Notes on the Distributed Computation of Merge Trees on *CW*-complexes

Aaditya G. Landge, Peer-Timo Bremer, Attila Gyulassy, and Valerio Pascucci

Abstract Merge trees are topological structures that record changes in super-level set topology of a scalar function. They encapsulate a wide range of threshold based features which can be extracted for analysis and visualization. Several distributed and parallel algorithms for computing merge trees have been proposed in the past, but they are restricted to simplicial complexes or regular grids. In this paper, we present an algorithm for the distributed computation of merge trees on *CW*-complexes. The conditions on the *CW*-complex required for the computation of the merge tree are discussed along side a proof of correctness.

1 Introduction

Analysis and visualization are crucial components in gaining scientific insight from scientific simulations. In this regard, topological techniques have been successful at extracting features of interest from scientific datasets [2, 8, 15]. Topological structures, such as merge trees, are combinatorial in nature and encode level-set based features of a scalar function. They enable threshold-based feature extraction and can be represented compactly, making them suitable for post process exploratory analysis. The continuous increase in computational resources available to scientists performing simulations of complex scientific phenomenon is leading to a corresponding increase in the size and complexity of data being generated. Concurrently, compute architectures are gradually moving towards multi-core and large scale distributed environments. In this scenario, its important to design and develop parallel topological analysis algorithms and techniques that can harness the parallelism provided by these massively parallel resources.

Preciously, several efficient serial [3,5], and streaming algorithms [**?**,2] have been presented for topological constructs. The first parallel algorithm for computing contour trees has been proposed by Pascucci et. al. [12]. Subsequently, [6, 7, 10, 11] introduced techniques to computecontour or merge trees more efficiently at scale. All the above algorithms have been developed for simplicial complexes or rectilinear 2 or 3-d grids. But there are several scientific phenomenon [4, 13] that are modelled using meshes like structured/unstructured curvilinear meshes, finite element zoo meshes [14], adaptive mesh refinement(AMR) meshes [1], etc. For these

Aaditya G. Landge, Attila Gyulassy, and Valerio Pascucci,
Scientific Computing and Imaging (SCI) Institute, University of Utah, Salt Lake City, UT, e-mail: aaditya, jediati, pascucci@sci.utah.edu

Peer-Timo Bremer
Lawrence Livermore National Laboratory, Livermore, CA, e-mail: bremer6@llnl.gov

types of meshes, its non-trivial and potentially costly to convert a mesh into a simplicial complex. However, the majority of such meshes can be represent as regular *CW*-complexes. Hence, there is a strong motivation to extend the topological analysis algorithms to handle *CW*-complexes. In this work, we present a distributed algorithm for computing the merge tree on a regular *CW*-complex. Furthermore, we present a proof of correctness for this approach, and apply our result to validate the previous algorithms.

2 Background

We briefly review some basic concepts from algebraic topology, and refer the reader to Massey [9] for further reading. Let there be a space $\mathbb{M} \subset \mathbb{R}^d$, which is represented using a a d-dimensional, finite, regular *CW*-complex, $\mathcal{K}^d$, such that a k-cell is an open k-ball, $0 \leq k \leq d$. In computational science, a continuous Morse function $F : \mathbb{M} \to \mathbb{R}$, is typically discretized by assigning values to 0-cells(vertices) and interpolating the function within every k-cell to obtain a function $f : \mathcal{K} \to \mathbb{R}$, such that each vertex is associated with a distinct function value and the interpolation scheme ensures f is C^0 on $\mathcal{K}^d$ with simple critical points. For example, in the cases when $\mathcal{K}$ is a simplicial complex or a $3d$-rectilinear grid, the interpolation schemes that could be used are linear and trilinear respectively.

Let there be a k-dimensional cell, $\alpha^k \in \mathcal{K}$, $k \leq d$. Then the closure of α, denoted as $\overline{\alpha}$, is the cell α and the limit points of α. Thus, the *boundary* of α, denoted as $\partial\alpha$, is given by $\partial\alpha = \overline{\alpha} \backslash \alpha$. If another cell, $\beta^m \in \mathcal{K}, k < m \leq d$ such that $\alpha \subset \overline{\beta}$, then α^k is the *face* of β^m, denoted as $\alpha \ll \beta$.

Definition 1. The **level set**, l_c, of f at a value $c \in \mathbb{R}$ is the set of points in the domain of f such that $l_c = f^{-1}(c)$. A connected component of the level set is called a **contour**.

Definition 2. The **super-level set**, L_c, of f is the set of points in the domain of f with value in f greater than $c \in \mathbb{R}$ and is given as $L_c = f^{-1}[c, \infty)$.

Definition 3. Given a super-level set, L_c, of f having n connected components denoted as $\{C_1, C_2, \ldots, C_n\}$, then $L_c = \cup C_i$ for $i = 1 \ldots n$ and $C_i \cap C_j = \emptyset$ $(i \neq j)$. We define an equivalence relation '$\sim$' on $\mathcal{K}$ such that two points $x, y \in \mathcal{K}$ are related if $f(x) = f(y) = c$ and $x, y \in C_i$ i.e. belong to the same level-set and the same super-level component of L_c. Then the quotient space, $\mathcal{K}/\sim$, is the **merge tree** of f on $\mathcal{K}$, denoted as $MT(\mathcal{K})$. The many-to-one map, $\phi : \mathcal{K} \to \mathcal{K}/\sim$, maps points from $\mathcal{K}$ onto a point in the merge tree.

Intuitively, the merge tree encodes the evolution of the frontiers of connected components of the super-level set of f on $\mathcal{K}$. 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* - leaves in the tree, *saddles* - interior nodes and the *global minimum* - the root of the tree respectively. Sometimes, the merge tree is augmented with valence-2 nodes, which are non-critical nodes that do

not correspond to any change in the super-level set topology. We call these nodes *regular* nodes. In this paper, we assume without loss of generality that $\mathcal{K}$ is simply-connected. In the case when $\mathcal{K}$ is not connected, we would obtain a forest of merge trees where each tree corresponds to a connected component of the domain.

3 Computing Merge Trees on CW-complexes

Our technique is based on the divide and conquer strategy where a merge tree is computed for each partition of the domain. These trees are then joined to form the merge tree of the domain. In this section, we first describe the domain decomposition used by the divide and conquer strategy. The strategy is then expressed as a recursive algorithm followed by the necessary modifications to adapt it to a distributed setting.

3.1 Domain Decomposition

In order to apply a divide and conquer strategy we partition the domain $\mathcal{K}$ into a finite number of *patches*.

Definition 4. A **patch**, P, is a d-dimensional sub-complex of $\mathcal{K}$, where P is composed of a set of d-cells along with their boundaries, i.e. let $P \subseteq \mathcal{K}$ such that if $\alpha^d \in P$, then $\beta \in P$ iff $\beta \subset \overline{\alpha^d}$.

Definition 5. The **patch-boundary**, ∂P, of P is the intersection of P with the closure of $\mathcal{K}\backslash P$. Thus $\partial P = (\overline{\mathcal{K}\backslash P}) \cap P$.

Definition 6. The **domain-decomposition** of $\mathcal{K}$ is given as a union of finite number of patches given as $\mathcal{K} = \cup P_i, 0 < i \leq n$ such that for any two patches $P_i, P_j \in \mathcal{K}$, $(P_i \backslash \partial P_i) \cap (P_j \backslash \partial P_j) = \emptyset$, $i \neq j$.

Definition 7. Let there be patches, $P, Q \in \mathcal{K}$. Then P and Q are **neighbors** if $P \cap Q \neq \emptyset$. The **boundary components**, $\partial_i P$, of P are the intersections of P with each of its neighbors, Q_i. Thus, the boundary components of P are $\partial_i P = P \cap Q_i$. We denote the set of all $\partial_i P$ as $\widehat{\partial} P$. Since P and Q_i are neighbors, each $P \cap Q_i$ is also a boundary component of each neighbor Q_i.

Definition 8. A **patch hierarchy** consists of *levels*, $h = 0, \ldots, h_{max}$, where $h = 0$ is the finest level and $h = h_{max}$ is the coarsest level. Each *level* a complex of patches such that patch at level h, denoted as P^h is the union of patches at level $h-1$. Thus, $P^h = \cup P_i^{h-1}, 0 < i \leq n$. At the finest level, $h = 0$, a patch, P^0 is a d-dimensional cell, $\alpha^d \in \mathcal{K}^d$ along with its *boundary*, $\partial \alpha^d$. Thus, $P^0 = \alpha^d \cup \partial \alpha^d = \overline{\alpha^d}$.

One may consider the hierarchy as a tree where each node is a patch. The leaves are patches composed of a single d-dimensional cell and the interior nodes are patches composed of the union of the children. We do not enforce any restriction on the neighborhoods of the children.

3.2 Joining Merge Trees

Given two neighboring patches, $P, Q \in \mathcal{K}$, we can compute their respective merge trees $MT(P)$ and $MT(Q)$ and glue them in a specific way to obtain the merge tree of

the union, $P \cup Q$. We call this the *join* operation. But to perform the join, we have to preserve the connectivity of the super-level set components that expand into the neighboring patch. This information is provided by points on the shared boundary $P \cap Q$ and one can achieve the join by adding all these points into the merge trees of the patches as noncritical, *regular* nodes. These can then be used in the join to form the merge tree of $P \cup Q$. But as there are infinitely many points on the boundary this approach is not feasible. Instead, we can restrict the number of noncritical points from the boundary by only adding the *boundary maxima* as regular nodes to the merge trees.

Definition 9. The maxima of f, when f is restricted to every boundary component of a patch, P, are known as the **boundary maxima** of P.

Once we have the merge tree along with the boundary maxima, we can now *join* trees from neighboring patches along these nodes. Before we look at the details of the join, let us define the inputs to this operator.

Definition 10. Let $P \in \mathcal{K}$ be a patch and let $\widehat{\partial} S$ be a set of boundary components. The **augmented merge tree**, $AMT(P, \widehat{\partial} S)$, is the merge tree of f restricted to the patch, P, augmented with the boundary maxima of all boundary components, $\widehat{\partial} S$.

The $JoinMT()$ routine described in Algo. 1 performs this operation on a set of merge trees. It assembles the resulting tree from the arcs and nodes of input trees by introducing each arc, along with its end nodes, in a descending order based on the function value of the lower node. Each time an arc, (u,v), bounded by nodes u and v, $f(u) > f(v)$, is introduced, it gives rise to one of the following cases in the tree being constructed, T:

1. $u, v \notin T$ then arc (u,v) does not attach to any of the existing arcs but gets added as a disjoint arc in T – this represents the creation of a new super-level set component
2. $u \in T$, $v \notin T$ and u has no descendants, then we attach (u,v) to u as a descendant – this represents the growth of an existing super-level set component by addition of the region represented by (u,v)
3. $u \in T$, $v \notin T$ and u has descendants with the lowest descendant being u', then we attach v as a descendant of u' by introducing an arc (u',v) – this means that the super-level set component containing u has already grown till u', so grow it further till v.
4. $v \in T$, $u \notin T$, then we attach (u,v) to v making v a saddle – this represents creation of a super-level set component that merges with another super-level set component at v. Note that since we are adding arcs in the descending order based on the function value of the lower node, v will not have any descendants present in T.
5. $u, v \in T$ and v is not a descendant of u, then we add the arc between the lowest descendant of u, given by u', and v – this represents merging of super-level components

6. $u, v \in T$ and v is a descendant of u, then we discard the arc (u,v) – this mean that the region of the super-level set component represented by (u,v) is already present in the domain.

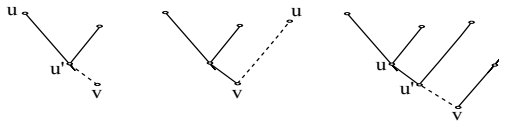

Fig. 1 Examples for case 3 (left), case 4 (middle), and case 5 (left) from the above description.

The above discussed process can be achieved by a union-find like traversal of the sorted nodes as seen in Algo. 1. The sorting of the nodes is performed in linear time as the input AMTs would have their nodes in sorted order. The $Find(u, AMT)$ routine returns the lowest descendant of u in AMT and adds u if it is not present in AMT. This is implemented as a union-find data structure and has amortized constant running time. The number of edges for a given node is constant and hence the Algo. 1 has a linear run time complexity given by the number of nodes in the input AMTs.

Algorithm 1: Join(AMT(...))

```
Data: Array of AMTs for individual patches
Result: AMT of the union of patches
AMT = [];
Nodes[] = Sorted list of the union of vertices from input AMTs in decreasing order of
function value;
for All nodes v in Nodes do
    for Arcs (u,v) in input AMTs do
        u' = Find(u,AMT);        // If u not in AMT, add and return u
        v' = Find(v,AMT);        // If v not in AMT, add and return v
        if u' ≠ v' then
            AddArc(u',v');
return AMT;
```

Definition 11. Given two augmented merge trees, $AMT(P, \widehat{\partial}P)$ and $AMT(Q, \widehat{\partial}Q)$, the **join** operator, $(+)$, joins them along the boundary maxima of $P \cap Q$ to form $AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$.

Theorem 1. $AMT(P, \widehat{\partial}P) + AMT(Q, \widehat{\partial}Q) = AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$.

Proof. From Def. 3 of a merge tree, a point $x \in P \cup Q$ is mapped to a point on $AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$ under the map ϕ. If x lies on the arc (u,v), where $f(u) > f(v)$, we say that x has the label u. We now have to show that the join operator on $AMT(P, \widehat{\partial}P)$ and $AMT(Q, \widehat{\partial}Q)$ produces the same set of labels for points in $P \cup Q$ as produced by $AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$. If the points have the correct labels, we have the correct connectivity of arcs under the join operation.

Let us assume that the correct label for a point $x \in P \cup Q$ is u. Then x lies on an arc $(u,v) \in AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$. Now, there are two cases,

1. $x, u \in P$. In this case, u must be the label of x in $AMT(P, \widehat{\partial}P)$. Let us assume that the join operation assigns a label w to x. This is possible only if $w \in Q \backslash P$. Now, if w is to be the label of x, we should have $f(u) > f(w) > f(x)$ and they must lie on the same super-level set component. This implies that the u and w must have a common ancestor. In this case, the join operation ensures that u, w and x lie on the same path to the root. Thus, w should be the label of x, but that contradicts our assumption that u is the correct label, hence, our assumption that the join operation assigns w as the label is false.
2. $x \in P$ and $u \in Q \backslash P$. Then before the join lets assume that x had a label w in $AMT(P, \widehat{\partial}P)$. Now, if u is the label of x, then $f(w) > f(u) > f(x)$ and u and x must be on the same super-level set component. But as x has a label w in $AMT(P, \widehat{\partial}P)$ implies that x and w are also on the same super-level set component. This implies that w and u have a common ancestor. This is possible only if all them lie on the same path from the ancestor to the root. If u, w are on the the same path the algorithm will assign the label u to x since $f(w) > f(u)$ thereby assigning the correct label.

Thus, the join operator always maintains the correct labels and hence generates the correct $AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$. □

The resulting tree from the join contains valence two nodes from $P \cap Q$ that are no longer required. On the other hand, we need to retain the boundary maxima of the boundary components of the union of patches, $P \cup Q$. The boundary maxima of the components of $\partial(P \cup Q)$ are already present in the joined tree. The following lemma proves this claim.

Lemma 1. *Given two neighboring patches $P, Q \in \mathcal{K}$ and their augmented merge trees $AMT(P, \widehat{\partial}P)$ and $AMT(Q, \widehat{\partial}Q)$, then the join of these trees contains the boundary maxima of $\widehat{\partial}(P \cup Q)$.*

Proof. $AMT(P, \widehat{\partial}P)$ and $AMT(Q, \widehat{\partial}Q)$ contain the boundary maxima of $\widehat{\partial}P$ and $\widehat{\partial}Q$ respectively and the join operation does not remove any nodes from the participating trees while creating the resulting tree. Thus, $AMT(P, \widehat{\partial}P) + AMT(Q, \widehat{\partial}Q) = AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$ contains the boundary maxima of $\widehat{\partial}(P \cup Q)$. □

3.3 Obtaining the Boundary Maxima and Pruning the Merge Tree

Definition 12. The $\mathcal{B}_{max}(\widehat{\partial}P)$ operator takes a patch and returns the boundary maxima for each boundary component, $\widehat{\partial}P$, of P.

The function $GetBoundaryMax(P, h)$ returns the list of boundary maxima for the patch P at level h. This call takes help of the $BuildMT()$ routine that generates a merge tree for a patch. We shall define this routine in detail later in this section. As the merge tree by definition preserves all the maxima of a function on a given domain, we obtain the boundary maxima by computing the merge tree of each boundary component of P and extract the maxima from the respective trees. Since, we

do not know the function interpolation scheme on $\mathcal{K}$, we take help of an oracle as defined below to obtain the merge tree of a cell in the *CW*-complex.

Definition 13. Let $\alpha^k \in \mathcal{K}^d$, $0 \leq k \leq d$, be a k-dimensional cell. The **oracle**, given as $OracleMT(f, \alpha)$, returns the merge tree, denoted as $MT(\alpha)$ of the cell and its boundary i.e. $\alpha \cup \partial\alpha$.

A similar approach of using an oracle was taken in [12], but the authors did not include the boundary maxima of the cell in their computation which can result in an incorrect join.

As we have seen in Theorem 1 that the boundary maxima are required in order to perform the correct join operation so we must include them in the merge tree returned by the oracle. Since, $\mathcal{K}$ is a regular *CW*-complex, the boundary components, $\widehat{\partial}\alpha$, are $(d-k)$-dimension sub-complexes where $1 \leq k \leq d$. To obtain the boundary maxima of $\widehat{\partial}\alpha$, we make use of the oracle to give us the merge trees of individual cells in $\widehat{\partial}\alpha$. We can then extract the maxima from these trees.

The $GetMaxFromMT()$ is a trivial routine that returns the maxima nodes from a merge tree. Algo. 2 describes the $GetBoundaryMax(P,h)$ function. As long as the $BuildMT()$ and the $OracleMT$ generate the correct merge tree this routine shall identify the correct boundary maxima.

Algorithm 2: GetBoundaryMax(P, h)

```
Data: Patch, P, and level, h, in the hierarchy
Result: Array with references to maxima, max[]
for all patch-boundary components ∂_i P^h ∈ P do
    if h > h_0 then
        MT = BuildMT(∂_i P^h, h);
        max[...] = GetMaxFromMT(MT);          // Returns maxima from MT
    else
        for all cells α_j in ∂_i P^h do
            MT = OracleMT(α_j);
            max[...] = GetMaxFromMT(MT);      // Returns maxima from MT
return max[...];
```

The max need to be relabeled in $AMT(P \cup Q, \widehat{\partial}P \cup \widehat{\partial}Q)$. The following operator performs this operation.

Definition 14. The **mark boundary max** operator, $\mathcal{M}(AMT(P,\ \widehat{\partial}S),\ \mathcal{B}_{max}(\widehat{\partial}R))$, such that $\widehat{\partial}R \subseteq \widehat{\partial}S$, returns the $AMT(P,\ \widehat{\partial}R)$, with the boundary max of all components of $\widehat{\partial}R$ marked as boundary.

The $MarkBoundary()$ routine performs the above operation. It first traverses the tree and unmarks all the nodes. It then finds the boundary maxima in the tree and marks them as boundary. This results in a tree that has only the boundary maxima marked as boundary.

Finally, we need to remove the redundant valence two or regular nodes that are not boundary. These are no longer required as they are not critical and do not lie on the boundary. We remove them from tree using the following operator.

Definition 15. The **prune** operator, $\mathcal{P}(AMT)$, removes the regular nodes that are not on the boundary from the AMT.

This is a trivial operation and can be performed by simply traversing the tree and deleting the regular nodes that are not boundary.

3.4 Recursive Computation

Now that the above operations have been defined, the merge tree for the entire domain $\mathcal{K}$ can be built in a recursive fashion. We compute the merge tree for every patch at the finest level in the patch hierarchy, $h = 0$, and *join* them using the join operator, find the boundary maxima of the union of the patches using the $\mathcal{B}_{max}$ operator, mark them using the $\mathcal{M}$ operator and finally prune the tree using the $\mathcal{P}$ operator to form the merge tree of a patch at the next coarser level, $h = 1$. We perform this operation recursively till we have obtained the merge tree of the final level, $h = h_{max}$. The recursive solution is given in the following theorem.

Theorem 2. *Given a patch, $P^h \in \mathcal{K}^d$ at level h, the $AMT(P^h, \widehat{\partial} P^h)$ is given by,*
$AMT(P^h, \widehat{\partial} P^h) = \mathcal{P}\Big\{\mathcal{M}\Big[\Big(\sum_{i=0}^{n} AMT(P_i^{h-1}, \widehat{\partial} P_i^{h-1})\Big), \mathcal{B}_{max}(\widehat{\partial} P^h)\Big]\Big\}$, *where*
$\sum_{i=0}^{n} AMT(P_i^{h-1}, \widehat{\partial} P_i^{h-1}) = AMT(P_0^{h-1}, \widehat{\partial} P_0^{h-1}) + \cdots + AMT(P_n^{h-1}, \widehat{\partial} P_n^{h-1})$
i.e. the join of all the merge trees of patches at level, $h-1$, within the patch P^h.

Proof. This can be proved using induction. At $h = 0$, the patch, $P^0 = \alpha^d \cup \partial \alpha^d$, is a d-cell and its boundary. We make use of the oracle to obtain $MT(\alpha)$. By using the $\mathcal{B}_{max}$ operator on $\widehat{\partial}\alpha$ we can obtain the boundary maxima, which can be added to $MT(\alpha)$ to give us $AMT(P^0, \widehat{\partial} P^0)$.

At $h = k$, let $P^k = \cup P_i^{k-1}$, $0 \le i \le n$. Lets assume that

$$AMT(P^k, \widehat{\partial} P^k) = \mathcal{P}\Big\{\mathcal{M}\Big[\Big(\sum_{i=0}^{n} AMT(P_i^{k-1}, \widehat{\partial} P_i^{k-1})\Big), \mathcal{B}_{max}(\widehat{\partial} P^k)\Big]\Big\} \tag{1}$$

At $h = k+1$, let $P^{k+1} = \cup P_j^k$, $0 \le j \le m$. Now, since we can compute $AMT(P_j^k, \widehat{\partial} P_j^k)$ using Eq.(1) and join them to get,

$$\sum_{j=0}^{n} AMT(P_j^k, \widehat{\partial} P_j^k) = AMT(\cup P_j^k, \cup \widehat{\partial} P_j^k) = AMT(P^{k+1}, \cup \widehat{\partial} P_j^k) \tag{2}$$

From (2), we have obtained the AMT of the union of patches, $\cup P_i^h = P^{k+1}$, that contains the boundary maxima of the union of the boundaries, $\cup(\widehat{\partial} P_i^k)$. But we need The boundary maxima of components of the boundary of the union i.e. $\widehat{\partial}(\cup P_j^k) = \widehat{\partial}(P^{k+1})$, which need to be marked in $AMT(P^{k+1}, \cup \widehat{\partial} P_j^k)$. The boundary max is obtained from $\mathcal{B}_{max}(\widehat{\partial} P^k)$. These can then be marked by using the $\mathcal{M}$ operator followed by a prune giving,

$$\mathcal{P}\Big\{\mathcal{M}\Big[AMT(P^k,\, \cup\widehat{\partial}P_i^{k-1}),\, \mathcal{B}_{max}(\widehat{\partial}P^k)\Big]\Big\} = AMT(P^{k+1},\, \widehat{\partial}P^{k+1}) \quad (3)$$

$$Thus,\, AMT(P^{k+1},\, \widehat{\partial}P^{k+1}) = \mathcal{P}\Big\{\mathcal{M}\Big[\Big(\sum_{j=0}^{n} AMT(P_j^k,\, \widehat{\partial}P_j^k)\Big), \mathcal{B}_{max}(\widehat{\partial}P^{k+1})\Big]\Big\} \quad (4)$$

□

The function $BuildMT(P,h)$, described in Algo. 3, where P is a patch or patch-boundary component and h is the level in the hierarchy shows the recursive construction. The merge tree for all patches within a higher level patch are computed and joined. The boundary maxima are computed and marked followed by pruning the tree. This is done recursively. Note that at the base level of the recursion the patch is a single d-cell. The oracle returns the merge tree of the cell but we still need to explicitly add the boundary maxima for the cell. Hence, we make the extra call to $GetBoundaryMax()$ and $MarkBoundary()$ within the *else* after we have invoked the oracle. At the end of the recursive computation, the $BuildMT()$ routine computes the $AMT(\mathcal{K},\, \widehat{\partial}\mathcal{K})$. We can easily delete the boundary nodes to obtain the $MT(\mathcal{K})$.

Algorithm 3: BuildMT(P, h)

```
Data: Patch, P, and level, h, in the hierarchy
Result: MT(P)
for all patches P_i^{h-1} ∈ P do
    if h > 0 then
        MT[i] = BuildMT(P_i^{h-1}, h − 1);
    else
        MT[i] = OracleMT(P_i^h);
        max[...] = GetBoundaryMax(P_i^h, h);
        MarkBoundary(MT[i], max[...]);
if h > 0 then
    MT = Join(MT[...]);
max[] = GetBoundaryMax(P, h);
MarkBoundary(MT, max[...]);
Prune(MT);
return MT(P);
```

3.5 Distributed Computation

In the above section, we have shown that we can compute the merge tree of a domain $\mathcal{K}$ by decomposing into a hierarchy of patch levels and recursively computing the tree on each level. In the distributed scenario, we unroll the recursion so as to perform the computation of every patch on to an individual compute resource. This results into a patch containing multiple d-cells being allocated to every compute resource. The information between patches is exchanged using a message passing interface. The merge tree of $\mathcal{K}$ can be computed by joining the merge trees from the distributed patches in a successive join hierarchy corresponding to the patch hierarchy, until the merge tree of the whole domain, denoted as the *global tree*, is computed.

The computation of the *global tree*, by simply unrolling the recursive algorithm is not an efficient distributed solution as it involves communicating entire intermediate merge trees by every patch incurring heavy communication costs. At the same time, the computation is highly load imbalanced as fewer compute resources are in use as one approaches higher levels in the hierarchy. This technique has been used by Pascucci et. al. [12] for computing merge trees of $3d$ regular grids. A more efficient strategy is used by [6, 10] where the global tree is distributed where each patch maintaining only the part of the global tree pertaining to that patch, referred as *local merge tree*. Here we extend the technique from [6] to d-dimensional, finite, regular CW-complexes and present its proof of correctness. This proof can be extended to all other existing parallel merge tree computation approaches like [10, 12].

Definition 16. Let P and S be d-dimensional sub-complexes of $\mathcal{K}$. The **local merge tree**, $LT(\mathcal{K},\ P,\ \widehat{\partial}S)$, of f on the domain $\mathcal{K}$, local to the patch P with respect to the boundary of $S \subseteq \mathcal{K}$ is given by:

- the arcs and/or nodes of $MT(\mathcal{K})$ that contain at least one point corresponding to the image of point/s in P under the map ϕ;
- the upper and lower nodes of the above arcs;
- these arcs and/or nodes are augmented with the maxima of f when f is restricted to individual *boundary-components*, $\widehat{\partial}S$, of S.

Given a domain decomposition of $\mathcal{K}$ into patches P_i, our goal is to compute the $LT(\mathcal{K},\ P_i,\ \widehat{\partial}\mathcal{K})$ for each $P_i \in \mathcal{K}$. By Def. 16, $MT(\mathcal{K})$ can be easily obtained once we have the individual $LT(\mathcal{K},\ P_i,\ \widehat{\partial}\mathcal{K})$.

In order to compute the $LT(\mathcal{K},\ P_i,\ \widehat{\partial}\mathcal{K})$, we start by computing the $LT(P_i,\ P_i,\ \widehat{\partial}P_i)$, for each patch. We compute these using the recursive algorithm from section 3.4 by invoking the $BuildMT(P_i,\ h=1)$ for each of the patch. Now instead of joining these trees with neighboring patches, we can modify the algorithm to reduce the communication cost of the distributed computation.

Definition 17. Given an arc with end nodes (u,v) in a merge tree, we say that u is the **parent** of v if $f(u) > f(v)$. Given a patch $P \subseteq S \subseteq R \in \mathcal{K}$ and having $LT(R,\ P,\ \widehat{\partial}S)$, the **boundary merge tree**, denoted as $BT(R,\ P,\ \widehat{\partial}S)$, is the set of nodes and arcs that lie on the monotonic descending paths from the parents of the boundary maxima of components of $\widehat{\partial}S$ to the root in $LT(R,\ P,\ \widehat{\partial}S)$. Thus, $BT(R,\ P,\ \widehat{\partial}S) \subseteq LT(S,\ P,\ \widehat{\partial}S)$.

Theorem 3. *Given patches $P,Q \in \mathcal{K}$ such that $P \cap Q \neq \emptyset$. The nodes and arcs of $LT(P,\ P,\ \widehat{\partial}P)$ that are not part of $BT(P,\ P,\ \widehat{\partial}P)$ remain unchanged in $LT(P\cup Q,\ P\cup Q,\ \widehat{\partial}(P\cup Q))$.*

Proof. We refer the reader to [10] for the proof. As the components of the super-level sets that reside entirely in $P\backslash Q$ and are not connected to the boundary, they cannot be affected by the join operation. □

The following operator extracts the boundary merge tree from a given merge tree augmented with boundary maxima of boundary components.

Definition 18. Given a local merge tree, $LT(P,\ P,\ \widehat{\partial}S)$ or a boundary merge tree, $BT(P,\ P,\ \widehat{\partial}S)$, the **extract boundary** operator, $\mathbb{E}$, with respect to $\widehat{\partial}R$, extracts the boundary merge tree, $BT(P,\ P,\ \widehat{\partial}R)$, where $\widehat{\partial}R \subseteq \widehat{\partial}S$ from $LT(P,\ P,\ \widehat{\partial}S)$ or $BT(P,\ P,\ \widehat{\partial}S)$. Thus, $\mathbb{E}(LT(P,\ P,\widehat{\partial}S),\widehat{\partial}R) = BT(P,\ P,\ \widehat{\partial}R)$.

The boundary merge trees can be easily extracted by traversing the tree from the leaves, identifying the boundary maxima and selecting the nodes and arcs from the parents of these maxima till the root of the tree.

3.5.1 Join Hierarchy of Boundary Trees

As the nodes and arcs that lie in the interior of a patch do not change after a join, this implies that only the *BT* is affected by the join. We exploit this property by joining only the boundary trees of patches to form boundary trees of patches at the next level. These in turn are joined successively creating a hierarchy of join stages.

Theorem 4. *Given the boundary merge trees of two patches P and Q as* $BT(P,\ P,\ \widehat{\partial}P)$ *and* $BT(Q,\ Q,\ \widehat{\partial}Q)$ *then,*
$BT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q) = BT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$

Proof. Let $R_P = LT(P,\ P,\ \widehat{\partial}P) \setminus BT(P,\ P,\ \widehat{\partial}P)$,
$R_Q = LT(Q,\ Q,\ \widehat{\partial}Q) \setminus BT(Q,\ Q,\ \widehat{\partial}Q)$,
$R = LT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q) \setminus BT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$
From Theorem 3,
R_P and R_Q remain unchanged in the join $LT(P,\ P,\ \widehat{\partial}P) + LT(Q,\ Q,\ \widehat{\partial}Q)$. Thus,
$LT(P,\ P,\ \widehat{\partial}P) + LT(Q,\ Q,\ \widehat{\partial}Q) = R_P \cup R_Q \cup (BT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q))$
Also, $LT(P,\ P,\ \widehat{\partial}P) + LT(Q,\ Q,\ \widehat{\partial}Q) = LT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$
The boundary maxima in the above join remain unchanged as the join operation does not give rise to new maxima or does not alter the existing boundary maxima. Thus, $BT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$ contains all boundary maxima from $\widehat{\partial}P$ and $\widehat{\partial}Q$. This implies, $R = R_P \cup R_Q$. Hence, $BT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q) = BT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$ □

After every join stage, the resulting tree is used by the patches to update their *LT*s with respect to the grown boundary. We refer to this as the *localization* described in the next section. After this the *BT*s are pruned and the *BT* for the next level patches are extracted. These are used by the next stage of the merge. As we consider $\mathcal{K}$ to have no boundary, $BT(\mathcal{K},\ \mathcal{K},\ \widehat{\partial}\mathcal{K})$ is empty. An example of this process is shown in Fig. 3.5.1.

3.5.2 Localization of Merge Trees to Patches

After every join stage of the *BT*s, the resulting boundary trees are joined with corresponding patches to obtain the *LT*s. This way a patch can obtain the connectivity information of its super-level set components that grow into neighboring patches by using the boundary trees of its neighbors.

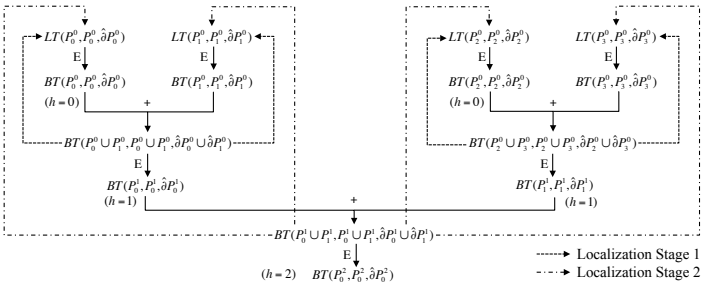

Fig. 2 An overview of the distributed computation for a domain consisting of four patches at level $h=0$, two patches at level $h=1$, and the whole domain at $h=2$.

Theorem 5. *Given neighboring patches $P, Q \in \mathcal{K}$, then*
$LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q)) \subseteq LT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q)$.

Proof. We know from def. 11 that
$LT(P,\ P,\ \widehat{\partial}P) + LT(Q,\ Q,\ \widehat{\partial}Q) = LT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$.
We also know from theorem 3 that the arcs that lie entirely in the interior of Q do not get affected by the join. So, only $BDT(Q,\ Q,\ \widehat{\partial}Q)$ which contains all the connectivity information participates in the join.
Thus, $LT(P,\ P,\ \partial P) + BT(Q,\ Q,\ \widehat{\partial}Q) \subseteq LT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q)$.
Now, by definition of LT,
$LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q)) \subseteq LT(P \cup Q,\ P \cup Q,\ \widehat{\partial}P \cup \widehat{\partial}Q))$ and contains all arcs local to P along with the correct connectivity with the arcs from $BT(Q,\ Q,\ \widehat{\partial}Q)$.
Thus, $LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q)) \subseteq LT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q)$. □

From the resulting tree of $LT(P,\ P,\ \widehat{\partial}P) + BT(Q,\ Q,\ \widehat{\partial}Q)$, we have to mark the boundary maxima of the components $\widehat{\partial}(P \cup Q)$ and prune the regular nodes. Let the tree obtained after marking the boundary maxima and prune operations be T. But $T \neq LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q))$ as T might contain arcs from Q that do not correspond to any point in P and hence cannot be in $LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q))$. To obtain the $LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q))$ from T we restrict it to P in the following way

- Let X is the set of nodes in T that correspond to the image of points in P under the map ϕ. Let $m \in X$ be the node with the minimum value in X. Let Y be all nodes and arcs that lie on monotonic descending paths from X to m
- We now add arcs, $(u,v), f(u) > f(v)$, along with the end nodes to Y such that $u \notin Y, v \in Y$.
- Lastly, we add an arc, $(m,v), f(m) > f(v)$ to Y, if it exists, such that $m \in X$ is the node with the minimum function value in X and $v \notin Y$.

The tree Y is the $LT(P \cup Q,\ P,\ \widehat{\partial}(P \cup Q))$. After every join stage we carry out this localization to obtain $LT(\mathcal{K},\ P,\ \widehat{\partial}\mathcal{K})$.

3.5.3 Time Complexity Analysis of the Distributed Merge Tree Computation

Let us assume that $\mathcal{K}$ has n cells distributed over p processors. Also, let us assume the patch hierarchy to be a k-way hierarchy, where k low level patches combine to

form a higher level patch. Assume the oracle of a cell with v vertices on average generates a tree with m nodes in t time. We expect the output trees to be sorted so t is at least $O(v \cdot log(v))$. However, if v is bounded, i.e. for regular grids, t will be constant. Thus, generating the merge trees for all of the *CW*-cells on each processor is $O(t \cdot n/p)$ and assuming a linear merge of the trees the time to construct the level 0 trees is $O((t+m) \cdot n/p)$. The additional steps to identify boundary maxima, extract the boundary tree, etc. are all linear in the size of the tree and thus do not add to the overall complexity. For known mesh types one could substitute any of the existing algorithms.

The expected size of a boundary tree is proportional to the size of the boundary. Assuming we merge spatial coherent patches, i.e. blocks of a regular grid or groups creates from a mesh partitioning scheme, the boundary trees of level 0 are expected to be of size $O((n/p)^{\frac{2}{3}})$. The merge on level 1 will thus take $O((n/p)^{\frac{2}{3}} \cdot k \cdot \log(k))$ with the additional $\log k$ factor corresponding to the priority queue needed during the merge. The resulting tree will be of size $O((k \cdot n/p)^{\frac{2}{3}})$. Thus the expected total time for all merges of all levels will be

$$O\left(k \cdot \log(k) \left(\frac{n}{p}\right)^{\frac{2}{3}} \cdot \sum_{i=0}^{\log_k p - 1} k^{\frac{2i}{3}}\right) = O\left(k \cdot \log(k) \left(\frac{n}{p}\right)^{\frac{2}{3}} \cdot \frac{(p^{\frac{2}{3}} - 1)}{(k^{\frac{2}{3}} - 1)}\right)$$

$$= O\left(\frac{k \cdot \log(k)(n^{\frac{2}{3}})}{(k^{\frac{2}{3}} - 1)}\right).$$

The localization step, which is derived from the join operator, occurs after every join operation in the hierarchy and is linear in the number nodes of the participating local tree and boundary tree. Furthermore, it is performed in parallel by each of the processor and hence does not add to the time complexity.

Complexity-wise the behavior for increasing mesh sizes is therefore dominated by the potentially $O((n/p)^2)$ behavior of the initial local compute, in case m is of order $O(n/p)$. In practice, this cost turns out to be negligible and the behavior is dominated by the number of merges and the increasing size of the boundary trees. Note, that in this respect the size of the boundary tree is a conservative estimate as the global domain boundaries actually do not contribute to the size of the boundary trees.

4 Conclusion

Topological analysis techniques have been extensively to analyze and visualize data generated by scientific simulations. But the growing compute power and enormity of data has created a need for scalable distributed algorithms. Furthermore, scientific simulations are moving towards using more sophisticated meshes in place of regular grids. For example, adaptive mesh refinement meshes are being adopted by various large scale scientific simulations.

In this paper, we have presented a distributed algorithm for computing merge trees on regular *CW*-complexes, which provides a theoretical foundation for computing merge trees for various types of meshes. We rely on an oracle to provide the merge tree of a single cell within a mesh and state the conditions and requirements from the oracle. Hence, as long as the oracle satisfies those conditions, the merge tree can be computed using the presented algorithm for any type of meshes that are regular *CW*-complexes.

Acknowledgements 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-XXXXX).

References

1. M. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics*, 82(1):64 – 84, 1989.
2. 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.
3. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24(3):75–94, 2003.
4. J. H. Chen. Petascale direct numerical simulation of turbulent combustionfundamental insights towards predictive models. *Proceedings of the Combustion Institute*, 33(1):99 – 123, 2011.
5. Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Computational Geometry*, 30(2):165 – 195, 2005. Special Issue on the 19th European Workshop on Computational Geometry.
6. A. G. Landge, V. Pascucci, A. Gyulassy, J. C. Bennett, H. Kolla, J. Chen, and P.-T. Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 1020–1031, Piscataway, NJ, USA, 2014. IEEE Press.
7. S. Maadasamy, H. Doraiswamy, and V. Natarajan. A hybrid parallel algorithm for computing and tracking level set topology. *20th Annual International Conference on High Performance Computing*, 0:1–10, 2012.
8. 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.
9. W. S. Massey. A basic course in algebraic topology. 1991.
10. D. Morozov and G. Weber. Distributed merge trees. *SIGPLAN Not.*, 48(8):93–102, Feb. 2013.
11. D. Morozov and G. Weber. Distributed contour trees. In P.-T. Bremer, I. Hotz, V. Pascucci, and R. Peikert, editors, *Topological Methods in Data Analysis and Visualization III*, Mathematics and Visualization, pages 89–102. Springer International Publishing, 2014.
12. V. Pascucci and K. Cole-McLaughlin. Parallel computation of the topology of level sets. *Algorithmica*, 38(1):249–268, 2004.
13. J. A. Rathkopf, D. S. Miller, J. Owen, L. Stuart, M. Zika, P. Eltgroth, N. Madsen, K. McCandless, P. Nowak, M. Nemanic, et al. Kull: Llnls asci inertial confinement fusion simulation code. *Physor 2000, ANS Topical Meeting on Advances in Reactor Physics and Mathematics and Computation into the Next Millennium*, 2000.
14. T. J. Tautges, C. Ernst, C. Stimpson, R. J. Meyers, and K. Merkley. *MOAB: a mesh-oriented database.* Apr 2004.
15. 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.