In-Situ Feature Extraction of Large Scale Combustion Simulations Using Segmented Merge Trees

Aaditya G. Landge*, Valerio Pascucci*, Attila Gyulassy*, Janine C. Bennett‡, Hemanth Kolla‡, Jacqueline Chen‡, and Peer-Timo Bremer*†

*SCI Institute, University of Utah, Salt Lake City, UT
†Lawrence Livermore National Laboratory, Livermore, CA
‡Sandia National Laboratory, Livermore, CA

***Abstract*—The ever increasing amount of data generated by scientific simulations coupled with system I/O constraints are fueling a need for in-situ analysis techniques. Of particular interest are approaches that produce reduced data representations while maintaining the ability to redefine, extract, and study features in a post-process to obtain scientific insights.**

This paper presents two variants of in-situ feature extraction techniques using segmented merge trees, which encode a wide range of threshold based features. The first approach is a fast, low communication cost technique that generates an exact solution but has limited scalability. The second is a scalable, local approximation that nevertheless is guaranteed to correctly extract all features up to a predefined size. We demonstrate both variants using some of the largest combustion simulations available on leadership class supercomputers. Our approach allows state-of-the-art, feature-based analysis to be performed in-situ at significantly higher frequency than currently possible and with negligible impact on the overal simulation runtime.

***Keywords*—topological data analysis, feature extraction, in situ analysis, merge tree computation, segmented merge tree**

I. Introduction

The continuing increase in available computing power allows scientists to simulate ever more complex phenomena at higher temporal and spatial resolutions. Correspondingly, the analysis of these datasets is becoming increasingly sophisticated, moving from global to local statistics and more recently to detailed studies of small, intermittent features of interest along with their characteristics and temporal evolution [1]–[3]. However, while the need for advanced data analysis techniques increases, the (relative) amount of data that can be permanently stored keeps decreasing. This can severely impede and may ultimately prevent an accurate and reliable analysis. State-of-the-art simulations are already reaching the point at which snapshots are stored too infrequently to accurately track fast moving or intermittent events, increasing the likelihood that potentially important phenomena are lost between snapshots.

While there exist a number of mitigating strategies such as compression [4] or advanced data management techniques [5], [6], the challenges discussed above will likely only be addressed by moving the analysis in-situ i.e., to perform it concurrently with the simulation. Since analysis results are typically orders of magnitude smaller than the original data, efficient in-situ algorithms would allow an effective analysis at much higher frequencies than otherwise feasible. To this end a number of in-situ visualization and analysis techniques have been proposed [7]–[11] either as stand alone tools or as part of existing systems. However, so far these efforts have been restricted to comparatively simple and largely data parallel operations and few solutions for more complex algorithms exist [12]. Furthermore, most of these analyses were designed in the context of a post-processing workflow, in which scientists test hypotheses by interactively adjusting input parameters to analysis algorithms that provide a single answer to a given question, to slowly converge to their results. In an in-situ setting, however, all parameters, spatial sub-domains, temporal windows, etc., must be specified *a priori*, making current algorithms ineffective at best and misleading at worst. Instead, a new kind of meta-analysis is required that can efficiently compute and encode a range of answers for an entire class of questions, effectively re-enabling a flexible and unbiased exploration of the results in post-processing.

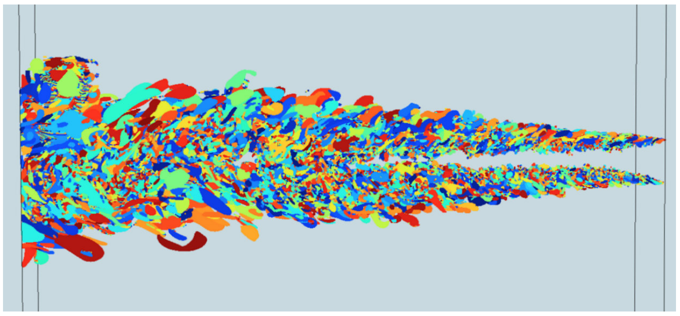


Fig. 1. Extinction regions in a lifted ethylene jet flame extracted using segmented merge trees and adaptive relevance thresholds.

One promising class of techniques are topology-based segmentations based on merge trees [13], contour trees [14], or Morse-Smale complexes [15]. These techniques segment the domain into features according to either the level-set (e.g. thresholding) or gradient behavior of one of the simulation variables. In particular, segmentations of the domain derived from merge trees have been shown to efficiently encode threshold-based features. For example, as shown in Fig. 1, segmented merge trees can be used to extract extinction regions defined as areas of high scalar dissipation in turbulent combustion simulations. Other examples include burning cells in turbulent combustion [13], eddies in the oceans [16], and bubbles in Raleigh-Taylor instabilities [17]. Segmented merge-trees provide two key advantages over traditional threshold-based segmentation techniques: 1) they efficiently encode a wide range of possible segmentations, and 2) they allow for the selection of localized thresholds [18]. However, existing approaches are restricted to off-line serial computations [13] and even solutions to the simpler problem of computing merge trees without segmentation have been limited to small-scale parallel efforts [14], [19].

In this paper we present the first large-scale parallel algorithms to compute segmented merge trees. In particular, we present a new framework consisting of three simple to implement functions that can be assembled into different variants of a parallel algorithm with different characteristics. We demonstrate two variants: first, computing globally consistent merge tree segmentations; and second, computing locally consistent approximations which nevertheless provide strong correctness guarantees. Our global algorithm performs on par with the best previous merge tree approach while at the same time additionally computing the corresponding segmentation. We show that our approach allows for the extraction of features large-scale at ten times the frequencies of current snapshots with minimal impact on the overall performance of the simulation. In addition, we show that our local approach strictly limits the computations at any scale, is an order of magnitude faster than the global approach, and yet guarantees correct segmentation of all features below a predefined size. We demonstrate our results using S3D [20] a large-scale direct numerical simulation code that models combustion in turbulence, and show that in an in-situ setting our approach allows for highly sophisticated, exploratory, feature-based analysis at the full scale of the simulation and with minimal overheads.

II. 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 [7], [21], [22], recent efforts are exposing additional analysis capabilities, [10], [11], [23]. Initial in-transit frameworks focused largely on mitigating I/O costs [24]–[27], however more recent efforts have begun to integrate analysis prior to dumping data to disk [9]. The development of infrastructures and frameworks that enable in-situ visualization and analysis is also on the rise with a focus to reduce I/O and perform efficient data management [28], [29] and/or use idle resources from the simulation for analysis [30].

Topology-based representations of scalar functions provide a discrete structure upon which to formulate queries for robust feature extraction. Reeb graphs [31] and their variants, contour trees [32] and merge trees, encapsulate level set behavior, and Morse- and Morse-Smale complexes [33] 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 [18], [33], [34]. Topology-based techniques have proven useful for sophisticated analysis in computational sciences, for example, in the identification of features from turbulent combustion [13], [18], the detection of bubbles in turbulent mixing [35], or the extraction of the core structure of a porous solid [36].

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 [32], [33], [37], [38], streaming out-of-core [13], [39], or small-scale parallel [14], [19], [40]. Similar to other complex analysis techniques, the global nature of topological feature extraction makes it challenging to develop scalable, parallel algorithms, (since any one feature may span the entire domain). Furthermore, the algorithms are highly data dependent leading to severe load imbalances and are largely not data parallel leading to poor scaling. As a result few parallel algorithm exists.

Gyulassy et al. [41] and Bennett et al. [42] propose hybrid solutions in which the initial data parallel aspects of the computation are executed in parallel while leaving the resolution of global features to post-processing. If only the features themselves are of interest and only after the simulation has completed these approaches provide a good alternative to pure post-processing. However, when statistics of other values conditioned on the features are of interest, e.g., the average temperature within a feature, or the results are needed to inform subsequent processing, e.g., feature-based uncertainty quantification, hybrid approaches do not suffice.

The closest comparison to the techniques proposed here are the parallel contour tree approach of [14] and the distributed merge trees of [19]. Both compute level set based trees, however neither includes the corresponding segmentation – which adds substantial additional processing. Pascucci and Cole-McLaughlin use a traditional divide-and-conquer technique followed by a hierarchical merge similar to our global algorithm. However, they construct the entire tree during the merge rather than only resolving the boundary artifacts. This severely amplifies the scaling problem as each successive round in the merge becomes more expensive yet utilizes fewer processors. Ultimately, the entire tree resides on the root processor which can easily lead to memory problems and would make incorporating the segmentation infeasible. Instead, Morozov and Weber [19] compute a truly distributed merge tree using a binary swap type approach. Their approach relies on maintaining a severely reduced and potentially simplified version of the global tree on each local processor thus avoiding the ever growing trees of [14]. Unfortunately, the information that is disregarded locally, while not required to represent the global tree, is necessary to construct the corresponding segmentation which is ultimately how features are described. As discussed in [19] maintaining this information would severely

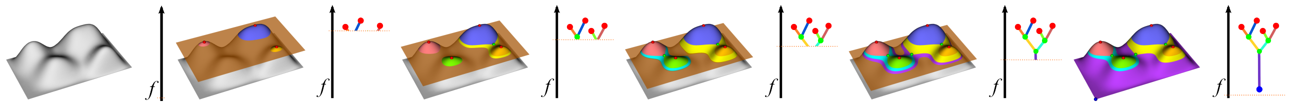


Fig. 2. A merge tree tracks the evolution of super-level sets. In a sweep from ∞ to $-\infty$ each maximum (red sphere) creates a new component, that is merged by saddles (green sphere). The merge tree ends at the global minimum (blue sphere). As indicated by the coloring, each arc in the merge tree represents a subset of the domain and thus the tree can be used directly for feature segmentation.

increase the size of the local trees with substantial impacts to the runtime.

Fundamentally, both approaches rely on exchanging and combining parts of the global tree (either reduced or in its entirety). Instead, our algorithm restricts these exchanges to the minimal set of boundary artifacts introduced by the domain decomposition and distributes all other computation amongst all processors. This allows us to compute both the merge tree itself as well as the corresponding segmentation on par with the tree computation of [19] for medium core counts and to demonstrate efficient computation for orders of magnitudes larger core counts than previously reported. Furthermore, for the first time, we present an approximate algorithm which strictly limits the computation and communication at all scales yet guarantees to correctly extract features of a given size.

III. 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 the merging of components creates *saddles*. The merge tree is composed of *nodes* and *arcs*. The nodes represent the critical points that create, merge, or destroy super-level set components which are the *maxima*, *saddles* and *minima* respectively. 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. We note that analogous definitions exist for split trees that encode the evolution of *sub-level* sets which are those regions of the domain with value in f less than $c \in R$, as c is swept from $-\infty$ to ∞. For simplicity sake, we limit our discussion in this paper to merge trees, however our techniques can be applied to compute split trees with negligible changes to the code.

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 for the pre-computation of various conditional feature-based statistics such as, for instance, average temperatures per feature [2]. Therefore, while the merge tree itself contains only information about the number of features at each threshold, combining the merge tree with its corresponding segmentation creates a powerful and highly flexible analysis tool.

IV. Technique

We start from the domain decomposition of the simulation, which in the case of our example code, S3D, is a block decomposition of a regular grid with each processor, P_i, assigned one block, B_i. However, as discussed below, our framework is flexible and could easily be adapted to non-regular decompositions or block-regular meshes. Similar to the existing parallel approaches, our algorithm operates in two stages. In the first stage we compute a *local tree*, LT_i, (and its segmentation) for the data of block B_i. In the second stage we join all local trees into a unified *global tree* and resolve the associated segmentations across all blocks. More formally, the local tree for a block B_i contains all arcs of the global tree whose corresponding contours intersect B_i.

To support efficient computation of the segmentations in addition to the tree itself, our scheme differs from previous parallel approaches in two ways: First, we communicate only the minimal subset of each local tree that could affect the trees residing on other blocks; and second, on each processor we maintain only the locally relevant portions of the global tree along with the associated information required to construct the local portion of the global segmentation.

Fig. 3(b) depicts the first phase of this computation for a small example, in which we compute the local trees LT_i from each B_i independently. In order to later *join* all LT_i into the global tree, we augment each LT_i with it's block's *restricted maxima*. A restricted maximum is a vertex on the shared boundary of a block that, when restricted to it's lowest dimensional boundary component (a corner, edge or face), is a maximum (see the grey nodes in Fig. 3(a)). Restricted maxima comprise the minimal amount of additional data that we must add to local trees in order to join them into the final global tree (we refer the reader to [14] for a more in depth justification). We call those nodes in the local trees that correspond to restricted maxima *boundary nodes*.

To construct the final distributed global tree, information is iteratively exchanged between pairs of local trees in a hierarchical fashion. Once information is shared between two local trees LT_i, and LT_j, they are considered *joined* and they encode those features that span blocks B_i and B_j on the associated processors P_i and P_j. Intuitively, two neighboring local trees are combined by gluing together their shared boundary nodes and "zipping-up" all arcs towards the root (see Fig. 4). Note that the portions of the local trees that are affected by this process lie beneath the boundary node, and we call these subsets *boundary trees*, *BT*s. Finally, when combining two boundary trees representing two spatial regions the resulting tree is the *augmented boundary tree*, *ABT*, of their combined

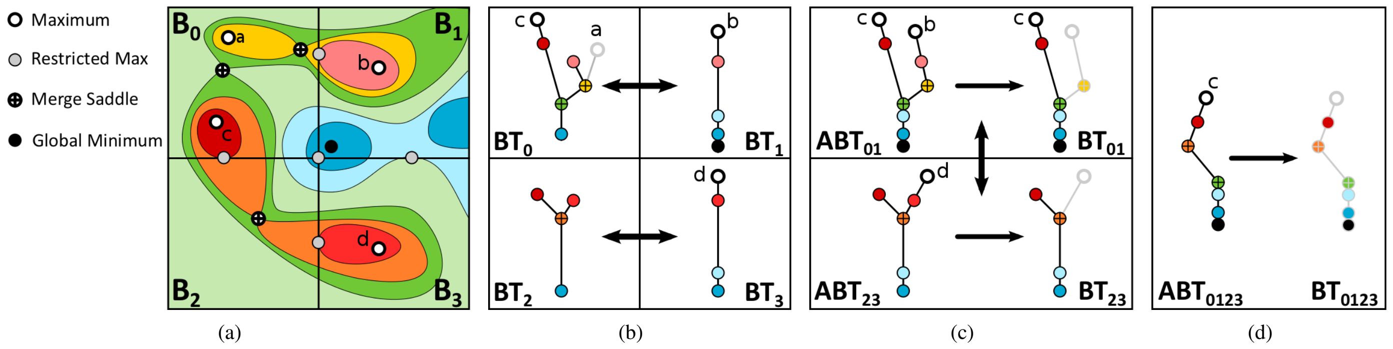


Fig. 3. An example of a parallel merge tree computation using four processes in a binary hierarchy. (a) The contour plot of a 2D function color coded by function value and split into four data blocks (B*) with the critical points and restricted boundary maxima highlighted. The contours of (a) give rise to the local trees in (b) with their respective boundary trees (BT*) drawn black and interior elements in grey. (c) The first horizontal join creates the augmented boundary trees (ABT*) which, after removing now interior nodes, results in the boundary trees of the combined blocks. (d) The following vertical join creates the final augmented boundary tree, whose corresponding boundary tree is empty as there exist no more (shared) boundaries.

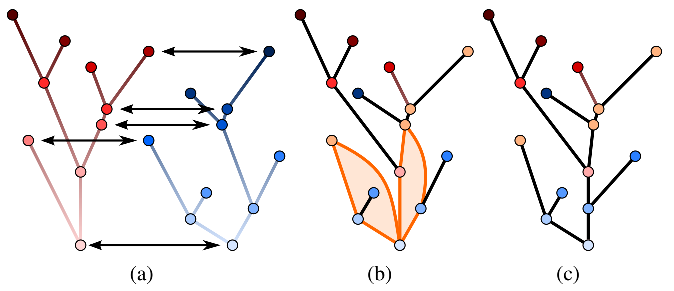


Fig. 4. Joining two trees by glueing loops. (a) Two local trees with some matching nodes indicated by the arrows. (b) Unifying the shared nodes leads to cycles which are removed by glueing their two sides leading to the joined tree of (c).

regions (Fig. 3(c)). The ABT is composed of the boundary tree of the combined regions plus the nodes and arcs on the previously shared but now internal boundary. Removing the internal nodes/arcs of an ABT creates a boundary tree.

A. Algorithm Framework

Given the definitions above, this section provides details regarding our general framework from which we subsequently assemble the two versions of the merge tree computation. In particular, our system consists of three simple routines that can be combined in a highly flexible manner: First, the *local compute* creates an initial local tree; Second, the *join* combines two boundary trees; and Third, the *correction* adjusts a local tree given an ABT.

Local Compute. We compute the local trees of a given block using a variant of Carr et al.'s contour tree algorithm [43]. It relies on pre-sorting all vertices followed by a union-find like traversal to construct the tree. Additionally, we record the corresponding segmentation by storing for each vertex of the domain the id of the corresponding arc in the tree. In practice, a merge tree based analysis typically indicates that one is interested exclusively in relatively high function values (or exclusively low 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. We use a ghost zone half a layer wide, essentially only extending the boundaries in positive x, y, and z direction by one vertex, to ensure that neighboring regions have identical shared boundaries. Typically no additional communication is needed, since the simulation itself maintains significantly larger ghost regions for the quantities of interest. Finally, we add all restricted maxima to the local trees. Given a local tree we extract its boundary tree in a traversal from the root(s) such that the resulting list of arcs is sorted based on the function value of their lower nodes.

Join Routine. Given two (or more) boundary trees as lists of arcs sorted on the function value of their lower nodes, we construct their resulting augmented tree using the Join algorithm shown in Algorithm 1. It takes as input two or more boundary trees, represented as sorted lists of arcs. The function $GetHighestArc()$ returns the arc with the highest-valued lower endpoint from the input boundary trees, and removes it from the corresponding boundary tree. The $AddNode()$ function adds a node with identifier id if a node with that id is not already present in ABT, and returns a reference to the newly added or existing node, respectively. The $LowestDescendant()$ function returns a reference to the lowest node reachable in ABT from its input node. $AddArc()$ updates the structure of ABT, adding an arc between its input nodes if necessary.

Algorithm 1 $Join(BT1, BT2)$

```
$ABT \leftarrow NewABT()$
**while** $a = (n_l, n_u) \leftarrow GetHighestArc(BT1, BT2)$ **do**
    $n_a = AddNode(n_l.id, ABT)$
    $n_b = AddNode(n_u.id, ABT)$
    $n'_a \leftarrow LowestDescendant(n_a, ABT)$
    $n'_b \leftarrow LowestDescendant(n_b, ABT)$
    **if** $n'_a \neq n'_b$ **then**
        $AddArc(n'_a, n'_b, ABT)$
    **end if**
**end while**
**return** $ABT$
```

Since the number of incoming boundary trees is constant the $GetHighestArc$ has constant time complexity. The lookup required for the $AddNode()$ function requires a map from the global index space of the node to the local one of the ABT. Implemented as a hash map the complexity is amortized constant. Finally, the $LowestDescendent()$ function is implemented as a Union-Find data structure with the Union operation keeping the label of the set with the lowest valued node. Therefore, searching for descendants also has constant amortized run time. Thus, the expected runtime for the $Join$ routine is linear in the number of incoming arcs.

Correction of Local Trees. The final component of our framework is the correction of the local trees using augmented boundary trees representing successively larger regions of the domain. The key idea is to only process and store the portions of the augmented tree relevant to the corresponding local block. The corrections to the local tree may involve splitting arcs when non-local saddles are introduced, joining arcs where non-local information determines two features to be connected, and changing the classification of critical points. As discussed above, at any point only portions of the local tree beneath boundary nodes can be affected. However, note that after successive corrections these may include nodes from boundaries of other blocks introduced by previous augmented trees. These represent contours that intersect the local block as well as the boundary of the region represented by the last augmented boundary tree.

The correction routine of Algorithm 2 is very similar to the Join Algorithm. For each boundary node in the LT that may be affected, we find the associated boundary node in the augmented boundary tree ABT, and join the two trees beneath these nodes, zipping towards the root. Note that we explicitly only maintain arcs (and their nodes) in the local tree whose contours intersect the local block by stopping the traversal according to the local function range. Lastly, as discussed above, we compute and store the segmentation of the initial local trees. However, as trees are corrected, their arcs change and in fact some labels disappear entirely (*i.e.* when removing regular nodes). We use the algorithm of [13] to track label changes and correct the segmentation once, after all corrections have finished. This requires a single search of all vertices in each local block for updated labels, which in practice takes a negligible amount of time.

Algorithm 2 $CorrectLocalTree(LT, ABT)$

```
$NodeList \leftarrow SortNodes(LT)$
for all $n_{LT} \in NodeList$ do
  if $n_{LT} = BoundaryNode\, or\, NonLocalMaxima$ then
    $n_{ABT} \leftarrow FindNode(n_{LT}, ABT)$
    // Traverse both trees
    while not reached root of $LT$ or root of $ABT$ do
      if $IsVisited(n_{LT})$ or $IsOutofRange(n_{ABT})$ then
        $break$ // stop traversal
      end if
      if $n_{LT} \rightarrow child \neq n_{ABT} \rightarrow child$ then
        Insert $n_{ABT} \rightarrow child$ into $LT$ and connect
      end if
      $MarkVisited(n_{LT})$
      $n_{ABT} = n_{ABT} \rightarrow child$
      $n_{LT} = n_{LT} \rightarrow child$
    end while
  end if
end for
$RemoveRegularNodes(LT)$
```

Example. Fig. 3 and 5 show an example of the algorithms discussed above using a binary merge (see Section IV-B). Fig. 3(a) shows super-levelsets on a domain split into four quadrants, their critical points, as well the restricted maxima. Fig. 3(b) shows the initial local trees with the boundary trees highlighted. Fig. 3(c) shows the augmented boundary trees after merging horizontally (left) and the resulting boundary trees representing the upper/lower half of the domain (right). Fig. 5(a) shows the local trees after the first round of corrections. Finally, Fig. 3(d) shows the final augmented boundary tree and the resulting boundary tree, which is empty since no more internal boundaries exist. Fig. 5(b) shows the final local trees.

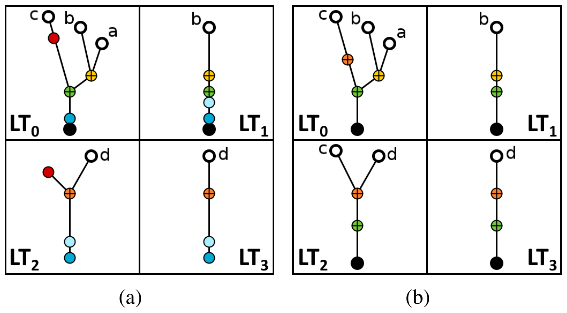


Fig. 5. Local merge trees from Figure 3 after the first (a) and second (b) round of corrections.

Given the three routines described above one can easily assemble different dataflows computing the entire segmented merge tree or local approximations. In particular, the following sections will discuss two different strategies: A hierarchical k-way reduction similar to existing approaches and a region growing strategy that allows partial computations with strong guarantees.

B. k-way Hierarchical Reduction

The simplest way to assemble a global merge tree algorithm from our components is to very similar to a traditional k-way reduction. The dataflow for the binary reduction from Figs. 3 and 5 is shown in Fig. 6. In this case the dataflow is

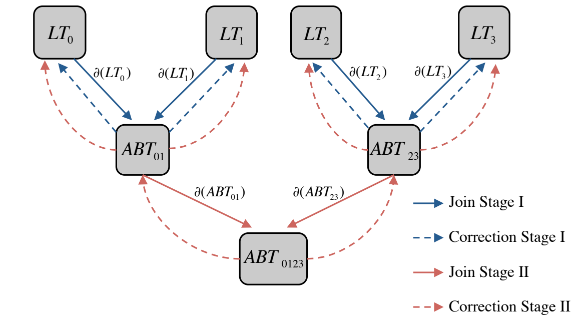


Fig. 6. Dataflow diagram for the binary reduction type merge tree computation of Figs. 3 and 5.

itself a tree, in which the leaves are local trees LT and internal nodes are *joins* of augmented boundary trees. First, all leaves compute their local trees (LT), extract their boundary trees ($BT = \partial(LT)$) and send them to the first join. The resulting augmented trees are sent back up to be used in the correction and the reduced boundary trees are sent downward to the next join. The final augmented tree is then sent back towards the leaves for the second correction. As indicated in the figure, the upward communication is routed along the dataflow to avoid any node having to send an excessive number of messages. This approach is different from existing approaches in that the reduction is restricted to only the minimal information necessary – the boundary trees – while all other computation is handled in a data parallel fashion at the leaves of the dataflow tree.

This dataflow pattern easily supports arbitrary k-way joins with different ks at different levels. One disadvantage is that, since all processors are typically assigned a local block, the joins must be assigned to a subset of these, causing load imbalance and potentially creating bottlenecks. We remark that in an in-transit setting as the one discussed in [42], one can interrupt the joins at any point, store the current local and augmented trees, and finish the computation off-line or on secondary computing resources. However, this comes at the expense of no longer being able to compute dependent statistics in-situ. Ultimately, while this strategy almost perfectly splits the data parallel and global portions of the computation, the remaining reduction still suffers form the traditional problem of too few processors performing work at the lower levels. Therefore, while typically fast in an absolute sense, scaling this approach is challenging.

C. Region Growing

One observation from the reduction based algorithm is that many of the (spatially) smaller features will get resolved relatively early while features spanning a large portion of the domain require the full computation. In general, spatially large features are what fundamentally limits the scalability of our reduction based as well as earlier algorithms [14], [19]. However, in many practical applications one is only interested in relatively small features which is how feature based analysis techniques achieve their speed and flexibility [2], [3], [13]. Therefore, restricting the computation to local neighborhoods may be sufficient to extract all or most features of interest while significantly reducing the scalability problem.

Unfortunately, existing approaches as well as the k-way reduction are not well suited to compute local features. Because the techniques successively join blocks along one of the axes, some decomposition boundaries are resolved in the first step while others are resolved much later. Depending on the actual spatial pattern any one small feature may cross one of the more persistent boundaries and thus stopping the computation early may result in artifacts and/or missing features. Instead, we propose a region growing pattern in which each local block maintains both its local tree as well as a boundary tree for an increasingly large region of the domain centered around itself.

For simplicity we have chosen a uniformly growing boundary as shown in Fig. 7. Each node computes its local tree and exchanges the resulting boundary tree with all its spatial neighbors (Fig. 7(a)). After joining all boundaries trees and performing the corrections, each node is left with a boundary tree that represents the boundary of its *region of influence*, which is the union of its neighbors boundaries – the green region in Fig. 7(a). In the second stage each node exchanges the boundary tree of its region of influence with the neighbors whose regions of influence border that of the node (Fig. 7(b)). This recursive pattern results in regions of influence centered

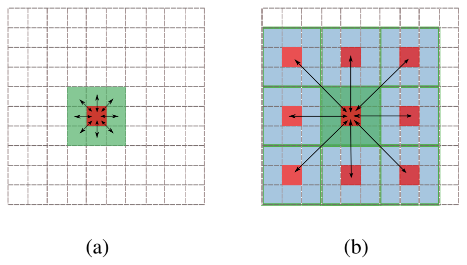


Fig. 7. The region grows exponentially in powers of 3. In (a), the first level of joining is shown where boundaries from immediate neighbors are joined to obtain the region of influence shown in green. In (b), the second level of joining is done where boundaries are exchanged with blocks that are at a distance of 3^1. The region of influence grows to a size of 9×9 shown in blue.

around each block that grow with factor of three thus creating $3 \times 3 \times 3$, $9 \times 9 \times 9$, etc., regions, and distance tripling communication. The generic dataflow for this approach is

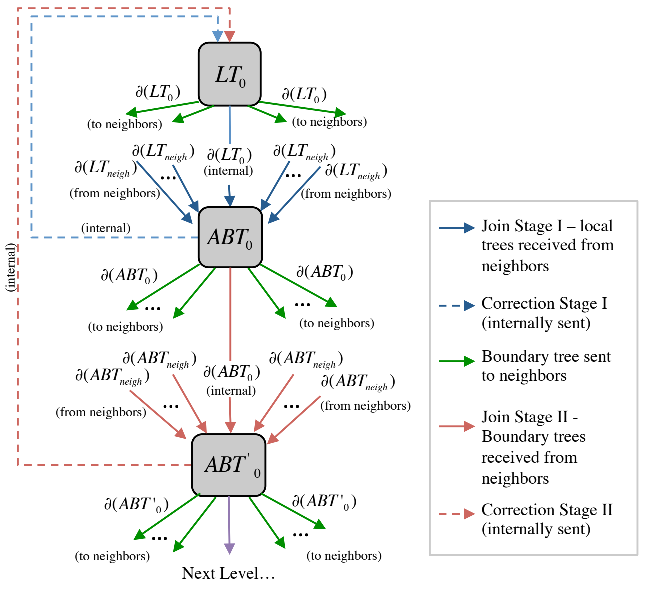

LT_0
$\partial(LT_0)$ (to neighbors)
$\partial(LT_0)$ (to neighbors)
$\partial(LT_{neigh})$ ... $\partial(LT_{neigh})$ (from neighbors)
$\partial(LT_0)$ (internal)
$\partial(LT_{neigh})$ ... $\partial(LT_{neigh})$ (from neighbors)
(internal)
ABT_0
$\partial(ABT_0)$ ... (to neighbors)
$\partial(ABT_0)$ ... (to neighbors)
$\partial(ABT_{neigh})$ ... $\partial(ABT_{neigh})$ (from neighbors)
$\partial(ABT_0)$ (internal)
$\partial(ABT_{neigh})$ ... $\partial(ABT_{neigh})$ (from neighbors)
(internal)
ABT'_0
$\partial(ABT'_0)$ ... (to neighbors)
$\partial(ABT'_0)$ ... (to neighbors)
Next Level...
Join Stage I – local trees received from neighbors
Correction Stage I (internally sent)
Boundary tree sent to neighbors
Join Stage II - Boundary trees received from neighbors
Correction Stage II (internally sent)

Fig. 8. The data flow for a single process which is replicated on all processes. The algorithm can be terminated at any level with guaranteed features of a certain size.

shown in Fig. 8 and after the initial local compute step consists of alternating boundary joins and local corrections.

This techniques has two significant advantages: First, all nodes are participating in the computation at all times; and Second, each local tree is correct for all features smaller than a given bounding box. More specifically, given a $k \times k \times k$ region of local blocks all features within $(k/2+1) \times (k/2+1) \times (k/2+1)$ blocks are guaranteed to be correctly resolved. Note that these are local blocks, not grid points of the simulation mesh. For example, a two-round region growing with $9 \times 9 \times 9$ regions of influence in the full scale HCCI dataset resolves all features within a $5 \times 5 \times 5$ block corresponding to a bounding box of $100 \times 100 \times 100$ grid points. As a result, if we know an expected feature size or a realistic upper bound, limiting the region growing provides an algorithm that strictly limits the computation and communication independent of weak scaling while guaranteeing that all features below a given size are correctly captured. As will be discussed in Section V, in practice even one or two rounds may resolve all features of interest. Finally, similar to the reduction based algorithm, the results of a limited number of joins can always be integrated into a globally correct result after the fact, at the cost of not computing statistics for data that is not also saved.

One potential problem with the region growing as described above is that many regions of influence will quickly touch the global boundary, as shown in Fig. 9(b). More importantly, neighbors needed to complete a region may lie "outside" the boundary. At the same time, neighbors which are closer in the same direction, in principle, contain the needed information, but their boundaries do not conform (Fig. 9(c)). The solution is to assume a periodic domain in which case all necessary neighbors exit (Fig. 9(d)). Interestingly, if the simulation is inherently non-periodic the local and boundary trees of regions spanning the periodic boundary will simply not share any nodes and thus, can be maintained naturally as two seperate entities. This strategy also balances the computation load better as boundary blocks, which in traditional schemes are often under-loaded, now contribute equally. However, care must be taken once the regions of influence of two conforming neighbors overlap around the periodic boundary, i.e. joining the blue region of Fig. 9(c) with its neighbor. In this case parts of the boundary will conform while others will not. To address this issue nodes of the corresponding boundary tree can be filtered to include only those on shared boudaries. However, since the region growing is specifcally designed to be local we currently do not support this mode of computation.

V. Results

To validate our approach and demonstrate its effectiveness we have applied it to two different large scale combustion simulations, originally generated by S3D, as well as to a commonly available medical dataset to compare with previous techniques. 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 characteristically spanning at least five 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 or their equivalents for lower core counts. Due to the large I/O overheads data is currently saved only every 500th time step, and this frequency expected to decrease further in the future. Analyzing the same type of extinction regions in very similar data, Mascarenhas et al. [18] have already shown that such snapshots are insufficient for reliably tracking features. Instead, here we show how applying the same analysis in-situ can be done every 50th time step – increasing the effective frequency tenfold – while adding less than one percent to the overall runtime of the simulation.

Datasets and Computing Environment. We use three 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 [19]

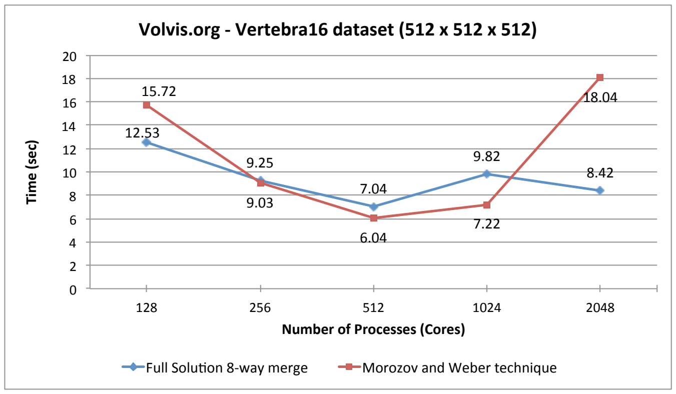
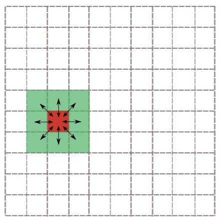


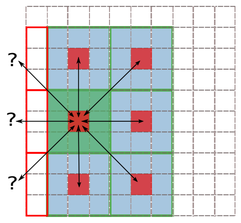
Fig. 12. Comparison with the distributed merge trees of [19]. Even though our techniques computes the full segmentation in addition to the merge tree at a significant additional cost we are on par with their fastest results and significantly outperform their technique at larger core counts.

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 twice to form a $1120 \times 560 \times 560$ volume, to conduct weak scaling study. 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.

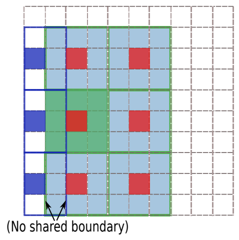
For our tests, we use both the Hopper system at the National Energy Research Scientific Computing Center (NERSC) and Titan at the OLCF. Hopper is a peta-flop Cray XE6 system consisting of 6,384 nodes each with 2 twelve-core AMD MagnyCours 2.1Ghz processors resulting in a total of 153,216 compute cores. 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. The Hopper system was primarily used for the comparisons.

Comparison with Previous Work. As mentioned above, ours is the first large-scale, parallel approach to compute segmented merge trees. However, to provide some context we compare our results with the distributed merge trees of [19]. Note that one of the main innovations of that approach has been a sparse representation of the local trees which significantly reduces their size and thus speeds up all computations. Unfortunately, the missing information is strictly required for the segmentation and thus these optimizations are not applicable when extracting spatial features rather than simply the tree itself. Therefore, our approach must deal with significantly larger trees as well as the additional computation to maintain the segmentation. Nevertheless, as shown in Fig. 12, our global reduction algorithm is on par for the fastest overall runtime and significantly outperforms [19] for larger core counts, indicating a better scaling behavior.

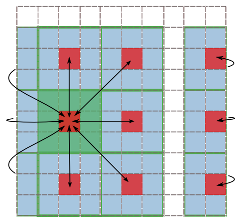

(a)



(b)



(c)


(d)

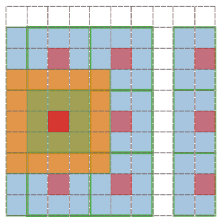
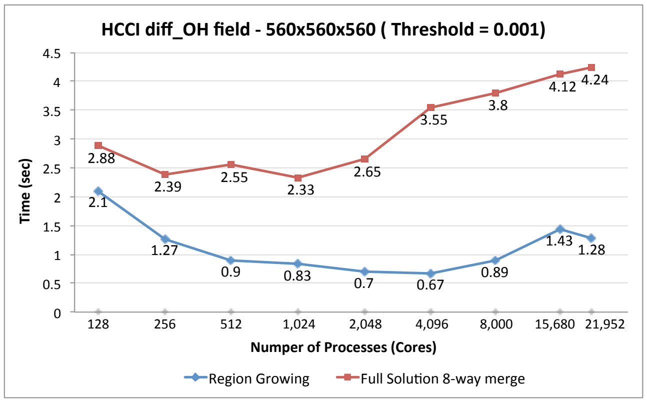
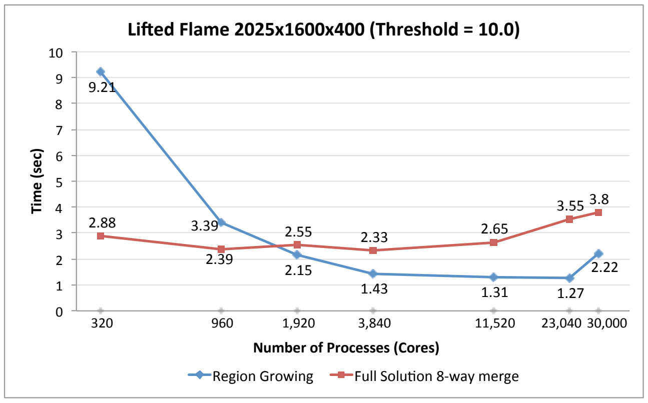

(e)

Fig. 9. The handling of region growing near the global domain boundary. In (a), the first level of joining is shown where boundaries from immediate neighbors are joined to obtain the region shown in green. In (b), the second level of joining is done where boundaries are exchanged with blocks that are at a distance of 3^1. Notice how the neighbors on the left are not present as we have reached the global domain boundary. In (c), we show that the blocks at the edge of the domain are not useful as they do not share a boundary. In (d), the solution to the problem is shown by considering the global domain boundary to be periodic and overloading the boundary trees from one side onto the other side. In (e), we show the guaranteed feature size of 5×5 within the grown boundary of 9×9.



(a)



(b)

Fig. 10. Time taken by the analysis on Titan for (a) the HCCI data set and (b)the Lifted Flame data set with various process counts.

In-situ Feature Extraction. The ultimate goal is to support in-situ feature extraction and thus we have tested our approach using two known use cases in large scale turbulent combustion. In the HCCI case we choose the diff-OH field at a time step just as most of the flame kernels start igniting. This is one of the most interesting times in the simulation with the largest number of features and thus expected the most complex analysis task. Features are define as regions of high diff-OH and form mainly flat sheets that for lower thresholds quickly become dominated by a single large structure as shown in Fig. 13. After consulting with domain experts we have chosen a conservative cut-off of 0.001 to ignore uninteresting portions of the tree.

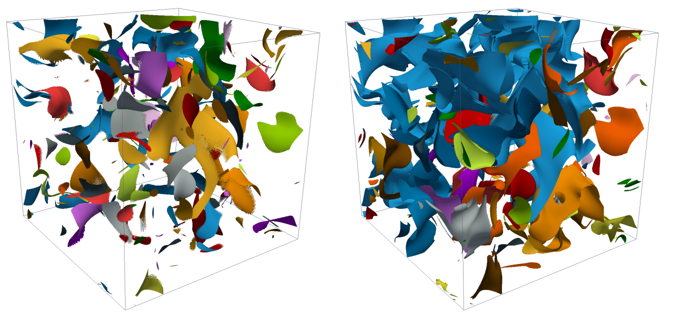


Fig. 13. Features extracted from the HCCI simulation for a diff-OH threshold of 0.12 (left) and 0.09 (right).

For the lifted ethylene flame we extract extinction regions indicated as regions of high scalar dissipation as done in [18] for a similar data set. The scalar dissipation field is know to have a high dynamic range and thus instead of a fixed threshold we use the relevance metric of [18] to set local thresholds. This extracts significantly more and better isolated features across all scales As shown in Fig. 1 the flame contains a large number of flat "pancake" like structures packed around the two flame sheets. Similar to the HCCI we have chosen a conservative threshold of 10 that nevertheless ignores unnecessary portions of the domain.

We first applied the reduction based global algorithm using an 8-way reduction that we experimentally found produced the fastest results for both datasets. Fig. 10 shows the running times for increasing core counts up to the original simulation configuration. In both use cases the absolute running times are very low and often at or even below the variance caused by the system [44] for different runs. Nevertheless, especially for the more space-filling features of the HCCI data, running times increase for larger core counts. As the core counts increase more work is shifted away from the data parallel local computation and local corrections and more towards the reduction which is know to scale poorly. Nevertheless, the running times at full scale, assuming an analysis every 50th timestep (a tenfold increase from current state-of-the-art), would require only 0.7% (lifted flame) and 2.5% (HCCI) of the total simulation runtime even assuming S3D scales perfectly

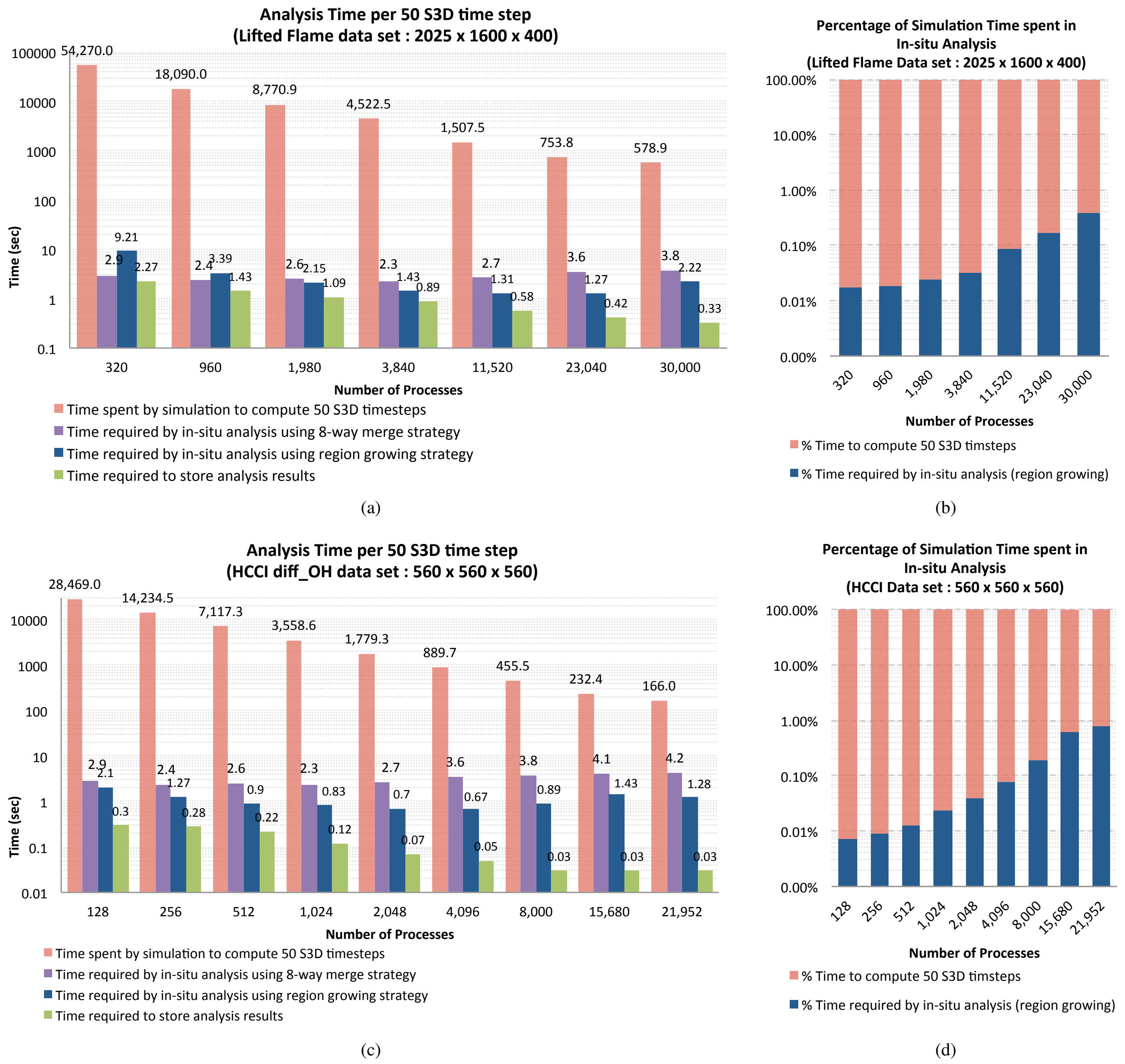


Fig. 11. Time spent by the in-situ analysis as compared to the time taken by S3D simulation to compute 50 time steps. The absolute times and the percentage overhead added by the analysis along with storing of the results for the Lifted Flame data set((a) & (b)) and the HCCI data set((c) & (d)). Notice that the time taken by the analysis along with writing the results is below 1% of the total simulation time.

(see Fig. 11).

However, as indicated by Figs. 13 and 1 most of the features of interest are spatially small and do not require a globally resolved segmentation. To verify this hypothesis we extracted the total number of features as well as features contained within various axis-aligned bounding boxes for wide range of thresholds and relevances respectively. Figs. 14 and 15 show the results and for context also the total volume classified as features as well as the volume of the largest feature. The latter is a rough indication for a reasonably threshold since in these cases scientists expect many small features and once the volume is dominated by a single large feature the parameter values are likely out of a practical range. For the HCCI dataset, regions of size $280 \times 280 \times 280$ cover virtually all features of interest for all thresholds. Even the smaller 100×100 region would likely be sufficient in practice as practical parameter ranges are above 0.12 resulting in only few missed features. Features in the lifted flame are generally smaller and even for a $52 \times 80 \times 80$ region virtually all features are correctly extracted for the entire parameter range.

Taking advantage of these results we used the region growing algorithm with a restricted number of rounds and report the results in Fig. 10. For both use cases we apply the most conservative approach and grow the biggest regions short of computing the global result. Note that since the size of the region is determined by the number of blocks in a

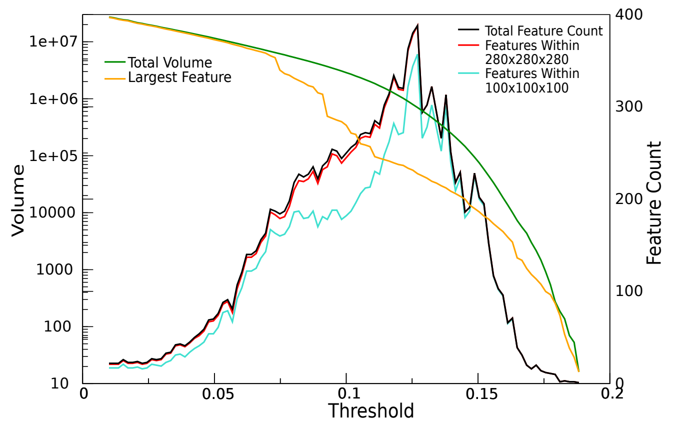


Fig. 14. Feature statistics for the HCCI diff-OH data set. Most features are captured for the $100 \times 100 \times 100$ regions and virtually all for the larger $280 \times 280 \times 280$ regions. A practically parameter is likely above 0.12 making even the smaller regions sufficient for the analysis.

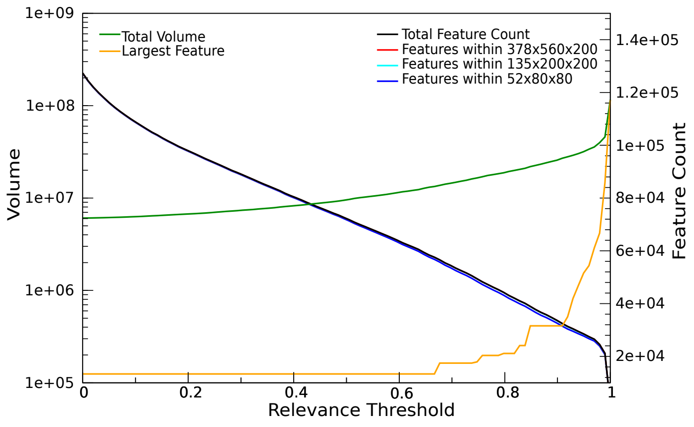


Fig. 15. Feature statistics for the Lifted Flame data set. Virtually all features are captured within $52 \times 80 \times 80$ regions corresponding to only two blocks of the original decomposition.

decomposition and block counts increase by a factor of three the effective bounding box sizes change for different configurations. As a result the running time across different core counts are not perfectly comparable. Tables I and II show the domain decompositions, regions of influence, and guaranteed feature sizes for the HCCI and lifted use case respectively. Note that at full scale both use cases resolve virtually all features of interest at significantly reduced runtimes. Furthermore, as discussed above, the sizes, especially for the lifted flame, could be further reduced resulting in even bigger time savings.

To better evaluate the region growing algorithm at equal region sizes and feature densities we use the HCCI data to construct a weak scaling study. Using various subsets as well as a twice repeated version of the data we use the original domain decomposition of $20 \times 20 \times 20$ grid points at different

Cores	Domain Decomposition (blocks)	Region Grown (blocks)	Feature Size (grid points)
128	$8 \times 4 \times 4$	$3 \times 3 \times 3$	$70 \times 140 \times 140$
256	$8 \times 8 \times 4$	$3 \times 3 \times 3$	$70 \times 70 \times 140$
512	$8 \times 8 \times 8$	$3 \times 3 \times 3$	$70 \times 70 \times 70$
1024	$16 \times 8 \times 8$	$9 \times 3 \times 3$	$175 \times 70 \times 70$
2048	$16 \times 16 \times 8$	$9 \times 9 \times 3$	$175 \times 175 \times 70$
4096	$16 \times 16 \times 16$	$9 \times 9 \times 9$	$175 \times 175 \times 175$
8000	$20 \times 20 \times 20$	$9 \times 9 \times 9$	$140 \times 140 \times 140$
15680	$28 \times 28 \times 20$	$27 \times 27 \times 9$	$280 \times 280 \times 140$
21952	$28 \times 28 \times 28$	$27 \times 27 \times 27$	$280 \times 280 \times 280$

TABLE I. THE FEATURE SIZES CAPTURED AT VARIOUS CORE COUNTS FOR HCCI SIMULATION

Cores	Domain Decomposition (blocks)	Region Grown (blocks)	Feature Size (grid points)
320	$5 \times 8 \times 8$	$3 \times 3 \times 3$	$405 \times 200 \times 50$
960	$15 \times 8 \times 8$	$9 \times 3 \times 3$	$675 \times 200 \times 50$
1920	$15 \times 16 \times 8$	$9 \times 9 \times 3$	$675 \times 500 \times 70$
3840	$15 \times 16 \times 16$	$9 \times 9 \times 9$	$675 \times 500 \times 125$
11520	$45 \times 16 \times 16$	$27 \times 9 \times 9$	$630 \times 500 \times 125$
23040	$45 \times 32 \times 16$	$27 \times 27 \times 9$	$630 \times 700 \times 125$
30000	$75 \times 40 \times 10$	$27 \times 27 \times 9$	$378 \times 560 \times 200$

TABLE II. THE FEATURE SIZES CAPTURED AT VARIOUS CORE COUNTS FOR LIFTED FLAME SIMULATION

core counts. To obtain a sufficient number of experiments we use two rounds of region growing, corresponding to $9 \times 9 \times 9$ blocks and a guaranteed feature size of $100 \times 100 \times 100$. For each run we double the size of the data and the core counts to keep the load constant on each core. As shown in Fig. 16 we achieve good scaling up to almost 44K cores by which time the data has grown by a factor of 16 at twice the runtime, which also remains very low in an absolute sense. Profiling the performance shows that at larger scale the algorithm is communication bound while the computation by construction remains virtually identical as each processor is doing the same amount of work (bar some data dependencies) at all scales. Therefore, the increasing diameter of the corresponding network as well as interference from other jobs on the machine, as shown by Bhatele et al. [44] is likely responsible for the increase in time.

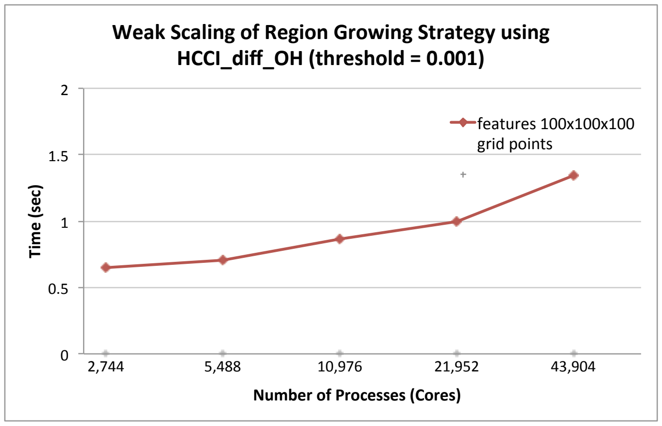


Fig. 16. Weak scaling results for the HCCI data, extracting all features contained with $100 \times 100 \times 100$ grid points.

VI. Conclusion

This paper introduces the first scalable algorithm to compute segmented merge trees for large-scale data. In particular, we demonstrate that applying sophisticated data analysis techniques in-situ at frequencies significantly higher than previously feasible is viable with minimal overhead to a simulation. Furthermore, we present a locally consistent, approximate algorithm with good weak scaling properties and strong correctness guarantees. By allowing frequent, in-depth analysis at low cost our approach as the potential for significant scientific impact. Going forward we plan to extent the framework to include on-the-fly statistics, shape characterizations, and or tracking. Furthermore, we plan to adjust the bulk synchronous model used here to the many-core architectures expected in future hardware.

Acknowledgment

This research was supported by the Department of Energy Office of Advanced Scientific Computing Research. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. 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. The authors wish to thank the members of the the ExaCT Center for Exascale Simulation of Combustion in Turbulence for useful discussions and support. This work is supported in part by BNSF CISE ACI-0904631, NSG IIS-1045032, NSF EFT ACI-0906379,DOE/NEUP 120341, DOE/Codesign P01180734, DOE/SciDAC DESC0007446, and CCMSC DE-NA0002375.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PROC-657419).

References

[1] M. Day, J. Bell, P.-T. Bremer, V. Pascucci, V. Beckner, and M. Lijewski, "Turbulence effects on cellular burning structures in lean premixed hydrogen flames," *Combustion and Flame*, vol. 156, pp. 1035–1045, 2009.

[2] 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.*, vol. 17, no. 12, pp. 1822–1831, 2011.

[3] W. Widanagamaachchi, C. Christensen, P.-T. Bremer, and V. Pascucci, "Interactive exploration of large-scale time-varying data using dynamic tracking graphs," in *Proc. IEEE Symposium Large-Scale Data Analysis and Visualization*, 2012.

[4] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *Proceedings of the 17th International Conference on Parallel Processing - Volume Part I*, ser. Euro-Par'11, 2011, pp. 366–379.

[5] H. Abbasi, J. Lofstead, F. Zheng, S. Klasky, K. Schwan, and M. Wolf, "Extending i/o through high performance data services," in *IEEE Cluster*. IEEE International, 2009.

[6] V. Vishwanath, M. Hereld, and M. Papka, "Toward simulation-time data analysis and i/o acceleration on leadership-class systems," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, 2011, pp. 9–14.

[7] 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*, vol. 30, no. 3, pp. 45–57, 2010.

[8] 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 *Eurographics Symposium on Parallel Graphics and Visualization, EGPGV 2011, Llandudno, Wales, UK, 2011. Proceedings*, T. Kuhlen, R. Pajarola, and K. Zhou, Eds. Eurographics Association, 2011, pp. 111–120. [Online]. Available: http://dx.doi.org/10.2312/EGPGV/EGPGV11/111-120

[9] 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.

[10] 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)*, October 2011, pp. 89 –96.

[11] 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.

[12] W. Hendrix, D. Palsetia, M. Patwary, A. Agrawal, W.-K. Liao, and A. Choudhary, "A scalable algorithm for single-linkage hierarchical clustering on distributed memory architectures," in *Proceedings of 3rd IEEE Symposium on Large-Scale Data Analysis and Visualization (LDAV)*, 2013.

[13] 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*, vol. 17, no. 99, 2010.

[14] V. Pascucci and K. Cole-McLaughlin, "Parallel computation of the topology of level sets," *Algorithmica*, vol. 38, no. 1, pp. 249–268, Oct. 2003.

[15] A. Gyulassy, P.-T. Bremer, V. Pascucci, and B. Hamann, "A practical approach to Morse-Smale complex computation: Scalability and generality," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1619–1626, 2008.

[16] 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.*, vol. 17, no. 12, pp. 2088–2095, 2011.

[17] 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*, vol. 12, no. 5, pp. 1052–1060, 2006.

[18] 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 *Topological Methods in Data Analysis and Visualization*, ser. Mathematics and Visualization, V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, Eds. Springer Berlin Heidelberg, 2011, pp. 229–240.

[19] D. Morozov and G. H. Weber, "Distributed merge trees," in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013*, A. Nicolau, X. Shen, S. P. Amarasinghe, and R. Vuduc, Eds. ACM, 2013, pp. 93–102. [Online]. Available: http://dl.acm.org/citation.cfm?id=2442516

[20] 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*, vol. 2, pp. 1–31, 2009.

[21] 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.

[22] 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.

[23] 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)*, November 2011, pp. 1 –11.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] B. Lorendeau, Y. Fournier, and A. Ribes, "In-situ visualization in fluid mechanics using catalyst: A case study for code saturne," in *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, Oct 2013, pp. 53–57.

[29] M. Dorier, R. Sisneros, T. Peterka, G. Antoniu, and D. Semeraro, "Damaris/viz: A nonintrusive, adaptable and user-friendly in situ visualization framework," in *Large-Scale Data Analysis and Visualization (LDAV), 2013 IEEE Symposium on*, Oct 2013, pp. 67–75.

[30] F. Zheng, H. Yu, C. Hantas, M. Wolf, G. Eisenhauer, K. Schwan, H. Abbasi, and S. Klasky, "Goldrush: Resource efficient in situ scientific data analytics using fine-grained interference aware execution," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 78:1–78:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503279

[31] 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*, vol. 222, pp. 847–849, 1946.

[32] 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*, ACM. New York, NY, USA: ACM Press, Jan. 2000, pp. 918–926.

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

[34] H. Carr, J. Snoeyink, and M. van de Panne, "Simplifying flexible isosurfaces using local geometric measures," in *IEEE Visualization '04*. IEEE Computer Society, 2004, pp. 497–504.

[35] 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*, vol. 12, no. 5, pp. 1053–1060, Sep. 2006.

[36] 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*, vol. 13, no. 6, pp. 1432–1439, 2007.

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

[38] 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*, vol. 12, no. 4, pp. 474–484, 2006.

[39] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas, "Robust on-line computation of Reeb graphs: simplicity and speed," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[40] 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*, vol. 14, no. 6, pp. 1619–1626, 2008.

[41] 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.

[42] J. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen, "Combining in-situ and in-transit processing to enable extreme-scale scientific analysis," in *Proc. ACM/IEEE Conference on Supercomputing (SC12)*, 2012.

[43] H. Carr, J. Snoeyink, and U. Axen, "Computing contour trees in all dimensions," *Comput. Geom. Theory Appl.*, vol. 24, no. 3, pp. 75–94, 2003.

[44] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There goes the neighborhood: performance degradation due to nearby jobs," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. IEEE Computer Society, Nov. 2013 (to appear), lLNL-CONF-635776.