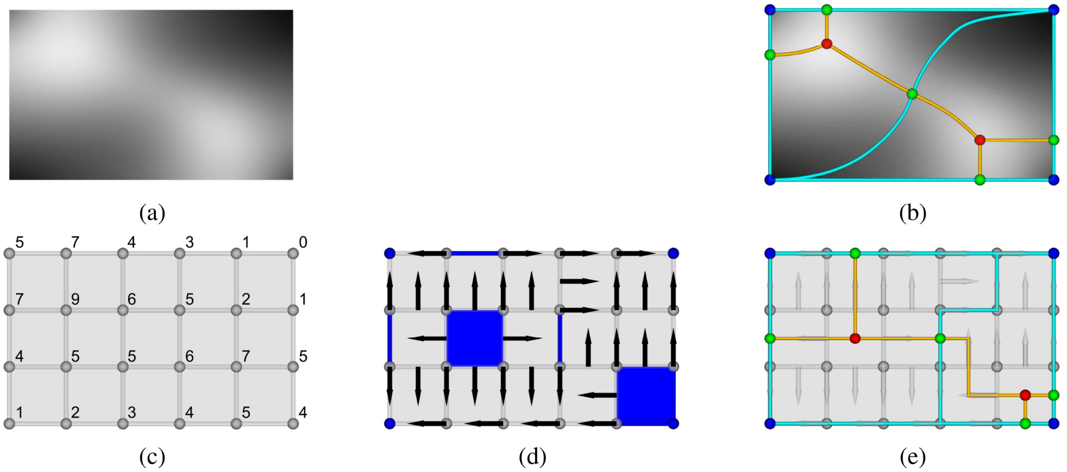
The following definitions are used to formalize notions about the implicit underlying mesh used in gridded data. A d-cell is a topological space that is homeomorphic to a d-ball $B^d = \{x \in \mathbb{R}^d : |x| \leq 1\}$. For example, a vertex is a 0-cell, an edge is a 1-cell, and a quad is 2-cell. For cells α and β, $\alpha < \beta$ means that α is a *face* of β and β is a *co-face* of α, *i.e.*, the vertices of α are a proper subset of the vertices of β. If $dim(\alpha) = dim(\beta) - 1$, we say α is a *facet* of β, and β is a *co-facet* of α. A cell α has dimension d, and we denote this as $\alpha^{(d)}$. We denote by K the regular cell complex that is a mesh representation of the underlying space $\mathbb{M}$.

The underlying principle of discrete Morse theory is a combinatorial representation of gradient flow. In a discrete vector field, flow is described by a pairing of cells: a $d+1$ cell, $\beta^{(d+1)}$, is paired uniquely with one of its faces, a d-cell, $\alpha^{(d)}$, and we say that $\beta^{(d+1)}$ is the head and $\alpha^{(d)}$ the tail of a *discrete vector*. Intuitively, this pairing indicates that flow passes through $\alpha^{(d)}$ and into $\beta^{(d+1)}$. V is a *discrete vector field* when each cell in K is paired in at most one discrete vector. The discrete equivalent of an integral line is a V-path:

$$\alpha_0^{(d)}, \beta_0^{(d+1)}, \alpha_1^{(d)}, \beta_1^{(d+1)}, \alpha_2^{(d)}, \ldots, \beta_r^{(d+1)}, \alpha_{r+1}^{(d)}$$

such that for each i = 0,..., r, the pair $\{\alpha_i^{(d)} < \beta_i^{(d+1)}\} \in V$, and $\{\beta_i^{(d+1)} > \alpha_{i+1}^{(d)} \neq \alpha_i^{(d)}\}$. When all V-paths are acyclic, the discrete vector field is a *discrete gradient vector field*. Note that unlike their smooth counterparts, discrete gradient arrows point in the direction of steepest descent.

Just as points with zero gradient in smooth Morse theory are critical, unpaired cells in a discrete gradient vector field are critical, with index of criticality equal to the dimension of the cell. Therefore in three dimensional regular grids, unpaired vertices are minima, unpaired edges are 1-saddles,



(a) (b)

(c) (d) (e)

Figure 2. A smooth scalar function is visualized in (a) using a grayscale ramp. The 1-skeleton of the two-dimensional MS complex is overlaid in (b). Minima are rendered as blue spheres, 1-saddles as green spheres, and maxima as red spheres. The minimum-1-saddle arcs of the complex are cyan tubes, and the 1-saddle-maximum arcs are gold tubes. The same function is coarsely samples at the vertices of a grid (c). A discrete gradient vector field (d) describes the flow behavior with gradient arrows. Unpaired cells are critical, and are rendered with blue. The 1-skeleton of the discrete complex (e) has nodes at the barycenters of critical cells, and arcs where V-paths connect them. While the locations of nodes can shift by 1/2 the width of a cell in either direction from the location of the "smooth" critical point, the connectivity of the complex remains unchanged.

unpaired quads are 2-saddles, and unpaired voxels are maxima. The nodes of the discrete MS complex are the critical cells of V, and the arcs are the V-paths connecting them. Figure 2(c-e) shows how this discrete interpretation can be used to recover topological information of an underlying function.

C. *Persistence-based Simplification*

A function f is simplified by repeated cancellation of pairs of critical points that differ in index by one. Forman [7] showed how a cancellation could be achieved in a discrete gradient field by reversing the gradient path between two critical cells. Gyulassy et al. [10] characterized the cancellation in terms of the 1-skeleton of the MS complex. The local change in the MS complex indicates a smoothing of the gradient vector field and hence of the function f. A cancellation removes two nodes and the arcs connecting them from the MS complex, and creates new arcs reconnecting nodes in their neighborhood. *Persistence* is a measure of the weight of a cancellation, and is computed as the absolute difference in function value of the canceled pair of nodes. Repeated application of the cancellation operation in order of persistence results in a hierarchy of MS complexes and a multi-resolution representation of the scalar function.

IV. Approach

Although our algorithm can be considered a two stage approach, first computing a local MS complex, and then merging the complexes together, it is implemented data-parallel; in other words, each of the steps listed is performed by every processing element in a distributed-memory supercomputer or cluster. Algorithm 1 lists and Figure 3 depicts these steps, which are described below in greater detail.

Algorithm 1 Overall algorithm

```
Decompose domain (section IV-A)
Read data blocks (section IV-B)
for all local blocks do
    Compute discrete gradient (section IV-C)
    Compute MS complex (section IV-D)
    Simplify MS complex (section IV-E)
end for
for number of rounds do
    Merge MS complex blocks (section IV-F)
end for
Write MS complex blocks (section IV-G)
```

A. *Domain Decomposition*

The data domain is a structured grid of regularly spaced hexahedral cells, with scalar values at the vertices. It is decomposed into a number of hexahedral blocks with a bisection algorithm that iteratively divides the longest remaining data dimension in half until the desired total number of blocks is attained. One layer of values is shared by two neighboring blocks. For example, if a block $B_{i,j,k}$ has size $X \times Y \times Z$, then $B_{i,j,k}[X-1][y][z] = B_{i+1,j,k}[0][y][z]$.

The total number of blocks may be greater than the number of processes, in which case blocks are assigned to processes in round-robin (block-cyclic) order. We designed

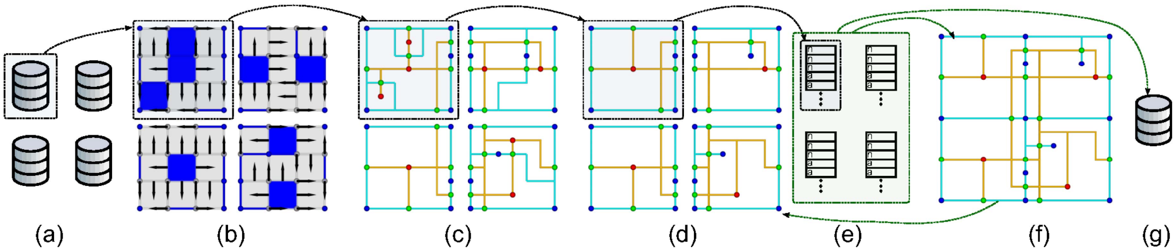


Figure 3. In this overview of our algorithm, the arrows and circled component indicate the sequence of operations performed by a single process:(a) parallel read, (b) local gradient computation, (c) local MS complex computation, (d) simplification, (e) preparing data structures for communication, (f) merging complexes, (g) parallel write. Each merge round repeats (d)-(f). We refer to steps (b) and (c) as the "compute" stage of the algorithm, and (d)-(f) as the "merge" stage.

the domain decomposition with flexibility in mind; depending on the distribution of nodes and arcs in the entire domain, multiple blocks per process may increase the chances that the computational load is better balanced across processes. In our tests, however, we found that computation scaled well using just one block per process and we did not further evaluate load balance.

B. Reading Data Blocks

Once a block decomposition and processor assignment of blocks has been determined, each process reads its blocks in parallel from storage. Currently, we support unsigned byte, single-precision floating point, and double-precision floating point data sets. We use an MPI-IO parallel read strategy whereby each process loops over its blocks, creates an MPI subarray data type for that block, sets an MPI file view using that datatype, and reads the block collectively with all other processes.

We considered more sophisticated approaches where blocks are first sorted and read in larger, contiguous chunks, before distributing to their final destinations. This is the method used by Kendall et al. [18], but in the data sizes that we have tested so far, the time to read the dataset was not a bottleneck.

C. Discrete Gradient Computation

We compute the discrete gradient vector field using the approach presented in [10], adapted to our parallel environment. In this algorithm, cells are sorted by increasing dimension, and then by increasing function value. Values are assigned to higher dimensional cells as the maximum of the values at the vertices. In this order, cells are paired in gradient arrows in the direction of steepest descent, if possible, otherwise are marked critical, and in either case are marked as assigned. A d-cell can be paired when it is the only unassigned facet of one of its unassigned co-facets. We use improved simulation of simplicity [11] to reduce the number of zero-persistence critical points found in flat regions.

The ability to glue MS complexes together in the merging stage of the algorithm (section IV-F) is predicated on the discrete gradients being identical on the shared boundary between them. To maintain consistency across neighboring blocks, we ensure that the discrete gradients computed on the shared block faces are identical by restricting the discrete gradient pairing. For a cell on the boundary of two or more blocks, we only consider for pairing other cells also on the boundary of those same blocks. In this manner, the shared boundary between two blocks will be assigned the same gradient arrows, independently from the interior of the blocks.

We use a refined grid to store the result of the gradient computation, where vertex i, j, k of the refined grid represents a d-cell of the implicit original grid, where $d = i\%2 + j\%2 + k\%2$. This refined grid is two times the length of the original structured grid in each dimension, and stores the discrete gradient pairing, criticality, and additional temporary values complactly in one byte per element.

D. MS Complex Computation

The finest-scale MS complex is computed by tracing V-paths in the discrete gradient field from critical cells. In a first pass through the gradient, all critical cells are added to the MS complex as nodes. V-paths are traced downwards from each node, and an arc is added to the MS complex for every path terminating at a critical cell. The list of cells in the V-path forms the geometric embedding of the arc, and is stored as a dynamically allocated array attached to a geometry object. The paths are guaranteed to terminate in the interior or boundary of a block, due to the restriction of the boundary gradient arrows. We use the data structure presented in [11] to store the 1-skeleton of the complex. In this data structure, nodes, arcs, and geometry objects are constant-sized elements stored in arrays. This structure is optimized for efficient simplification.

E. MS Complex Simplification

Persistence-based simplification is performed on the local complex to reduce the number of critical points and arcs. An input threshold value determines how far the simplification will proceed. In each cancellation, two critical points are destroyed, along with all the arcs connected to them, and several new arcs are created to reconnect the complex. The geometry of the new arcs is inherited from the deleted arcs, and a new geometry object is created that references the geometry objects that were merged in the cancellation. A thorough description of the data structure updates can be found in [14]. To ensure consistency across block boundaries, we do not consider for cancellation any arc having boundary nodes.

F. Merging MS Complex Blocks

So far, we described computing a strictly local MS complex for each block in the dataset. At this point, we could simply write each of these MS complexes in parallel to storage and terminate. While the resulting 1-skeleton is a valid MS complex and thus a solution to the problem, it may not be the best solution for later analysis, because global features spanning multiple blocks remain unresolved.

Without this step, depending on the dataset, the MS complex can be several times larger than necessary. Such bloating masks the original critical points and leads to large output file sizes, both of which can be alleviated by merging MS complex blocks prior to storing. In some cases, a complete merge down to a single output block is desired, while in others a partial merge down to a reduced number of output blocks, compared to the number of input blocks, is sufficient. Preparing for communication, performing the communication, and computing the merged complex are the main steps to consider.

1) Preparing for Communication: Preparing the local complexes for communication involves three steps: identifying the portions of the hierarchy that correspond to living elements at the desired simplification threshold; cleaning up the memory after computing the simplified MS complex; and translation of local indices to global ones.

The approach we use to simplify the MS complex computes a feature hierarchy. To reduce the memory footprint after simplification, we remove from memory all but the coarsest levels of the MS complex hierarchy.

We refer to the *address* of a cell as the location in the discrete gradient array at which it is stored. This address encodes the geometric location of the cell in the volume. When two complexes are merged (section IV-F3), the complexes are glued at nodes on their shared boundary. To detect that two nodes are co-located, we compare their address. However, when computed at a single block, the nodes and geometric locations of arcs of the MS complex are represented in local addresses. Let X^G, Y^G, Z^G be the length of the x, y, and z sides of the three-dimensional grid storing discrete gradients for the entire dataset, and X^L, Y^L, Z^L be the lengths of the sides of the block. Furthermore let S^x, S^y, S^z be the x, y, z offsets in the global gradient array of the first element of a block. Then the $(i,\ j,\ k)$th element of the block is at global address $a = (i + S^x) + (j + S^y) \times X^G + (k + S^z) \times X^G \times Y^G$. The local addresses are translated to global ones prior to the first round of communication.

2) Communication: Previous research demonstrated that HPC architectures such as Cray XT and IBM Blue Gene have ample bisection bandwidth to support multiple channels of concurrent communication. For example, Peterka et al. [22] designed a flexible image compositing algorithm with configurable radix values at each round. Our merge algorithm is inspired by this idea of specifying the number of rounds and radix of each round and is described in detail in [23]. It allows us to merge completely or partially, depending on the number of rounds and radix (communicating group size) per round.

We restrict merge groups to contain two, four, or eight members (radix-2, radix-4, or radix-8). However, instead of swapping subsets of information among all group members, as in image composition, we designate one member of the group as the "root," and the remaining group members send all of their information to the root of the group. The root performs the merge and retains the result for later steps, which can consist of more merge rounds or writing to storage. The other, non-root members of a group do not participate in these later steps. The number of resulting MS complex blocks after merging is the number of input blocks divided by the product of radices in each merge round.

3) Merge Computation: MS complexes are merged at a root of a group. After the data structures are communicated, the root has a list of independent complexes. The merge is performed by enlarging the root's MS complex, MS_{root}, by gluing each non-root complex, MS_i to the root. Our technique for computing the discrete gradient ensures that it is identical on the shared boundary between blocks B_{root} and B_i. Therefore, any critical cell in this shared boundary is a node in both MS_{root} and MS_i. These shared nodes anchor the gluing process.

To glue MS_{root} and MS_i, first, each node n_j in MS_i that is not on the shared boundary is added to MS_{root}. Next, each arc from MS_i is added to MS_{root} along with its corresponding geometry objects only if both its endpoints are not on the shared boundary. When both endpoints of an arc are on the shared boundary, the arc is guaranteed to exist in MS_{root} already. Once all other MS complexes are glued to MS_{root}, the boundary status of each node is updated according to the bounds of the merged blocks. The newly interior nodes become candidates for cancellation, and we use the procedure in section IV-E to create a new hierarchy.

G. Writing MS Complex Blocks

MS complex blocks are written to the output file collectively. The number and radices of merge rounds determine how many output blocks, if any, each process contributes to

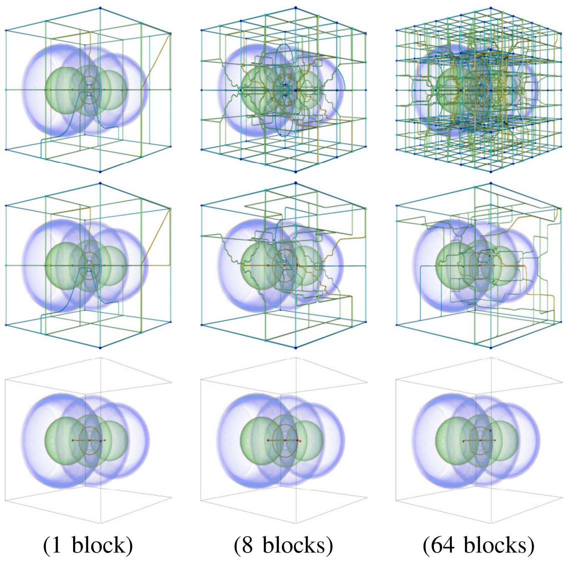


Figure 4. The full MS complex (top row) computed with various number of blocks. For all three experiments, we use the exact same simplification threshold and the same filters to extract relevant filters. After 1% persistence simplification, block boundary artifacts are removed (middle row). Important features are selected (bottom row) by choosing 2-saddle-maximum arcs and nodes with value greater than 14.5. Although the geometric embedding of features can shift by the width of a cell due to discretization, we recover large-scale features spanning several blocks.

the collective write. Those processes with no output blocks participate in the collective operation by issuing a ”null” write consisting of zero bytes. The output file is a binary collection of all of the output blocks, followed by a footer that provides an index to the MS complexes contained in the file. The file format is documented in detail in [23].

V. Properties of the Parallel MS Complex

The main challenges in parallelizing the computation of the MS complex are lack of locality and data-dependency of the output. The MS complex is, by nature, a global structure, that encodes both large- and small-scale features of a scalar function. Furthermore, the size of the complex is primarily determined by the number of features in the input function. In the following, we discuss the stability of nodes and arcs of the MS complex computed in parallel with respect to the MS complex computed in serial, that is, we answer the question: which arcs of the complex computed in serial are guaranteed to be present in the parallel computation? Furthermore, we discuss the expected size of the MS complex, a result that motivates per-block MS complex simplification to manage the size of the output.

A. Stability

When the complex is computed with a varying number of blocks, we observe that while certain nodes and arcs of the complex are preserved, others are not. Figure 4 illustrates this phenomenon for a byte-valued scalar function representing the spatial probability density of a hydrogen atom residing in a strong magnetic field. Our approach to computing the discrete gradient (section IV-C) restricts the gradient pairing of cells on the boundary of a block. This introduces spurious critical cells, corresponding to critical points of the restriction of the scalar function to the two-dimensional boundary of a block. These show up as nodes in the MS complex having zero persistence; they are, however, necessary “handles” for gluing two neighboring complexes together (section IV-F). It is the cancellation of these boundary artifacts that directly connects important critical points in the interiors of neighboring blocks using arcs of the MS complex, therefore resolving global features.

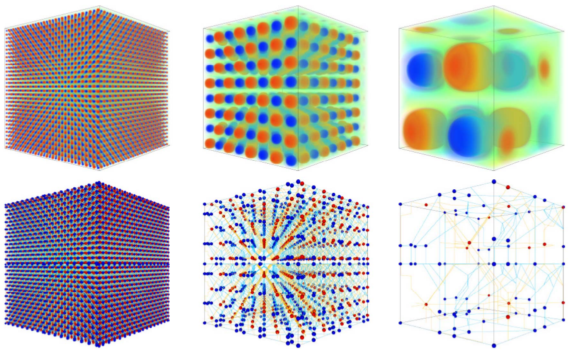
Even after simplification, however, the MS complex computed in parallel may differ from the MS complex computed in serial. Upon closer inspection, differences originate at locations where the gradients do not have a unique direction of steepest descent, for example, in flat regions. Note that this level of variability is present even in different serial implementations, and any robust analysis only accounts for stable critical points.

More formally, nodes are stable under blocking strategy when the Hessian of the underlying scalar function at the node location is non-singular. In this case, the algorithm that constructs the discrete gradient guarantees a nearby critical cell. The existence of a stable critical point is an entirely local decision, which is the reason that the main features are preserved in the parallel implementation. Let a be an arc of the complex connecting a node of index $d-1$ to a node of index d, and P be any plane that transversally intersects a. The geometric embedding of a is stable when the point of intersection with P is a critical point of index $d-1$ of the scalar function restricted to P.

The volume “outside” the hydrogen atom in Figure 4 (middle row) has constant value, and hence critical points and the geometric embedding of arcs connected to them are unstable and can shift dramatically. However, important features are preserved where the stability conditions are met. For instance, in Figure 4 (bottom row), both the parallel and serial computation of the MS complex reveal three stable maxima connected by stable arcs in a line, and the loop representing the toroidal region. Note that, although the arc representing this loop is stable, the location of the maximum is not, since the function has a plateau along the arc.

B. MS Complex Size in Practice

The memory resources needed to store the MS complex depend on the topological complexity of the scalar function, and the geometric size of the embedded arcs. While there may be $O(n)$ critical points in a sampled function with n samples, and $O(n^2)$ arcs connecting them, the expected value is much lower for practical data. The MS complex itself is a mesh structure embedded in the domain, with nodes



(32 features/side) (256 points/side) (8 features/side) (256 points/side) (2 features/side) (256 points/side)

Figure 5. An artificially generated 256^3 dataset is volume rendered (top row), and the corresponding complex is illustrated (bottom row) for varying feature counts.

at critical points and arcs connecting them. The expected number of arcs for a mesh is linear in the number of nodes. While severely noisy data may in fact have O(n) critical points, we argue that features from simulation data are much more widely spaced, else the simulation itself would lack numerical stability. In practice, we see that the number of important features, k, of a dataset will be several orders of magnitude less than the number of sample points n. In our examples in section VI-D, the number of significant features was six orders of magnitude fewer than the number of sample locations for the combustion example, and three for the mixing fluids.

In our results, we found that the cost of storing the geometric embedding of the arcs was directly proportional to the length of one side of the dataset. The arc geometries are one-dimensional objects embedded in a three dimensions, therefore, for n samples, the cost of storing the geometry of one arc was O($n^{1/3}$). Finally, we can estimate the storage requirements of the MS complex with $k \times c + k \times n^{1/3}$, where k is the expected number of features, and c is a constant that represents the cost of storing one node or one arc. Figure 6 illustrates this behavior for an artificially generated dataset.

VI. Performance Results

A. Test Environment

The IBM Blue Gene/P *Intrepid* is a 557-teraflop supercomputer operated by the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory. It consists of 40 racks, each rack containing 1,024 nodes, for a total of 40,960 nodes. Each node has four cores, for a grand total of 163,840 cores. The nodes are connected in a 3D torus topology. Our tests are conducted in *smp* mode, that is, one process per node. This allows each process 2GB of memory, which we found is necessary for some of our larger datasets.

B. Data Size and Complexity Study

To better characterize our algorithm's dependence on factors such as process count, data size, and data complexity, we conducted the following study. We generated synthetic datasets of various size and complexity by computing a sinusoidal scalar field. The data are 3D 32-bit floating point values, on a cubic grid of a given number of points per side of the cube. In other words, 512 points per side represents a $512 \times 512 \times 512$ volume. The complexity, or number of features per side, is how many times the sine function has a ± 1 value along the length of one side of the volume. Figure 5 shows examples of three levels of complexity.

Figure 6 shows the effect of process count, data size, and data complexity on compute time, merge time, and output MS complex size. All plots are in log-log scale. Several interesting correlations are evident in the discussion below.

We first examine the compute time in the upper row of Figure 6. This is the time to generate a discrete gradient field from the dataset and compute the local MS complex on it. It scales linearly with process count, and compute time increases with data size, as the individual lines in each panel show. In fact, the compute time shows a weak scaling efficiency of 1; the compute time only depends on the size of the blocks. As we scan horizontally across the row of panels, however, we note that compute time is not related to topological complexity.

Next we consider the merge time in the center row of Figure 6. This is the time to merge the previously-computed MS complexes down to a smaller number of output blocks. We performed two rounds of radix-8 merging for this test. Similar to compute time, merging scales linearly with process count. Unlike compute time, however, the lines within a panel reveal that merge time is unaffected by data size, because the individual lines within each panel coincide. Instead, it is a function of complexity, as a horizontal scan across the row of panels shows.

The output size is shown in the lower row of Figure 6. Output size increases slowly with process count, because we performed a constant number of merges. Starting with more processes, therefore, results in a greater number of output blocks, and hence unresolved boundary artifacts that add to the size. This accounts for the slope within each panel. When the topological complexity is high (right panel), the output size is dominated by the nodes and arcs of the MS complex, and the overhead of the boundary artifacts is less noticeable. When the feature complexity is low (left panel), the output size is dominated by the geometric embedding of the arcs, accounting for the roughly factor of two increase as the number of points per side is doubled.

C. Selecting the Merge Strategy

Scientists will have some leeway in selecting the degree of merging to execute when using our algorithm, even given constraints of file size, memory size, and run time. In this

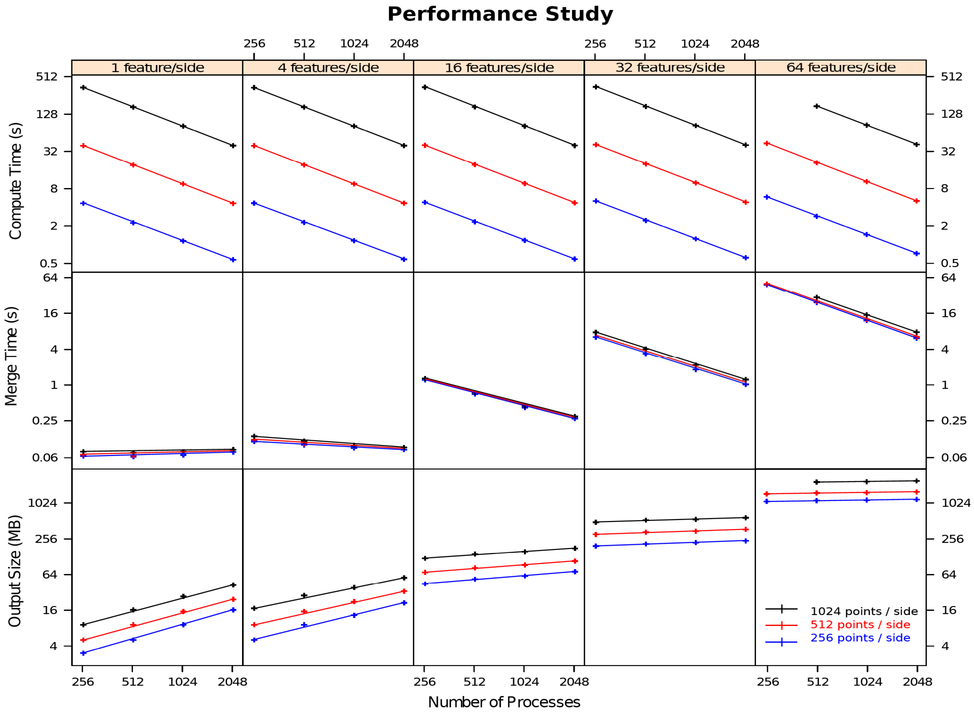


Figure 6. Compute time, merge time, and output size as a function of number of processes, data size, and data complexity are plotted in log-log scale.

Table I
Cost of Merging 2048 Blocks

Number of Rounds	Round Radices	Total Merge Time (s)	Final Round Merge Time (s)
1	4	0.598	0.598
2	4 8	1.310	0.712
3	4 8 8	2.635	1.325
4	4 8 8 8	9.843	7.208

Table II
Merge Strategies for Full Merge of 256 Blocks

Number of Rounds	Round Radices	Compute + Merge Time (s)
3	4 8 8	144.040
3	8 8 4	144.528
4	4 4 2 8	144.955
4	4 4 4 4	145.012
8	2 2 2 2 2 2 2 2	149.174

section, we lend some guidance in making those decisions based on two studies of the number of rounds and the radix in each merge round.

1) Cost of Each Merge Round: The cost of merging is one factor that influences to what extent input blocks are merged into a smaller number of output blocks. Table I contains an example of merging 2048 input blocks across 2048 processes. In this example, a full merge consisted of four rounds of radices [4, 8, 8, 8], appearing in the last row of the table. The first row of the table assumes that we perform only one round of radix-4, and the following rows add one more round each time, so that the second round is two rounds of radices [4, 8], and so on.

The third column is the total time required to perform this merge, and the fourth column is the time required by the last round, so that when read from top to bottom, the fourth column shows the individual round times of the first, second, third, and fourth rounds, respectively. As merging progresses, it becomes more expensive, because MS complex blocks grow larger, take longer to communicate, and gravitate toward fewer processes. This is one reason why the performance and scalability in Figure 9 diminishes when scaling to high process counts and performing full merging.

2) Merge Strategy: Once a number of output blocks is determined, it remains to decide how many rounds to use and what the radix of each round should be to get there. We call this the merge strategy, and an example appears in Table II for performing a full merge from 256 input blocks to one output block. This result is typical of our results, and from it we can generalize the following merge strategy.

A smaller number of rounds with higher radices is desired. We suggest using radix-8 whenever possible. When radix-8 cannot be used for a round because the ratio of input to

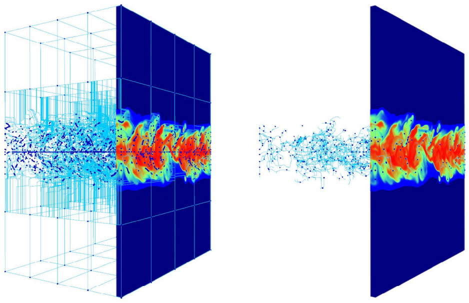


Figure 7. The MS complex that results from a partial merge (left) and a full merge (right) of the jet mixture fraction data, with a volume rendering of a slice of the original scalar field.

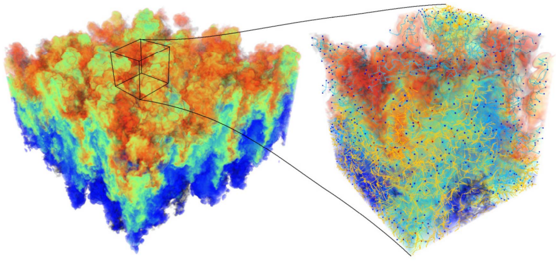


Figure 8. Volume rendering and MS complex for Rayleigh-Taylor mixing dataset.

output blocks is not divisible by eight, the remaining smaller radices are slightly better in early rounds rather than later. This is evident in comparing the first and second rows of Table II. As the previous section showed, rounds become more costly as they progress, so it is better to optimize later rounds to radix-8 whenever possible.

D. Run Time and Strong Scaling

1) Jet Mixture Fraction Dataset: The JET simulation is a temporally-evolving turbulent CO/H_2 jet flame undergoing extinction and re-ignition at different Reynolds numbers [16]. In this simulation, structures called *dissipation elements* are correlated to flame extinction, and are centered around minima of mixture fraction. We find important minima by computing and simplifying the MS complex. We tested the performance of our algorithm on a single time-step. The data size consists of 32-bit floating-point scalars on a $768 \times 896 \times 512$ regular grid, 1.4 GB in size. Our domain decomposition consisted of one block per process.

Figure 7 shows the result of a partial merge and a full merge of the jet mixture fraction MS complex. The total run time, component times, and scalability are shown in Figure 9. The timing results represent a full merge down to one output block, using radix-8 merging whenever possible, as indicated by our merging guidelines presented earlier. For 8192 processes, for example, there were 8192 input blocks that were merged in five rounds with radices of [2, 8, 8, 8, 8]. The output file size of the fully merged MS complex is approximately 26 MB.

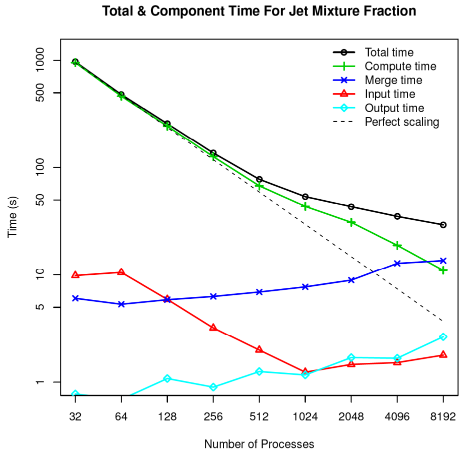


Figure 9. Overall time and four components: read data, compute, merge, and write results, plotted in log-log scale. At small numbers of processes, time is dominated by computing, and at higher numbers of processes by merging.

Figure 9 reveals that most of the time is spent in computing the discrete gradient field and MS complex, and merging blocks. The total run time is 970 s. at 32 processes and 29 s. at 8192 process, for an end-to-end strong scaling efficiency of 13%. Efficiency is computed as the ratio of the factor decrease in time divided by the factor increase in number of processes, using 32 processes as the base efficiency of 1.0 in this case.

The efficiency at 2048 processes is 35%, and there are two reasons for the relatively flat scaling beyond 2048 processes. First, the problem size is not large enough to warrant more processes than that. Second, Figure 9 shows the rapidly increasing merge time beyond 2048 processes. While we could have performed less merging and thereby improved our scalability, the object of this test is to evaluate the worst-case performance for this dataset. The next benchmark is a more realistic scenario and shows that our algorithm is more efficient at higher process counts, with larger data, and a partial degree of merging.

2) Rayleigh-Taylor Mixing Dataset: Our largest benchmark comes from a simulation of mixing fluids in a Rayleigh-Taylor instability [4]. When a heavy fluid is placed on top of a lighter one, vertical perturbations in the interface create a structure of rising bubbles and falling spikes. The scalar field we study is density, and here the 1-skeleton of the MS complex can detect when isolated bits of one fluid penetrate the other. Figure 8 shows a volume rendering of the data as well as a cut-away view of the topology extracted at a late time-step.

Figure 10 shows the performance of our algorithm on this dataset. The data size consists of 32-bit floating-point scalars

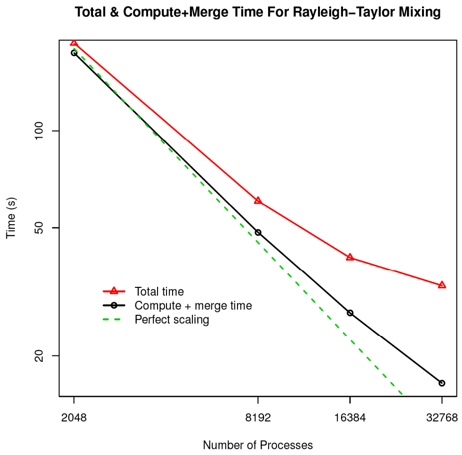


Figure 10. Overall time and compute+merge time, plotted in log-log scale. The strong scaling efficiency of the compute+merge time is 66%, and it is 35% for the overall end-to-end time.

on a $1152 \times 1152 \times 1152$ regular grid, 5.7 GB in size. We ran with one block per process and performed a partial merge of two rounds of radix-8 merging.

This test demonstrates scalability of our algorithm to 32,768 nodes out of a total machine size of 40,960 nodes. We did not test on the entire machine because a reservation was not available at the time. The upper curve in Figure 10 shows 35% efficiency in overall time to 32,768 processes. The output size is approximately 4 GB. Excluding the I/O time and measuring only the time to compute and merge the MS complexes results in strong scaling of 66% to 32,768 processes.

VII. Summary

A. Conclusions

We designed and implemented, for the first time, a scalable algorithm to compute the MS complex of a scalar field in parallel on a distributed-memory architecture. The algorithm consists of decomposing the domain, reading the dataset from storage, locally computing the discrete gradient and MS complex, simplifying the MS complex, merging to a smaller number of output blocks, and writing the MS complex blocks to storage.

To better understand the correlation between data size, data complexity, process count, compute time, merge time, and output size, we conducted a study using synthetic data of varying size and complexity. We found that compute time decreases linearly with process count and that it increases with data size, independent of complexity. Merge time also scales linearly at small process counts, but unlike compute time, it is independent of data size and is linear in the data complexity. The output size is primarily governed by data complexity.

We also presented heuristics for choosing a merge strategy. The cost of merging increases with each round, so deciding on the degree of merging ought to take this into consideration. For a given number of output blocks, radix-8 or the highest radix possible should be selected in order to minimize the number of rounds. When the optimal radix cannot be used, smaller radices should be used in earlier rounds rather than later rounds.

We benchmarked performance and scalability on two datasets from combustion and physics science domains out to 32,768 nodes, or 80% of the Argonne Leadership Facility's Blue Gene/P machine. The cost of merging and of output I/O were the primary limitations to scalability at high process counts, although we were able to achieve 35% strong scaling in overall end-to-end performance.

B. Future Work

There are several directions that we plan to explore as we continue this work. We will continue to improve output I/O performance. We have also ported our implementation to the Jaguar XT5 system at the Oak Ridge Leadership Computing Facility, and we are testing our benchmarks there as well. From there, we plan to embed our algorithm into the S3D combustion code and generate parallel MS complexes in situ with combustion simulations. In the longer term, we plan to experiment with global persistence simplification in the context of our parallel structure. We anticipate that this can be performed using a series of nearest-neighbor communication operations. This will allow us to further reduce the size of the output data and to reduce the complexity of the resulting MS complex.

VIII. Acknowledgment

We gratefully acknowledge the use of the resources of the Argonne Leadership Computing Facility at Argonne National Laboratory. This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357. Work is also supported by DOE with agreement No. DE-FC02-06ER25777. We thank Jacqueline H. Chen and Ray Grout for the JET data.

References

[1] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. A topological hierarchy for functions on triangulated surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):385–396, 2004.

[2] P.-T. Bremer, G. Weber, V. Pascucci, M. Day, and J. Bell. Analyzing and tracking burning structures in lean premixed hydrogen flames. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):248–260, 2010.

[3] C. Chevalier and F. Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel Comput.*, 34:318–331, July 2008.

[4] A. W. COOK, W. CABOT, and P. L. MILLER. The mixing transition in Rayleigh-Taylor instability. *Journal of Fluid Mechanics*, 511:333–362, 2004.

[5] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale complexes for piecewise linear 3-manifolds. In *Proc. 19th Ann. Sympos. Comput. Geom.*, pages 361–370, 2003.

[6] 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.

[7] R. Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134(1):90–145, 1998.

[8] R. Forman. A user's guide to discrete Morse theory. In *Proc. of the 2001 Internat. Conf. on Formal Power Series and Algebraic Combinatorics, A special volume of Advances in Applied Mathematics*, page 48, 2001.

[9] D. Gregor and A. Lumsdaine. The parallel bgl: A generic library for distributed graph computations. 2005.

[10] 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.

[11] A. Gyulassy, P.-T. Bremer, B. Hamann, and V. Pascucci. Practical considerations in Morse-Smale complex computation. In V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, editors, *Topological Methods in Data Analysis and Visualization*, pages 535–542. Springer-Verlag, 2010.

[12] 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.

[13] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. Topology-based simplification for feature extraction from 3d scalar fields. In *Proc. IEEE Conf. Visualization*, pages 535–542, 2005.

[14] A. Gyulassy, V. Natarajan, V. Pascucci, P.-T. Bremer, and B. Hamann. A topological approach to simplification of three-dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):474–484, 2006.

[15] A. Gyulassy, V. Natarajan, V. Pascucci, and B. Hamann. Efficient computation of Morse-Smale complexes for three-dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1440–1447, 2007.

[16] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. *Journal of Physics: Conference Series*, 16(1):65, 2005.

[17] D. Hennessey, D. Brooks, A. Fridman, and D. Breen. A simplification algorithm for visualizing the structure of complex graphs. In *Proceedings of the 2008 12th International Conference Information Visualisation*, pages 616–625, Washington, DC, USA, 2008. IEEE Computer Society.

[18] W. Kendall, J. Huang, T. Peterka, R. Latham, and R. Ross. Visualization viewpoint: Towards a general i/o layer for parallel visualization applications. *To appear in IEEE Computer Graphics and Applications*, 31(6), 2011.

[19] H. King, K. Knudson, and N. Mramor. Generating discrete Morse functions from point data. *Experimental Mathematics*, 14(4):435–444, 2005.

[20] D. Laney, P.-T. Bremer, A. Mascarenhas, P. Miller, and V. Pascucci. Understanding the structure of the turbulent mixing layer in hydrodynamic instabilities. *IEEE Trans. Visualization and Computer Graphics (TVCG) / Proc.of IEEE Visualization*, 12(5):1052–1060, 2006.

[21] T. Lewiner, H. Lopes, and G. Tavares. Applications of Forman's discrete Morse theory to topology visualization and mesh compression. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):499–508, 2004.

[22] T. Peterka, D. Goodell, R. Ross, H.-W. Shen, and R. Thakur. A configurable algorithm for parallel image-compositing applications. In *Proceedings of SC 09*, Portland OR, 2009.

[23] T. Peterka, R. Ross, W. Kendall, A. Gyulassy, V. Pascucci, and H.-W. Shen. Scalable parallel building blocks for custom data analysis. In *Submitted to Proceedings of LDAV'11*, Providence, RI, 2011.

[24] T. Peterka, R. Ross, B. Nouanesengsey, T.-Y. Lee, H.-W. Shen, W. Kendall, and J. Huang. A study of parallel particle tracing for steady-state and time-varying flow fields. In *To appear in Proceedings of IPDPS 11*, Anchorage AK, 2011.

[25] T. Peterka, H. Yu, R. Ross, K.-L. Ma, and R. Latham. End-to-end study of parallel volume rendering on the ibm blue gene/p. In *Proceedings of ICPP 09*, Vienna, Austria, 2009.

[26] D. Pugmire, H. Childs, C. Garth, S. Ahern, and G. H. Weber. Scalable computation of streamlines on very large datasets. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, 2009. ACM.

[27] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[28] S. Smale. Generalized Poincaré's conjecture in dimensions greater than four. *Ann. of Math.*, 74:391–406, 1961.

[29] S. Smale. On gradient dynamical systems. *Ann. of Math.*, 74:199–206, 1961.

[30] H. Yu, C. Wang, and K.-L. Ma. Massively parallel volume rendering using 2-3 swap image compositing. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11, Piscataway, NJ, USA, 2008. IEEE Press.