SAND2013-3589C

In-situ Analysis of Large Scale Combustion Simulations

Peer-Timo Bremer
Lawrence Livermore National Laboratory
SCI Institute
University of Utah

Aaditya G. Landge,
Attila Gyulassy,
Valerio Pascucci
SCI Institute,
University of Utah

Janine C. Bennett,
Hemanth Kolla,
Jacqueline Chen
Sandia National Laboratories

ABSTRACT

The ever increasing amount of data generated by scientific simulations coupled with system I/O constraints are resulting in a growing interest in in-situ analysis techniques. Such approaches analyze data with minimal overhead while distilling the quantities of interest, potentially reducing the output size by orders of magnitude. Of particular interest are techniques that produce reduced data representations while maintaining the ability to re-define, extract, and study features in a post-process to obtain novel scientific insights. In this work, we present a fast and scalable in-situ topological analysis technique that describes the evolution of level sets of a function. Our approach computes both merge trees and a compact representation of the geometry of the associated features which can be permanently stored with low overhead. We deploy our in-situ analysis technique with S3D, a massively parallel turbulent combustion simulation, and measure run-time performance on leadership class supercomputers. Finally, we demonstrate that our in-situ approach allows much higher analysis frequency than previous post-process approaches with negligible impact on the simulation.

1. INTRODUCTION

With the continued increase in available computing power, scientists are simulating ever more complex phenomena at higher spatial and temporal resolutions. Unfortunately, file I/O rates are increasing at a significantly slower pace, and therefore the gap between the amount of data created and the amount that can be permanently stored continues to grow. In practice, this means that relatively fewer time steps of a simulation are saved. However, currently the majority of data analysis is performed in post-process, using only these saved snapshots, and reducing their number quickly compromises our ability to reliably analyze the data. State of the art simulations are already reaching the point at which snapshots are stored too infrequently to accurately track fast moving events, increasing the likelihood that rare but potentially important phenomena are lost between snapshots.

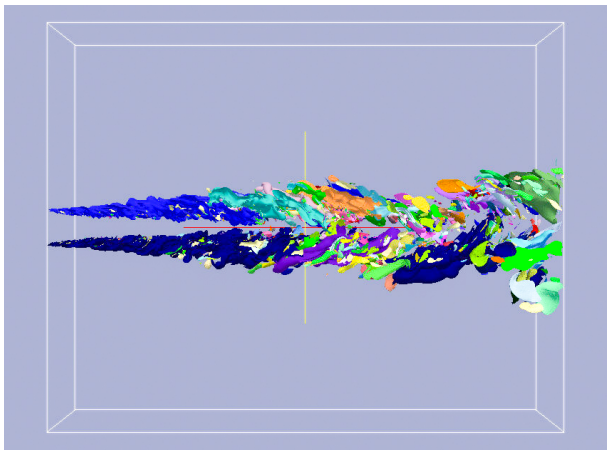


Figure 1: The segmentation of the Lifted Flame data set as extracted by our algorithm at a threshold value of 100

To address this challenge the data analysis paradigm must shift from an off-line, post-processing model to an on-the-fly, in-situ approach. Since the results of an analysis are typically several orders of magnitude smaller than the raw data, in-situ analysis can circumvent the file I/O issue without compromising scientific impact. However, this shift in paradigm introduces challenges and constraints on the data analysis algorithms that must be addressed prior to their successful deployment in-situ. First, since data movement is anticipated to be a primary bottleneck, particularly as we move to exascale computing environments, analysis algorithms must operate on the same or a very similar data decomposition as the simulation. However, simulation and data analysis algorithms tend to have very different characteristics (e.g. floating point operations, memory access patterns, and communication patterns), and so a data decomposition that is optimal for the simulation is typically too fine and the core count too large for analysis algorithms to perform efficiently. Second, processing times must be small enough to not impact the overall execution time unduly, even when multiple different analyses are performed. Third, many analysis algorithms require input parameters, whose values may dramatically impact the overall quality of the resulting analysis. However, in an in-situ setting, these parameters are typically not yet known, and therefore a reasonable range of possibilities must either be sampled or otherwise stored. Fourth and finally, the analysis results

must be small enough to allow for high frequency storage to disk.

While there exist a number of efforts to develop in-situ analysis capabilities [34, 29, 31, 14, 4], there are few complex analysis algorithms beyond computing global or local descriptive statistics or subsetting the data fulfilling all the requirements listed above. In particular, sampling one or multiple parameter ranges is too expensive in both time and space to be viable, yet few algorithms are capable of the kind of meta-analysis required to simultaneously process and efficiently encode results for ranges of parameter values. Here, we present a new parallel, in-situ framework to compute segmented merge trees of scientific data. Segmented merge trees efficiently encode threshold-based features such as extinction regions [22], eddies [32], or burning cells [7] for a large range of parameters and have proven highly effective for large scale analysis and visualization [3]. Previous approaches have been restricted to off-line serial computation [7] or to smaller scale parallel efforts [25, 24] that ignore the segmentation. Instead, we present an approach that is faster than previous techniques, applicable to orders of magnitude more cores, and flexible enough to adapt to various hardware environments and simulations. Using a combination of divide-and-conquer and streaming techniques, the algorithm shows good scaling behavior for sufficiently large data and remains fast enough at extreme scales to be viable in an in-situ scenario. Using two examples from the S3D [11] combustion simulation, we show that at current scales our algorithm can analyze and store the corresponding results at ten times the temporal frequency of the original simulation using only fractions of a percent of the overall simulation time. Furthermore, using only one of the cores on each compute node, we demonstrate that, even at significantly larger core counts, this fraction is not expected to increase. Finally, the algorithm is simple to implement and highly flexible making it an ideal choice for run time optimizations as well as alternate computing scenarios such as staging areas [31, 12, 23]. In summary, this paper presents a uniquely sophisticated in-situ data analysis algorithm and demonstrates its viability using a state of the art simulation code.

2. RELATED WORK

As the performance gap between compute and I/O capabilities increases, the need for concurrent workflows for data-intensive analysis is growing. Recently, several algorithms have been presented that bypass the I/O barrier by operating directly on in-memory simulation data in-situ, or use asynchronous data movement to compute in separate dedicated resources in-transit. While successfully deployed in-situ algorithms have largely focused on visualization techniques [30, 33, 34], recent efforts are exposing additional analysis capabilities, [4, 14, 19]. Initial in-transit frameworks focused largely on mitigating I/O costs [35, 2, 12, 1], however more recent efforts have begun to integrate analysis prior to dumping data to disk [31].

Topology-based representations of scalar functions provide a discrete structure upon which to formulate queries for robust feature extraction. Reeb graphs [28] and their variants, contour trees [8] and merge trees, encapsulate level set behavior, and Morse- and Morse-Smale complexes [13] capture gradient flow based features. In each case, a continuous function is converted to a representation that is efficiently stored using an annotated graph data structures. This discrete representation can be orders of magnitude smaller than the original data, yet maintains enough information, for example, to enable the exploration of features at multiple threshold values. Furthermore, topological simplification is used to represent features hierarchically, classifying their importance based on a user selected metric, possibly persistence, volume, hypervolume, or relevance [13, 10, 22]. Topology-based techniques have proven useful for sophisticated analysis in computational sciences, for example, identifying extinction regions in non-premixed turbulent combustions [22], analyzing lean pre-mixed hydrogen flames [7], detecting of bubbles in turbulent mixing [20], and extracting of the core structure of a porous solid [16].

While many algorithms have been proposed to compute topological representations, the lack of inherent spatial locality of features has led to few successful distributed implementations. Most algorithms fall under the categories of fully in-memory [8, 13, 5, 17], streaming out-of-core [26, 7], or small-scale parallel [25, 15]. Fundamentally, the bottleneck to scalability of topology-based algorithms for distributed algorithms is the discrepancy between the local view of the data, and the global span of features. Recently, an algorithm for computing Morse-Smale complexes was shown to scale to large process counts, but only by prematurely terminating the analysis, with global features resolved only in a post-process [18]. This kind of approach does not suffice when the full representation is needed at run-time, for example when the analysis is performed in-situ. Morozov and Weber [24] presented an approach for computing a local-global representation of merge trees in a distributed setting that uses communication to position the merge tree of a local process within a sparsified tree of the global domain. Their algorithm is designed as a query system to allow on the fly analysis while the tree is kept in (distributed) memory. Instead, our approach aims at in-situ computation where the specific query is not known and any result must be computed and stored as quickly as possible. As a result our representation must support changing the thresholds and queries in a post-process, ideally on a commodity desktop machines. Our algorithm, in addition to being significantly faster, also computes and stores a segmentation of the domain corresponding to the global merge tree to enable later interactive exploration, shape analysis, feature tracking, and visualization.

3. BACKGROUND

Let f be a real-valued map $f : M \to R$ defined on a compact manifold. The region of the domain with value in f greater than $c \in R$ is called the *super-level set* of c. The merge tree encodes the evolution of connected components of the super-level set as the value c is swept from ∞ to $-\infty$. Local maxima in f create new components, while saddles can merge existing components. The merge tree is composed of *nodes* and *arcs*. The nodes represent the critical points that create, merge, or destroy super-level set components. An arc exists between two nodes if the super-level set component created by the upper node is merged or destroyed by the lower node. Figure 2 shows the merge tree for a simple two-dimensional example.

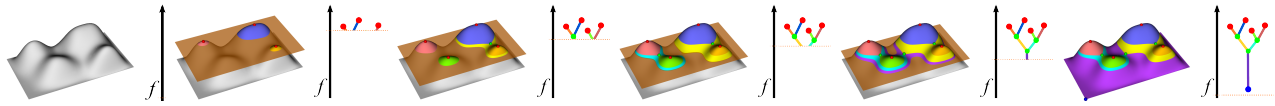


Figure 2: A merge tree tracks the evolution of super-level sets. In a sweep from ∞ to $-\infty$ each maximum (red sphere) creates a new component, that is merged by saddles (green sphere). The merge tree ends at the global minimum (blue sphere).

Merge trees describe the topological changes in the super-level sets of a function. Additionally, the geometric descriptions of the super-level sets are often needed for analysis, for example, to determine volumes, shapes, track features, or for visualization. The *segmentation* of the domain according to a merge tree is a partitioning where two points belong to the same region if and only if the contours passing through them appear on the same arc of the merge tree. Storing the segmentation along with a merge tree enables the geometric reconstruction of super-level sets during a post-process. Furthermore, access to the segmentation at run-time allows the precomputation of various conditional feature based statistics such as, for instance, average temperatures per feature [3]. Therefore, while the merge tree itself contains only information about the number of features at a given threshold, combining it with the corresponding segmentation creates a powerful and highly flexible analysis framework.

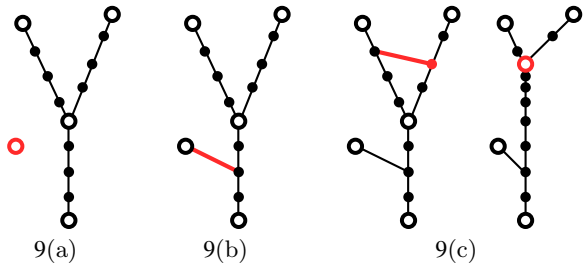


Figure 3: The core of the merge tree algorithm comprises three operations: adding a vertex, adding and edge, and gluing arcs. In (a), the addition of a vertex adds a component. In (b), the addition of an edge connects two components. In (c), the addition of an edge connects vertices in different subtrees, and the associated arcs are glued together.

4. TECHNIQUE

The core of the algorithm is inspired by the serial streaming approach of Bremer et al. [6], re-formulated and extended for large-scale parallel computation. The original algorithm takes as input a set of input mesh vertices and edges and produces a merge tree as a set of nodes and arcs. The fundamental idea is to stream through all edges and vertices of the input mesh while maintaining the correct merge tree of all elements seen so far. The only restriction is that the vertices of an edge must be processed before the edge itself. Ignoring some optimizations, the algorithm is very simple and as shown in Figure 3 consists of only three different operations. One can add a vertex 3(a), add an edge 3(b), and glue arcs 3(c). Each time a vertex is added it creates a new component. Each edge either connects two components (Figure 3(b)) or connects vertices within the same component. In the latter case the edge can either connect two vertices that are part of the same sub-tree (i.e. the higher vertex is in the subtree rooted at the lower vertex) or vertices in different subtrees which creates a loop in the tree (Figure 3(c)). The loop is a (temporary) invalid configuration and is corrected by gluing the paths to their earliest common root. The gluing is achieved by merge-sorting vertices on each path according to their function values. Once all vertices and edges are processed we have constructed an *augmented merge trees* (AMT) containing all vertices of the original mesh. In this AMT all non-valence two vertices correspond to critical points which form the nodes of the final merge tree. The arcs of the final tree represent the corresponding chain of valence two vertices.

However, as described, the algorithm is slow; Bremer et al. [6] discusses several improvements most notably the removal of valence two vertices from the intermediate tree and an index structure to accelerate the local searches involved in the adding of an edge and the gluing of arcs. The details are beyond the scope of this paper and we refer the reader to the original work for a more complete description. However, even with the improvements, the serial streaming algorithm has problems once the data kept in memory becomes too large, as the local searches become expensive. To address these issues, we divide the algorithm into several pieces, each exploiting or correcting a particular characteristic of the original algorithm. The primary insight is that the algorithm trivially supports a divide-and-conquer strategy. Given two merge trees from two pieces of a domain that share boundary vertices, we can process the list of their nodes and arcs in a manner that is analogous to the processing of the original vertices and edges to produce a joined merge tree. The only caveat is that sufficient information must be preserved along the shared boundary to allow a correct merge.

This is trivial in case of an AMT since all boundary vertices are part of both trees ensuring the correct connections. The merge trees, however, have removed all valence two nodes. In particular, there exist cases where a critical node u in tree $T1$ appears as regular vertex (valence two) in tree $T2$ and thus might be removed (see Figure 4). When trying to combine both trees we cannot directly find u in $T2$ and thus may construct an incorrect joined tree. Notice, however, that we do not need to know u's actual position in $T2$ as long as we know the subtree of which u would be the root. Starting from any node in this subtree we can walk downward until we find a node with a smaller function value than u which must be its direct lower neighbor. Therefore, for all boundary components we must preserve any vertex that is a local maximum with respect to this boundary com-

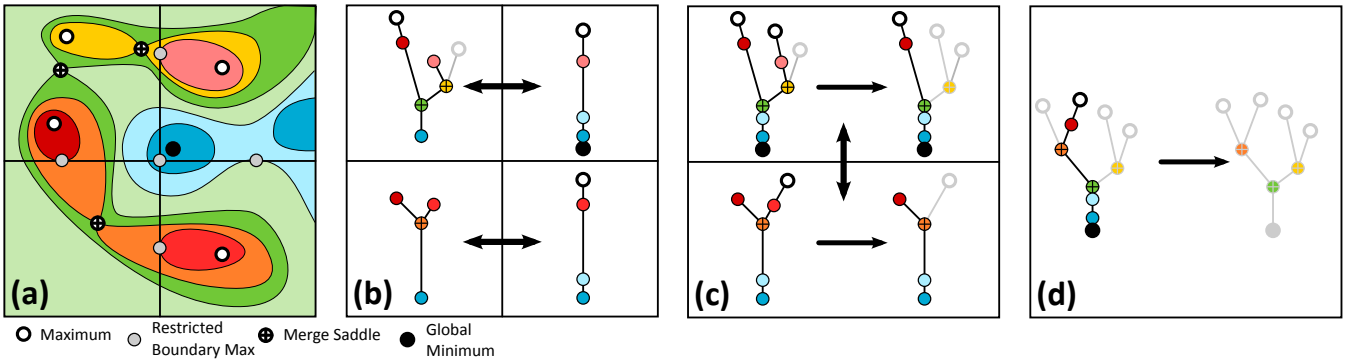


Figure 4: An example of a parallel merge tree computation using four processes in a binary hierarchy. For illustration purposes the entire tree is shown at each state with elements of the boundary tree shown in black and interior elements in grey. Note that in practice only the black portions would be communicated and processed. (a) The contour plot of a 2D function color coded by function value with the critical points and restricted boundary maxima highlighted. Note that the global boundary does not induce restricted maxima. (b) The local merge trees of the four pieces with the first merges indicated by the arrows. (c) The AMT after the merge (left) and the cleaned up tree (right). (d) The final AMT (left) and the (empty) cleaned up tree (right).

ponent. Any such vertex may be a critical point in one of the subdomains that share this boundary and in fact may be the overall highest shared vertex. Explicitly, entering all such restricted boundary maxima to the corresponding trees ensures that we can always find all potentially shared critical points between neighboring trees. In practice, for a regular grid divided into blocks this results in adding all eight corners of a subdomain as well as all vertices locally maximal with respect to the twelve edges and six faces. Interestingly, assuming all such restricted maxima are part of all local trees, the algorithm discussed above will create the correct joined tree without any modifications. Finally, it is easy to see that when merging two trees only nodes relevant to restricted boundary maxima can possibly be affected by a merge. Therefore, it is sufficient to only process the corresponding *boundary trees* defined as all nodes and arcs containing at least one restricted boundary maximum in their subtree as well as nodes directly above restricted maxima.

Given the discussion above we can now describe a simple yet highly efficient split of the serial, streaming algorithm into two massively parallel portions and a hierarchical merge. The algorithm is split into three phases: (1) A local compute phase which computes the merge tree of each subdomain in parallel; (2) A hierarchical merge phase in which boundary trees are combined through successive k-way merges; and (3) A local correction phase which successively integrates changes introduced by the merging of trees back into the local trees, see Figure 5.

Local Compute Phase. Given a subdomain we compute the merge tree of the given function using a variant of Carr et al.'s contour tree algorithm [9]. It relies on pre-sorting all vertices followed by a union-find like traversal to construct the tree. For the remainder of the paper we will refer to the trees attached to specific subdomains as *local* trees (see Figure 4(b)). In practice, a merge tree based analysis typi-

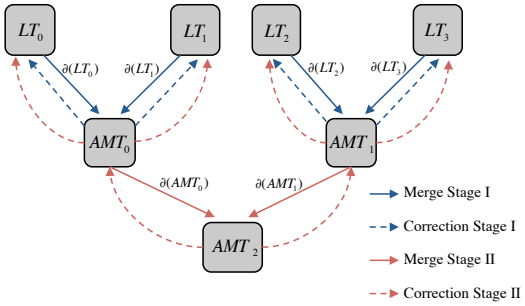


Figure 5: The various stages of the algorithm and the communication involved.

cally indicates that one is interested in high function values and thus the structure below some threshold is often of no interest. Since the lower portion of a merge tree is actually more expensive to compute and store (more vertices in fewer branches) without being useful, we allow the user to specify a cut-off below which vertices are ignored. This can dramatically reduce the file sizes that must be stored and also speeds up the computation in general. To ensure properly shared boundary, we use a ghost zone half a layer wide, essentially only extending the boundaries in positive x, y, and z direction by one vertex. Typically, this information is readily available from the simulation. Furthermore, we detect all restricted boundary maxima and, if they are not already part of the tree. add them as valence two nodes. This phase is embarrassingly parallel and correspondingly scales well. Subsequently, we extract the corresponding boundary trees by walking downwards from all restricted boundary maxima and send them to their appropriate merge location using a point-to-point MPI send.

Merge Phase. At start-up we determine a merge hierarchy consisting of a tree whose leaves correspond to subdomains and/or their local trees and whose other nodes represent

merges of boundary trees, see Figure 5. Constructing the hierarchy is extremely flexible and supports arbitrary k-way merges in any order the user chooses. In its current form, the algorithm merges based on the row major order of the subdomains in the obvious manner. We merge k trees using the variant of the streaming-serial algorithm discussed above except that we initially maintain the AMT rather than removing valence two nodes. We then send the AMT back up the hierarchy to all leaves where it will be used to integrate global information into the respective local tree. Subsequently, we reduce the AMT by removing all valence two nodes that are not part of the boundary of the union of all corresponding subdomains. Finally, we compute the boundary tree of the result and send it to the next stage of the merge hierarchy. Figure 4 shows a 2D example using four processors in a binary hierarchy. In this manner each merge processes information from increasingly larger subdomains but as only the remaining boundary information is actually used the intermediate trees remain small. By construction, the output of the root of the merge hierarchy will be an empty tree as no more shared boundaries exist and thus the boundary tree must be empty.

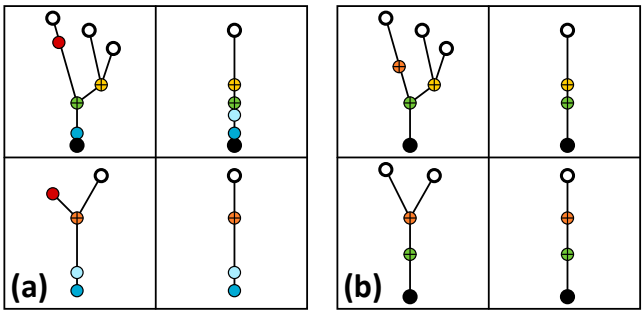


Figure 6: Local merge trees from Figure 4 after the first (a) and second (b) round of corrections.

Correction Phase. The initial local trees may be incorrect in the sense that they may contain nodes and arcs not part of the global merge tree (see Figure 4(b)). In particular, there can exist local maxima and saddles which are regular vertices with respect to the global tree and the local minimum is likely not the global one. Each merge phase integrates the information of multiple trees into a more complete joined tree and we use this joined AMT to correct the local trees (see Figure 6). More specifically, for each node in the local tree beneath a restricted boundary maximum we search whether a corresponding node appears in the joined AMT. If so, we combine both copies and, if a split is created, glue both trees as described above. When this process creates valence two nodes no longer part of a shared boundary, these are removed. In the process we copy all necessary nodes from the AMT to the local tree and delete the AMT at the end. To avoid accumulating unnecessary information, we only keep nodes with function values close to the local range of the corresponding subdomain. More specifically, we maintain the lowest critical points above the local maximum and the highest critical point below the local minimum. Once the AMTs of all merges have been processed, each subdomain maintains a small portion of the global tree with some unavoidable duplications between nodes (see Figure 6(b)).

Segmentation. In practice the merge tree itself is of limited use without the segmentation describing the shape and location of the corresponding features. We compute the segmentation in two parts. First, during the local compute phase we also compute the corresponding segmentation by identifying each vertex with the next highest critical point in the local tree. During the correction phase we then keep track of the changes that occur, in particular of the nodes that are removed and their immediate higher neighbor at the time of the removal. Using the algorithm discussed in [6] this is sufficient to update the segmentation information of all local vertices in a union-find type sweep through all local vertices. Finally, we dump all vertices, their segmentation id, as well as all nodes and arcs of the local trees to disk as individual files per subdomain. These files are then integrated in a simple post-process into the global merge tree and the corresponding segmentation.

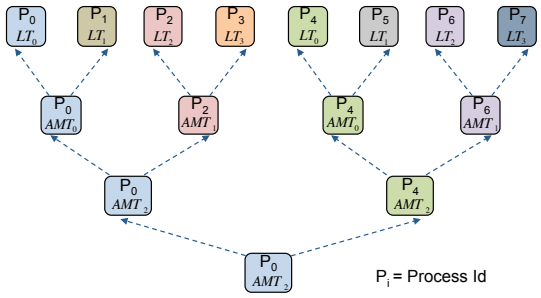
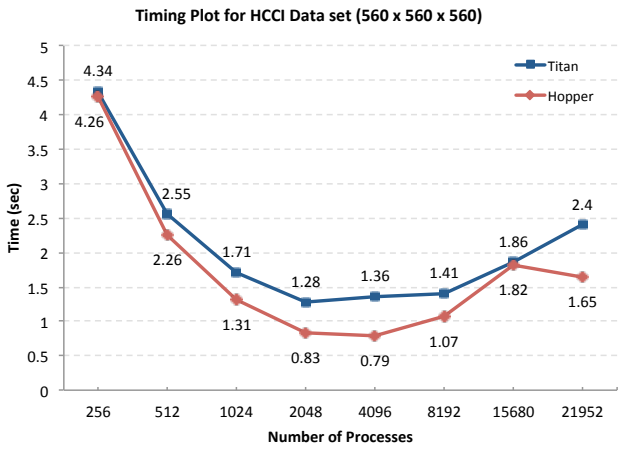


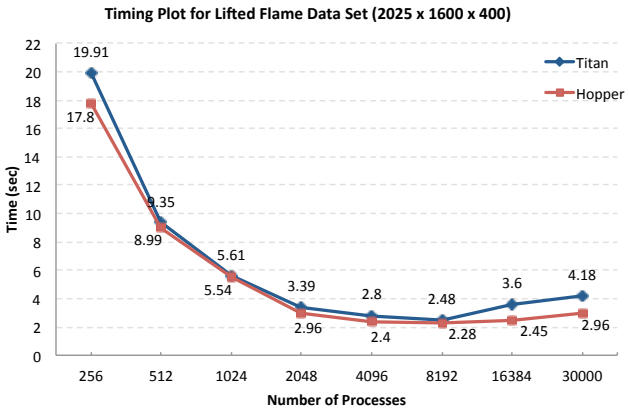
Figure 7: The process layout for an 8 process run. A snapshot of the upstream communication during the final correction phase is shown. Notice the way in which messages are relayed to reach the leaves instead of the root doing a broadcast to the leaves.

Layout and Communication. One significant advantage of the algorithm described above is its flexibility no only in terms of deciding the merge hierarchy but also in placing the corresponding computation. Each merge represents an independent compute kernel and can be placed arbitrarily on any compute core. In principle, the same is true for the local compute and the correction phase but since these are tied more closely to the input data, one typically places them with the data to avoid unnecessary data movement. For simplicity we currently assign each merge to the core with the lowest MPI rank within its subtree as shown in figure 7. However, other assignments are possible and may lead to more optimal load balancing. All communication is handled as point-to-point MPI messages except those between cores on the same processor which our communication layer automatically handles through a direct memory transfer. We further optimize communication to route messages through the merge hierarchy instead of sending individual messages to all leaves during the correction phase. This avoids any one process having to send an excessive number of messages. Figure 7 illustrates this behavior by showing the communication involved in the final correction phase of an 8 process run.

5. RESULTS



9(a)



9(b)

Figure 8: Time taken by the analysis on Hopper and Titan for (a) the HCCI ($560 \times 560 \times 560$) data set and (b)the Lifted Flame data set with various process counts.

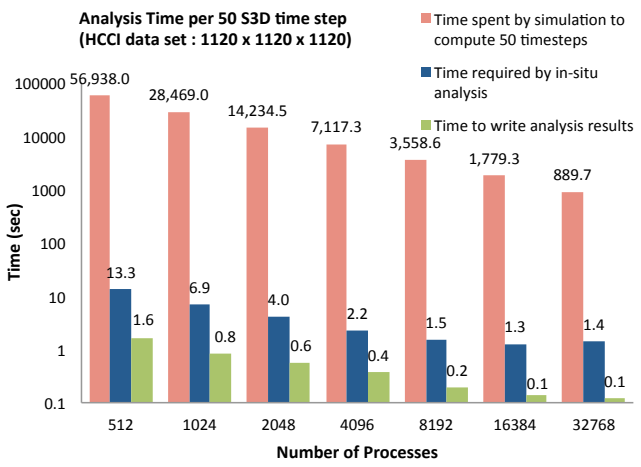
To demonstrate and validate our approach we have applied it to several test data sets as well as two different large scale combustion simulations originally generated by S3D, a massively parallel turbulent combustion code. S3D performs first principles based direct numerical simulations of turbulent combustion. In these simulations, both turbulence and chemical kinetics associated with burning gas phase hydrocarbon fuels introduce spatial and temporal scales spanning typically at least 5 decades. S3D operates on a three-dimensional regular grid typically using a domain decomposition around $30 \times 30 \times 30$ grid points per processor. For our in-situ test case described below, we use the identical core counts, domain decompositions, and data distribution used in the original simulations. We use the DIY library [27] for reading the data at these decompositions and obtaining the same data distribution. Due to the large I/O overheads data snapshots are normally saved only every 500th time step which has been shown in the past to be too few for reliably tracking, for example, regions of locally high scalar dissipation [21]. Instead, here we show how performing the same type of analysis demonstrated in [21, 3] in-situ can be done every 50th time step, thus increasing the effective temporal resolution by an order of magnitude while adding less than one percent to the overall execution time.

Datasets and Computing Environment We use four different regular grid datasets for our experiments. The Vertebra data set is a rotational angiography scan of a head aneurysm obtained from http://www.volvis.org with dimensions $512 \times 512 \times 512$. This is the largest publicly available dataset used in previous work on distributed merge tree computation [24] and is included for comparison. The HCCI data set is a $560 \times 560 \times 560$ simulation of a homogenous charge compression ignition process in which a lean, premixed fuel-air mixture is compressed until it ignites spontaneously in many separate locations. The HCCI data was generated on Jaguar (now Titan) at the Oak Ridge Leadership Computing Facility (OLCF) using 21,952 cores with $20 \times 20 \times 20$ grid points per processor. We have constructed a larger version by repeating the periodic HCCI data eight times to form a $1120 \times 1120 \times 1120$ cube, to act as a stand-in for future larger simulations. Finally, the Lifted Flame dataset is a $2025 \times 1600 \times 400$ volume used to investigate turbulent lifted flames with the goal of better understanding direct injection stratified spark ignition engines for commercial boilers, as well as fundamental combustion phenomena. The lifted flame data was originally generated on Jaguar using 30,000 cores with $27 \times 40 \times 40$ grid points per processor.

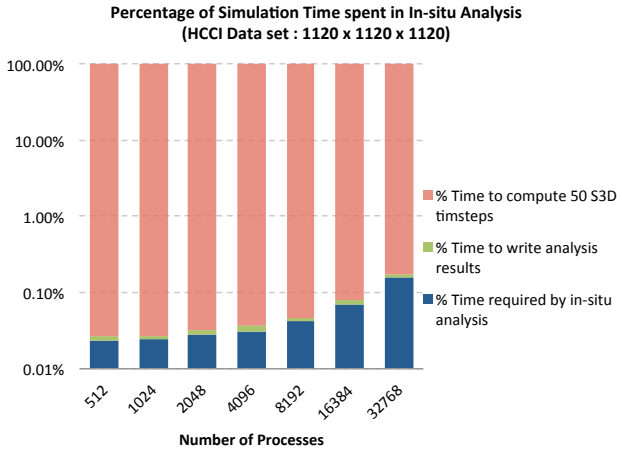
For our tests, we use both the Hopper system at the National Energy Research Scientific Computing Center (NERSC) and Titan at the OLCF. Hopper is a peta-flop Cray XE6 system consisting of 6,384 nodes each with 2 twelve-core AMD MagnyCours 2.1Ghz processors resulting in a total of 153,216 compute cores. Titan is a peta-flop Cray XK7 system with 18,688 nodes each with a sixteen-core AMD Opteron 2.2 Ghz processor for a total of 299,008 compute cores.

5.1 Performance of the Analysis Technique

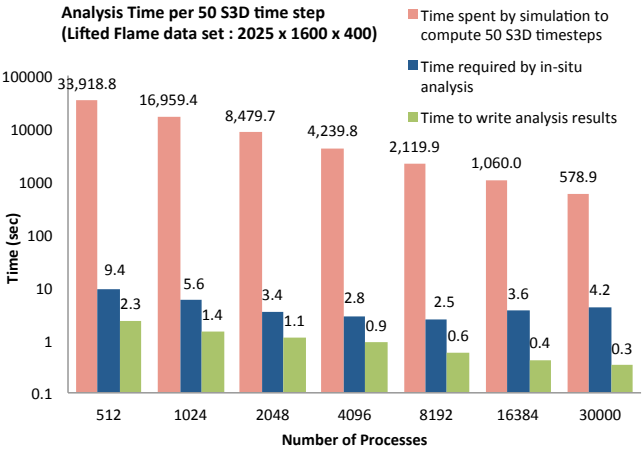
Figure 9 shows the times taken by the analysis for various data sets and process counts on Hopper. For the combustion data sets, our approach achieves reasonable strong scaling until approximately 4K to 8K processes. For data sets on higher processor counts, the workload per-process becomes too small for additional cores to be beneficial and the run-times are dominated by the communication necessary to resolve the global structures in the data. Nevertheless, as discussed in more detail below, the absolute times are small enough that they have only a marginal impact on the overall execution time of the simulation. Furthermore, past 4K processes the absolute time taken for the analysis is actually smaller than the variational difference between runs due to different node allocations and general system noise. Figure 8 shows a comparison of the time taken by the analysis for HCCI and Lifted Flame data sets on Hopper and Titan. We can see that the trend is similar on both the machines. The analysis takes less time on Hopper than Titan, however the absolute times are too low to make a fair assessment.



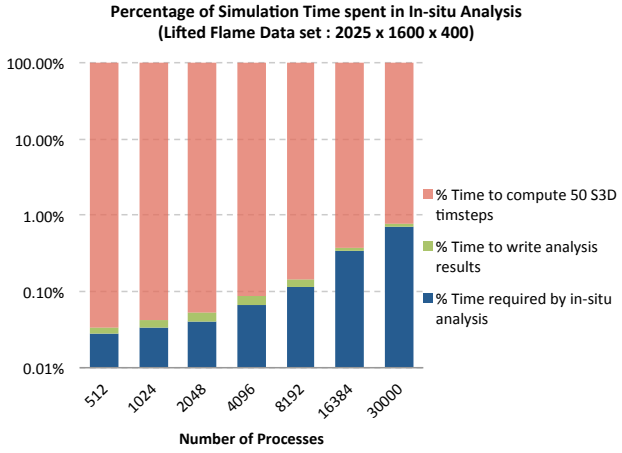
9(a)



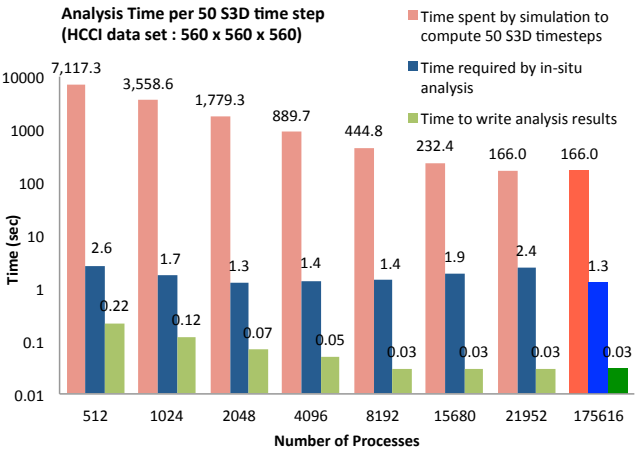
9(b)



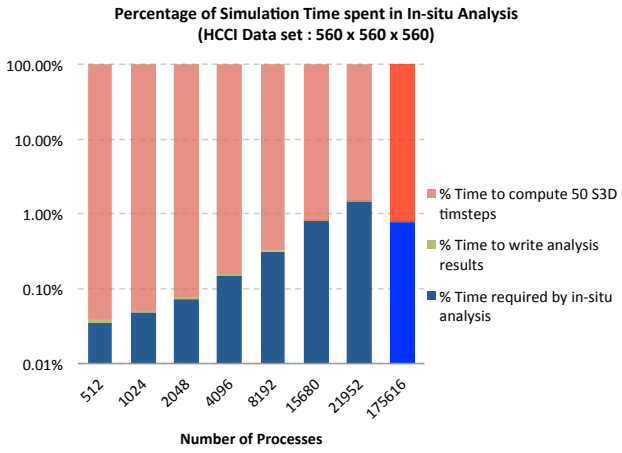
9(c)



9(d)



9(e)



9(f)

Figure 10: Time spent by the in-situ analysis as compared to the time taken by S3D simulation to compute 50 time steps. The absolute times and the percentage overhead added by the analysis along with storing of the results for the HCCI($1120 \times 1120 \times 1120$) data set((a) & (b)) , for the Lifted Flame data set((c) & (d)) and the HCCI($560 \times 560 \times 560$) data set((e) & (f)). Notice that the time taken by the analysis along with writing the results is below 1 percent of the total simulation time and also note the projected performance of the analysis for 175,616 cores in rightmost columns of (e) & (f).

Data set	Original Size	Threshold Value	Reduced Size	Data Reduction
HCCI ($560 \times 560 \times 560$)	670 MB	0.01	220 MB	67%
	670 MB	0.02	182 MB	73%
	670 MB	0.1	106 MB	84%
	670 MB	0.2	68 MB	90%
Lifted Flame ($2025 \times 1600 \times 400$)	4.9 GB	10.0	970 MB	80%
	4.9 GB	20.0	641 MB	88%
	4.9 GB	50.0	336 MB	93%
	4.9 GB	70.0	252 MB	95%

Table 1: Data Size Reduction After Segmentation

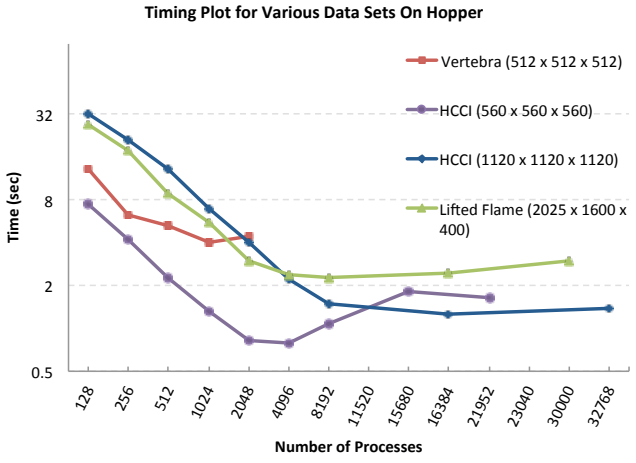


Figure 9: Time taken by the analysis for various data sets and process counts on Hopper.

As discussed above, one important degree of freedom is the choice of fan-in for the hierarchical merge phase. Larger fan-ins produce a more shallow merge hierarchy with fewer dependencies and shorter chains of messages. However, larger fan-ins also reduce the number of active cores more quickly resulting in a more unbalanced load, potentially causing problems. Our experiments suggest that the optimal performance is typically achieved by keeping the tree roughly at a constant depth of around 4 to 5 levels. For up to 256 processors we use a binary merge, up to 1024 a 4-way, up to 8K cores an 8-way, and beyond that a 16-way merge. However, this trend does not continue much beyond 16 on current systems. Experiments with a 32-way merge show a clear slowdown, most likely due to messages begin forced off node. Hopper, for example, has only 24 cores per node and assembling consecutive groups of 32 subdomains for a merge creates a significant increase in the amount of off-node communication, causing a slow down.

For the Vertebra data set, our algorithm takes 5.34 seconds at 512 processors which is 12% faster than the algorithm presented in [24] at 6.04 seconds for the same data set on the same machine. Furthermore, their algorithm slows down at 1024 processors while our algorithm still improves to 4.04 seconds, only becoming marginally slower at 2048 cores. This suggests better scaling of our approach even though we compute the segmentation information in addition to the merge tree. Morozov and Weber [24] report results only up to 2048 processor making it difficult to compare the performance at core counts relevant for in-situ analysis.

In terms of in-situ analysis, the absolute times of the previous section are less important than the relative times with respect to the corresponding simulation. Assuming perfect scaling, S3D's running times have been observed to be on average 4.15e-4 seconds per grid point per time step for the HCCI data set and 2.68e-4 seconds per grid point per time step for the Lifted Ethylene Jet data set. For our experiments we assume an analysis is performed every 50th time step, a ten-fold increase beyond what is currently possible in post-processing. Using these numbers, we can estimate the time S3D would require to execute 50 time steps and compare it to the time our analysis would add to the process. Figure 10(a), 10(c) and 10(e) show the resulting run-time break downs on a log scale for the three data sets. Even at the largest scale, the analysis, including the corresponding file I/O, adds only around 1% to the overall execution time while increasing the amount of information provided to the scientists by an order of magnitude.

Nevertheless, the trend is somewhat troubling: with increasing core counts the analysis may in fact use more time while the predicted times for the simulation are decreasing. However, this trend assumes a perfect scaling of S3D which is unlikely to occur in practice and thus it overstates the problem. Indeed, predictions for future runs include additional chemical species, further increasing the per-vertex cost of the simulation. Furthermore, there exists an obvious mitigation strategy to use fewer cores per node for the analysis. Given that all cores on a node share a common memory system, one can easily access all data on a node from any single core. Apart from effects like non-uniform memory accesses, we have simulated such a situation by increasing the data size per-core. In the right most column of Figure 10(e) we use the larger version of the HCCI data to simulate a corresponding run on 175,616 cores. Using the eight times larger domain with the same decomposition of $20 \times 20 \times 20$ grid points per processor and assuming 16 cores per node, we use 10,976 cores on hopper, each operating on $80 \times 40 \times 40$ grid points. This is equivalent to a simulation running on 175,616 cores where only one core per node is involved in the analysis. However, there are some notable differences between running on 11K nodes rather than 176K nodes. In the standard data layout, many of the messages, especially those early in the merge phase, remain on-node while at full

scale, by construction, all messages would be sent over the network. To compensate for this effect we alter the data layout to assign data blocks in a round robin fashion which ensures that the vast majority of messages are in fact off-node. Note that this in fact overcompensates, as during a full scale run at 176K cores each analysis core would have access to the entire network bandwidth of a node while in our test the network is shared amongst all 24 cores on each node of Hopper. As shown in Figure 10(e), even assuming a perfect scaling of S3D, our algorithm would again add less than 1% overhead.

5.2 Data Reduction

As discussed in Section 4 we compute both the merge tree as well as the corresponding segmentation to allow as much flexibility in post-processing as possible. In fact, assuming a reasonable threshold, the tree combined with the segmentation has the same information in terms of feature based analysis as the original field. However, depending on the threshold, the segmentation is significantly smaller than the original data and the trees are of negligible size on the order of several MB. For simplicity, we currently store the segmentation as a flat list of vertex - segmentation id pairs for all vertices above the threshold. Table 1 reports the resulting data sizes and corresponding data reduction for different thresholds. For all timing results we use the lowest reported thresholds (0.01 for HCCI and 10 for the Lifted Flame) as the worst case behavior. This results in a data reduction between 67% and 80%. If necessary the data could easily be compressed but even in its current form the file I/O has virtually no impact on the overall run time making this optimization a low priority. Furthermore, in practice one would likely use more aggressive thresholds. Notice for example, the differences between the Lifted Flame at a threshold of 10 (Figure 11(d)) compared to a threshold of 50 (Figure 11(f)). The majority of additional volume is added on the outside of the flame in single solid "feature" that is of little practical interest. From a physics perspective, the goal is to extract the individual regions of high scalar dissipation that start to appear at higher thresholds. Thus a threshold of 50 appears to be far sufficient to include all phenomena of interest, which improves the data reduction to 93%. In summary, our approach allows scientists to freely and interactively explore feature based segmentations at ten times the temporal resolution of current approaches, and has the potential for significant scientific impact.

6. CONCLUSION AND FUTURE WORK

If recent hardware trends continue, in-situ analysis will rapidly become a crucial component of scientific computing. This paper presents, for the first time, a framework capable of computing a state of the art topological analysis in-situ with data sizes and core counts matching the corresponding simulation yet having negligible impact to the overall execution time. Furthermore, our framework is highly flexible and as a result can be easily adapted to various computing environments and scenarios. We achieved ten-fold increase in temporal resolution of the analysis compared to existing approaches with minima cost, which enables domain scientists to reliably track features and analyze their evolution over time. Future research will be to explore other options such as co-processing or in-transit solutions to further reduce the impact on the simulation as well as improving the overall performance to the extent possible.

Acknowledgments

This work was supported in part by funding from the DOE Office of Science Advanced Scientific Computing Research (ASCR). The authors wish to thank the members of the the ExaCT Center for Exascale Simulation of Combustion in Turbulence for useful discussions and support. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

7. REFERENCES

[1] H. Abbasi, G. Eisenhauer, M. Wolf, K. Schwan, and S. Klasky. Just In Time: Adding Value to The IO Pipelines of High Performance Applications with JITStaging. In Proc. of 20th International Symposium on High Performance Distributed Computing (HPDC'11), June 2011.

[2] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. In Proc. of 18th International Symposium on High Performance Distributed Computing (HPDC'09), 2009.

[3] J. Bennett, V. Krishnamoorthy, S. Liu, R. Grout, E. R. Hawkes, J. H. Chen, J. Shepherd, V. Pascucci, and P.-T. Bremer. Feature-based statistical analysis of combustion simulation data. IEEE Trans. Vis. Comp. Graph., 17(12):1822–1831, 2011.

[4] J.-M. F. Brad Whitlock and J. S. Meredith. Parallel In Situ Coupling of Simulation with a Fully Featured Visualization System. In Proc. of 11th Eurographics Symposium on Parallel Graphics and Visualization (EGPGV'11), April 2011.

[5] P.-T. Bremer, H. Edelsbrunner, B. Hamann, and V. Pascucci. Topological hierarchy for functions on triangulated surfaces. IEEE Transactions on Visualization and Computer Graphics, 10:385–396, 2004.

[6] P.-T. Bremer, G. Weber, J. Tierny, V. Pascucci, M. Day, and J. B. Bell. Interactive exploration and analysis of large scale simulations using topology-based data segmentation. IEEE Trans. on Visualization and Computer Graphics, 17(99), 2010.

[7] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell. Interactive exploration and analysis of large-scale simulations using topology-based data segmentation. IEEE Transactions on Visualization and Computer Graphics, 17:1307–1324, 2011.

[8] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In Proc. of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 918–926, New York, NY, USA, Jan. 2000. ACM, ACM Press.

[9] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. Comput. Geom. Theory Appl., 24(3):75–94, 2003.

[10] H. Carr, J. Snoeyink, and M. van de Panne.

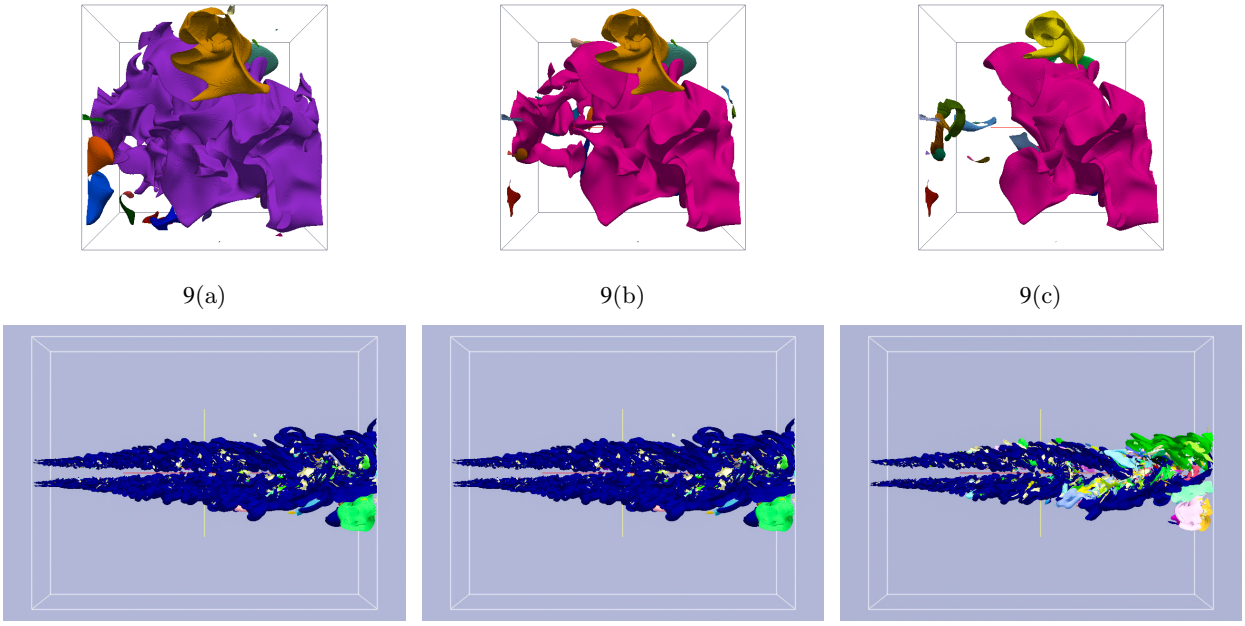


Figure 11: Visualization of the segmentation for the HCCI($560 \times 560 \times 560$) data set for threshold values (a) 0.01, (b) 0.02 and (c) 0.05, and for the Lifted Flame data set for threshold values (d) 10.0, (e) 20.0 and (f) 50.0.

Simplifying flexible isosurfaces using local geometric measures. In IEEE Visualization '04, pages 497–504. IEEE Computer Society, 2004.

[11] J. H. Chen, A. Choudhary, B. de Supinski, M. DeVries, E. R. Hawkes, S. Klasky, W. K. Liao, K. L. Ma, J. Mellor-Crummey, N. Podhorski, R. Sankaran, S. Shende, and C. S. Yoo. Terascale direct numerical simulations of turbulent combustion using s3d. Computational Science and Discovery, 2:1–31, 2009.

[12] C. Docan, M. Parashar, and S. Klasky. DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows. In Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10), June 2010.

[13] H. Edelsbrunner, J. Harer, and A. Zomorodian. Hierarchical Morse-Smale complexes for piecewise linear 2-manifolds. Discrete Computational Geometry, 30:173–192, 2003.

[14] N. Fabian, K. Moreland, D. Thompson, A. Bauer, P. Marion, B. Gevecik, M. Rasquin, and K. Jansen. The paraview coprocessing library: A scalable, general purpose in situ visualization library. In Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV), pages 89 –96, October 2011.

[15] 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.

[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, 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.

[18] A. Gyulassy, T. Peterka, R. Ross, and V. Pascucci. The parallel computation of Morse-Smale complexes. IEEE International Parallel and Distributed Processing Symposium, to appear, 2012.

[19] S. Lakshminarasimhan, J. Jenkins, I. Arkatkar, Z. Gong, H. Kolla, S.-H. Ku, S. Ethier, J. Chen, C. Chang, S. Klasky, R. Latham, R. Ross, and N. Samatova. Isabela-qa: Query-driven analytics with isabela-compressed extreme-scale scientific data. In Proc. of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 1 –11, November 2011.

[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 Transactions on Visualization and Computer Graphics, 12(5):1053–1060, Sept. 2006.

[21] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. Chen. Topological feature extraction for comparison of terascale combustion simulation data, pages 229–240. Mathematics and Visualization. Springer, 2011.

[22] A. Mascarenhas, R. W. Grout, P.-T. Bremer, E. R. Hawkes, V. Pascucci, and J. H. Chen. Topological

feature extraction for comparison of terascale combustion simulation data. In V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, editors, Topological Methods in Data Analysis and Visualization, Mathematics and Visualization, pages 229–240. Springer Berlin Heidelberg, 2011.

[23] K. Moreland, R. Oldfield, P. Marion, S. Jourdain, N. Podhorszki, V. Vishwanath, N. Fabian, C. Docan, M. Parashar, M. Hereld, M. E. Papka, and S. Klasky. Examples of in transit visualization. In Proceedings of the 2nd international workshop on Petascal data analytics: challenges and opportunities, PDAC '11, pages 1–6, New York, NY, USA, 2011. ACM.

[24] D. Morozov and G. H. Weber. Distributed merge trees. In A. Nicolau, X. Shen, S. P. Amarasinghe, and R. Vuduc, editors, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013, pages 93–102. ACM, 2013.

[25] V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. Algorithmica, 38(1):249–268, Oct. 2003.

[26] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. ACM Trans. Graph., 26(3), July 2007.

[27] T. Peterka, R. Ross, W. Kendall, A. Gyulassy, V. Pascucci, H.-W. Shen, T.-Y. Lee, and A. Chaudhuri. Scalable parallel building blocks for custom data analysis. In Proceedings of Large Data Analysis and Visualization Symposium LDAV'11, Providence, RI, 2011.

[28] G. Reeb. Sur les points singuliers d'une forme de pfaff completement intergrable ou d'une fonction numerique [on the singular points of a complete integral pfaff form or of a numerical function]. Comptes Rendus Acad.Science Paris, 222:847–849, 1946.

[29] A. Tikhonova, H. Yu, C. D. Correa, J. H. Chen, and K.-L. Ma. A preview and exploratory technique for large-scale scientific simulations. In T. Kuhlen, R. Pajarola, and K. Zhou, editors, Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK, 2011. Proceedings, pages 111–120. Eurographics Association, 2011.

[30] T. Tu, H. Yu, L. Ramirez-Guzmanz, J. Bielak, O. Ghattas, K.-L. Ma, and D. R. O'Hallaron. From Mesh Generation to Scientific Visualization: An End-to-End Approach to Parallel Supercomputing. In Proceedings of ACM/IEEE Supercomputing Conference, 2006.

[31] V. Vishwanath, M. Hereld, and M. Papka. Toward simulation-time data analysis and i/o acceleration on leadership-class systems. In Proc. of IEEE Symposium on Large Data Analysis and Visualization (LDAV), October 2011.

[32] S. Williams, M. Petersen, P.-T. Bremer, M. Hecht, V. Pascucci, J. Ahrens, M. Hlawitschka, and B. Hamann. Adaptive extraction and quantification of atmospheric and oceanic vortices. IEEE Trans. Vis. Comp. Graph., 17(12):2088–2095, 2011.

[33] H. Yu, T. Tu, J. Bielak, O. Ghattas, J. C. López, K.-L. Ma, D. R. O'Hallaron, L. Ramirez-Guzmanz, N. Stone, R. Taborda-Rios, and J. Urbanic. Remote Runtime Steering of Integrated Terascale Simulation and Visualization. In ACM/IEEE Supercomputing Conference HPC Analytics Challenge, 2006.

[34] H. Yu, C. Wang, R. Grout, J. Chen, and K.-L. Ma. In Situ Visualization for Large-Scale Combustion Simulations. IEEE Computer Graphics and Applications, 30(3):45–57, 2010.

[35] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf. PreDatA - preparatory data analytics on peta-scale machines. In Proc. of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), April 2010.