Data Parallel Hypersweeps for in Situ Topological Analysis

Petar Hristov
University of Leeds
Gunther H. Weber
Lawrence Berkeley National Laboratory, University of California, Davis
Hamish A. Carr
University of Leeds
Oliver Rübel
Lawrence Berkeley National Laboratory
James P. Ahrens
Los Alamos National Laboratory

***Abstract*—The contour tree is a tool for understanding the topological structure of a scalar field. Recent work has built efficient contour tree algorithms for shared memory parallel computation, driven by the need to analyze large data sets in situ while the simulation is running. Unfortunately, methods for using the contour tree for practical data analysis are still primarily serial, including single isocontour extraction, branch decomposition and simplification. We report data parallel methods for these tasks using a data structure called the hyperstructure and a general purpose approach called a hypersweep. We implement and integrate these methods with a Cinema database that stores features as depth images and with a web server that reconstructs the features for direct visualization.**

I. Introduction

Computational scientists use massive numerical simulations to study physical phenomena. As these simulations increase in size, techniques for analyzing and displaying the data are increasingly important. However, due to limited bandwidth to disk and in the human visual system, this increasingly depends on running analytics and visualization tools *in situ* during simulation rather than *post hoc*.

An important analytic tool available for scalar fields is the contour tree, which captures the relationship between the isocontours in the data, annotated with geometric measures, such as volume and intensity, that are of significance to the science behind the data. In order to apply these tools at scale, recent work has focused on building parallel algorithms and data structures for computing, manipulating and interpreting contour trees, first in data parallel environments, and in the future in hybrid clusters with heavy on-node data parallelism. Data parallel algorithms to compute and augment the contour tree have been reported [6], [11], but not secondary computations. Those secondary computations are geometric measures, branch decomposition, simplification and single isocontour extraction.

The first contribution of this paper is to introduce data parallel algorithms for those secondary computations. To compute geometric measures we develop a method we call a hypersweep that is a modification of the parallel tree contraction algorithm [21]. The hypersweep method arises naturally from the computation of the contour tree and unlike parallel tree contraction it respects the semantics of the contour tree as a data structure. For branch decomposition and simplification we replace the standard inherently serial priority queue computation with a local and trivially parallelisable algorithm. The second contribution is an implementation of those secondary measures in the open source VTK-m library. The final contribution of this paper is to link the resulting code with the existing *in situ* Cinema database to demonstrate viable data-parallel contour algorithms for the entire analysis and visualization pipeline.

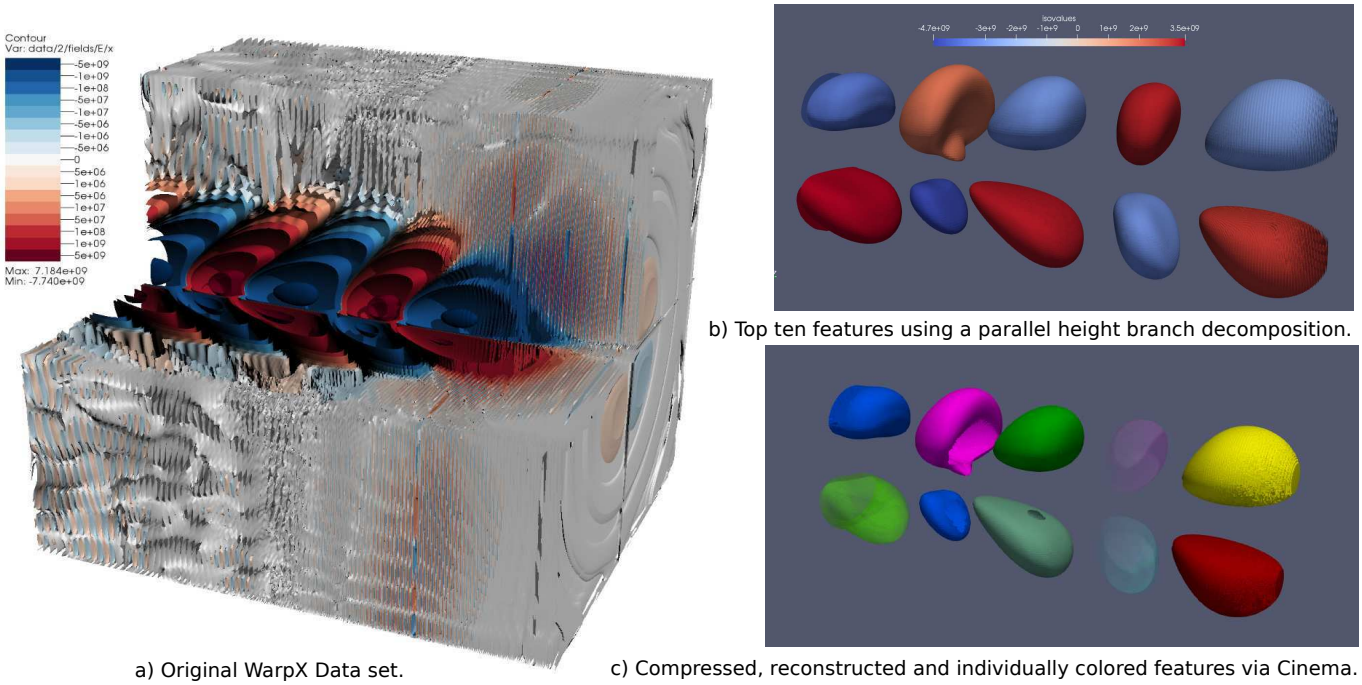
We review the relevant background literature in Section II, then introduce the hypersweep in Section III, showing how to adapt branch decomposition to data-parallel computation. We then describe how to extract significant isocontours in data-parallel (Section IV), report how we integrated it with the Cinema database (Section V), and show its application to a WarpX laser plasma particle accelerator simulation (Section VI). We then evaluate performance (Section VII) and discuss conclusions and future research directions (Section VIII).

II. Background

Given a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, a *level set* is the set of points at an *isovalue* h: $f^{-1}(\{h\}) = \{x \in \mathbb{R}^n \mid f(x) = h\}$. We call individual connected components of level sets *contours*. As h varies, contours appear, disappear, connect or disconnect at *critical points* where the gradient vector is zero, which may be *peaks, pits or saddles*. If we contract each contour to a single point, the resulting structure is called a *contour tree* [5]. For functions over general manifolds, the structure may have cycles, and is called a *Reeb graph*. In both, critical points are known as *supernodes* and are connected by *superarcs*, with regular (non-critical) mesh vertices represented as *nodes* strung on *arcs* along the superarcs. An *augmented contour tree* contains regular nodes, and is more expensive, but often more useful.

If we instead take the connected components of sublevel sets such that $f^{-1}((-\infty, h]) = \{x \in \mathbb{R}^n \mid f(x) \leq h\}$ we can construct

e-mail: p.hristov@leeds.ac.uk
e-mail: ghweber@lbl.gov.us
e-mail: h.carr@leeds.ac.uk
e-mail: oruebel@lbl.gov.uk
e-mail: ahrens@lanl.gov.uk



a) Original WarpX Data set.

b) Top ten features using a parallel height branch decomposition.

c) Compressed, reconstructed and individually colored features via Cinema.

Fig. 1: a) Isosurface visualization of the transverse electric field Ex of a WarpX laser plasma particle accelerator simulation [11]. b) Visualization of the ten most-significant contours detected automatically using a branch decomposition of the contour tree using our data-parallel, height-based simplification method that correctly captures the topology of the data set. c) For interactive, post-hoc visualization, we compute and store features in a Cinema image database in situ and reconstruct them via a web interface. We store features individually and this allows us to manipulate their properties such as color, scale, opacity, etc.

a tree known as the *join tree*. If we take the connected components of superlevel sets or $f^{-1}([h,\infty)) = \{x \in \mathbb{R}^n \mid f(x) \geq h\}$ we obtain the *split tree*. Collectively, the join and split trees are referred to as *merge trees*. They are important in computing the contour tree, but can also be used independently for analysis [4].

In serial, a single sweep computes the join tree, a second one computes the split tree, then superarcs are transferred from the outside of the merge trees inwards to construct the contour tree [8]. Subsequent work parallelised this in shared memory [6], [11], [15], [16], in distributed clusters [18], [23], [24], [26] or on hybrid models [1], [20], [28]. The most performant shared-memory approach is the PPP algorithm [6], [11], which we use as the basis for our computations.

The PPP algorithm [11] computes the merge trees in two phases: topology graph construction and parallel peak pruning (PPP). In the first phase, the input mesh is abstracted to a *topology graph* [7] in which all critical points are represented, and edges represent monotone paths between critical points. This is then used as the input to the second phase, where superarcs from peaks to the topologically nearest saddle are found and added to the merge tree in parallel, then removed from the topology graph. What remains of the topology graph now has new peaks which used to be saddles, but some saddles become regular and can be collapsed out. As a result, the join (or split) tree can be computed in a logarithmic number of passes, and in later passes, large numbers of superarcs are transferred at once.

PPP batches superarc transfers from the merge trees to the contour tree, alternating between maxima and minima. In every stage leaves can be transfered in parallel because that is a local operation. To speed up computation long chains of degree two vertices are transfered in a single stage. Due to the specifics of the computation those chains need to be monotone in the function values at the vertices.

The original PPP algorithm has been recently extended to compute the augmented contour tree efficiently [6]. The extension was to record the monotone chains from the merge phase to guarantee the ability to search for regular nodes in logarithmic time. The endpoints of the monotone chains are recorded as hyperarcs, similary to the already existing superarcs and regular arcs. The hyperarcs of the contour tree form what we call the hyperstructure [6].

We illustrate the idea of the hyperstructure with the right hand subfigure of Figure 2. The supernodes of the contour tree are labeled with the number of the iteration they are transfered in the merge phase of the algoritm. The supernodes in the tree are connected by superarcs and hyperarcs with small and large arc widths respectively. The hyperarcs store a monotone path of supernodes (sorted by value) that are collapsed in a single iteration of the merge phase.

We can think of hyperarcs as shortcuts for more efficient computation. Since the supernodes in a hyperarc are in monotone order we can insert regular nodes by comparing against the endpoints of the hyperarc. If the regular node's value is not in that interval, we move along the next hyperarc and skip a potentially large number of supernodes. Otherwise we use binary search on the supernodes in the hyperarc. In

this paper we'll describe how we can use the hyperstructure to speed up secondary computations such as geometric measures and contour extraction.

Since the merge phase of the contour tree algorithm and the hyperstructure process monotone paths an issue emerges with non-monotone paths. Non-monotone paths in the contour tree are refered to as W-structures [17] because of the way they zig-zag up and down. We refer to the size of a w-structure as the number of maximal monotone paths. W-structures are significant because they serialize the computation of the merge phase and can be used to show that persistent homology differs from branch decomposition [17].

A. Simplification and Branch Decomposition

Once the contour tree has been computed, it can be simplified so that only significant features are represented. This is usually done by removing the least significant leaf edge in the contour tree, collapsing regular nodes if necessary, and iterating until only one master branch remains [27]. Note that this is an inherently serial computation.

This simplification process forms a hierarchy of branches called the *branch decomposition*. For this purpose, "least significant" can be interpreted by computing the difference in function value between an extremum and a saddle, or by computing *geometric measures* [9] such as volume or integrated function value (hypervolume) for the set of contours corresponding to a given superarc or subtree.

A related idea is present in persistent homology [12], where the difference between a peak and a saddle in the sort order of the mesh vertices gives the *persistence*, and is used to pair, or cancel, peaks and saddles (or pits and saddles), or alternately, the difference in function value between peak and saddle. Recent work has confirmed [17] however that the cancellation pairs from persistent homology are only guaranteed to match branches in the branch decomposition if no W-structures are present. In practical data, W-structures exist (as we will see in Section VII) and persistent homology gives a different result from branch decomposition. We use the term *height* of a feature to refer to the difference in value along a superarc to avoid confusion with the formal definition of persistence.

Given a simplified contour tree, visualization interfaces can be built that show only the most significant features or contours, and allow visual manipulation [9] of the remaining features, or extraction for subsequent processing with other algorithms. In essence, the goal of this paper is to replace the previous serial algorithms for geometric measure computation, simplification, branch decomposition and single isosurface extraction with data-parallel equivalents so that they can be run in an *in situ* environment efficiently.

B. Parallel Tree Operations

A fundamental parallel tree algorithm is parallel tree contraction [14], [21]. Parallel tree contraction is a bottom up technique where we start at the leaves of a rooted tree and move inwards in stages. In every stage all leaves with a different parent are processed independently in parallel. Once all leaves are processed they are discarded (raked) and new vertices become leaves. If the tree is unbalanced the rake operation serializes the computation along chains of vertices of degree two. Those chains can be contracted using pointer doubling or a prefix scan. After a logarithmic number of rake and contract operations the whole tree is contracted to its root. At the end every vertex accumulates the value that corresponds to evaluating the expression over the subtree whose root is that vertex.

As we have noted above, the merge phase of the PPP algorithm [10] is a variation of parallel tree contraction, but with several differences. First, the hyperstructure only collapses chains whose vertices are monotone in value: this property is required to support binary search for data values along a path in the tree. Second, due to the need to keep intermediate results updated, the hyperstructure transfers upper leaves and lower leaves in alternating passes. While it is tempting to view each pair of upper and lower iterations in the hyperstructure as equivalent to the contraction phases, variations are visible even in small trees, as shown in Figure 2.

C. The Cinema In Situ Database

Advances in processing power for extreme scale scientific computation have greatly outpaced data bandwidth and I/O, impeding visualization and analysis. A Cinema database [2] is a large collection of images which are sampled based on time, visualization object and camera position, and stored along with metadata that allows interactive querying [25]. Cinema is used with image processing techniques to combine images to obtain new camera and time locations or even to reconstruct the original object using Depth Image Based Rendering [19]. Cinema has been implemented in ParaView as well as the open source Topology Toolkit TTK [30]. However, since the images and the metadata are orders of magnitude smaller than simulation raw output they can be transfered for post hoc analysis and visualization. This requires sophisticated techniques for identifying features of interest, hence the interest in contour trees for analysis at scale. Our approach allows us to compute the triangles of connected components in situ and, by storing them as Cinema image collections, reduce their size and visualize large-scale simulation runs interactively on commodity hardware.

III. Hypersweeping Geometric Measures

The first contribution in this paper is to describe data-parallel computation of geometric measures such as volume and height (if not persistence), and to use them to construct branch decompositions. Geometric measures describe properties of a region bounded by a given contour, i.e. a region corresponding to a subtree of the original tree. This means that the volume (for example) is determined by all superarcs in the subtree, not just the final superarc at which the subtree is rooted. We will need to evaluate arithmetic expressions over subtrees of the contour tree and so we look to the parallel tree processing technique we discussed in Section II-B.

Since parallel contraction is well-established, we will not illustrate the process in detail, restricting ourselves to the computations of interest, and commenting on how the variation of

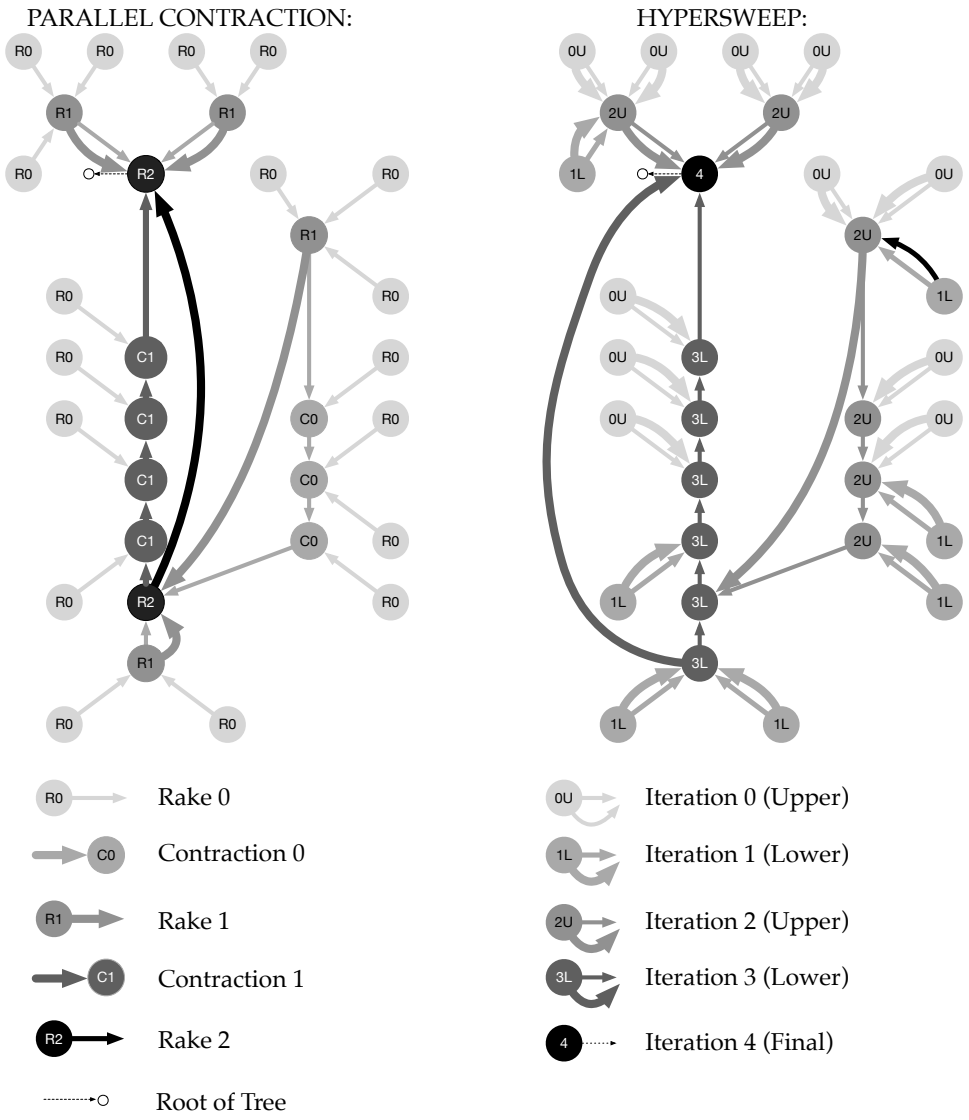


Fig. 2: On the left is a contour tree whose vertices are annotated based on how and when they're processed by the parallel tree contraction algorithm. On the right is a hypersweep of the same contour tree annotated with the hyperstructure. While the two methods are substantially similar, minor differences arise because PPP [11] alternates upper and lower leaves, and because only monotone chains can be compressed. Even in a small tree, this leads to differences between the two methods, as can be seen in the operations Rake 2 and Iteration 3 (Lower).

the hypersweep from parallel contraction affects the algorithmic analysis. We illustrate this in the left-hand column of Figure 3, where we compute an approximation of contour volume by counting the number of contained regular nodes [29]. In this image, we use shading to indicate which nodes belong to which iteration of the contraction. Each pair of iterations in the hypersweep normally corresponds to a single iteration of the parallel contraction, although there are edge cases where the exact order is different.

In the absence of W-structures [17], the chains in each pair of sweeps will remove the same supernodes as a single iteration of the parallel tree contraction: since this is a constant factor, the overall analysis is unchanged. In the presence of W-structures, however, the hypersweep cannot be bounded by $O(\lg t)$ time complexity and $O(t \lg t)$ work where t is the tree size. We have demonstrated elsewhere [6], [17] that in practice the work is still bounded by $O(t \lg t)$, and that the time complexity is typically better than $O(\lg t)$.

Subtree Volume: We know [9] that the number of regular nodes in a subtree approximates the volume of the regions represented by branches of the contour tree. While we could do a hypersweep with regular nodes rather than supernodes, it is less efficient. We therefore use prefix sum operations to compute the number of regular nodes on each superarc as the initial value at each supernode, as shown in the left column of Figure 3. We use shading to indicate the iteration in which these values are propagated inwards by prefix sums, resulting in the final tree sizes visible in the lower register.

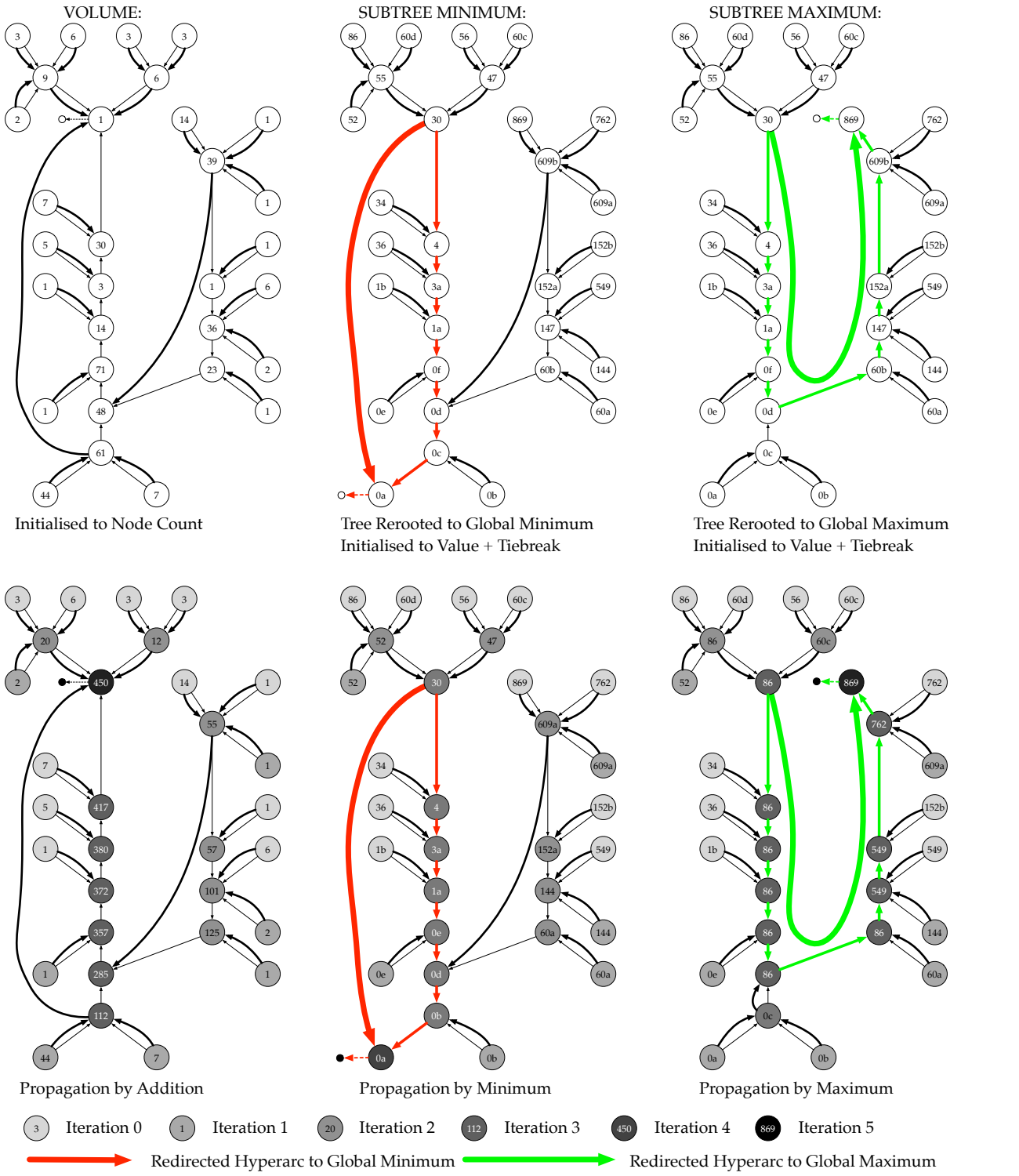


Fig. 3: Hypersweep Computation of Geometric Measures based on the Parallel Tree Contraction [21]. Vertices are labeled with their scalar value. Those with equal value are also labeled with a letter indicating their order using simulation of simplicity [13]. For volume approximation (Left), we initialize each supernode to the number of regular nodes on its superarc, then propagate towards the root with a prefix-sum. For sub-tree minimum and maximum (Centre and Right), we re-root the tree to the global minimum (maximum), initialize to the supernode's data value, then propagate by prefix-minimum (maximum). Shades of grey are the iteration in which a node gets a final value.

A. Branch Decomposition and Subtree Height

Once we have established subtree volume, we build branches by having each vertex choose locally the superarc with the highest ascent and descent. The branches are then groups of adjacent superarcs that greedily maximize subtree volume or

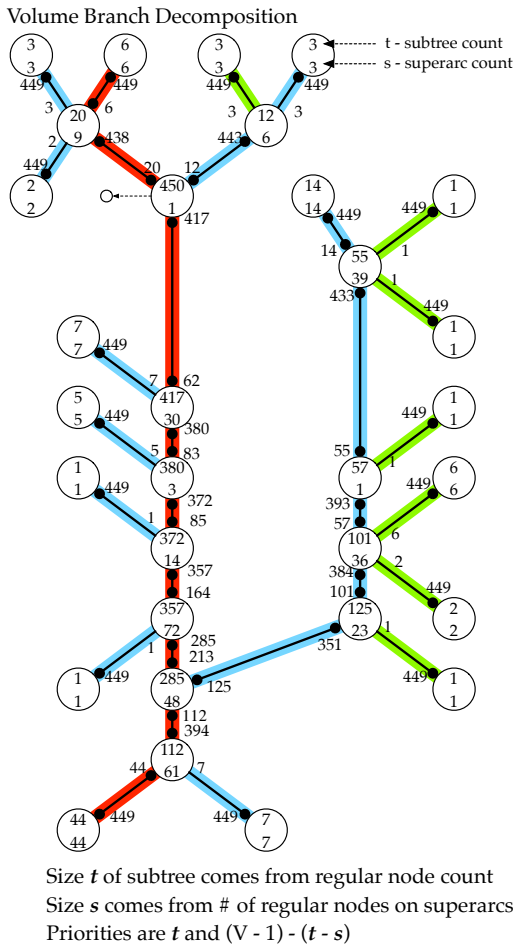


Fig. 4: Branch Decomposition By Volume. Different geometric measures lead to different branch decompositions. Notice the master branch is different from the one in Figure 5.

for that matter any geometric measure we have defined on the subtrees. We demonstrate this in Figure 4 with a black dots on the edge adjacent to the best up and best down of each supernode. After each vertex chooses the "best" ascent and descent, we use pointer-doubling to collect the branches.

Building the standard branch decomposition [27] based on branch height is more difficult. When there are no W-structures in the contour tree each vertex can select the highest (or lowest) reachable maxima (or minima). In the presence of W-structures we need to compute the longest branch in every subtree. This makes the existing branch height decomposition difficult to compute in parallel. We will deal with this in more detail in Section VII-B. Now we will introduce an alternative geometric measure that is readily parallelizable.

Instead of branch height, we consider subtree height for the branch and all child branches. We define subtree height as the difference in function value between the maxima and the minima of a subtree. This means that we need the minimum and maximum values in every subtree from the root outwards. As we will see later Figure 10, this gives a slightly different branch decomposition than previous definitions, but only in the presence of W-structures.

We can now frame this in terms of a hypersweep operation: to find the minimum value in each subtree, we re-root the hyperstructure at the global minimum, then apply a hypersweep with the minimum operator. Re-rooting the hyperstructure is fairly straightforward: we select the global minimum m, and identify the hyperarcs along the path P between it and the previous root r, at a cost of at most $|P|$.

All paths from the leaves to the root terminate at the root r or at this path P. We convert this path to a new hyperarc (which may not be monotone) with at most the same number of iterations as before. This new hyperac is shown in the upper register of the middle column in Figure 3 as a thick red edge. We then hypersweep to propagate minima through the tree towards the minimum m, as shown. Similarly, the right column of Figure 3 shows the re-rooting and hypersweep to compute subtree maxima.

In the next stage of the computation, shown in Figure 5, we annotate every edge in the tree with *two* values: the minimum in the direction of the hypersweep, and the minimum in the other direction. Of these, the minimum in hypersweep direction is set to the value just computed. The minimum in the other direction will always be the global minimum, since it is the new root of the tree.

For example, in the left top corner, the vertex with value 86 forms a subtree, and the propagated minimum value, 86, is the value we use when pruning towards the root: the global minimum value, 0, is the value when pruning away from the root. Now, for each possible pruning (i.e. at each end of the superarc), we add the value of the supernode itself, then take the maximum and minimum of the three values: thus, if the supernode value is the lowest, it replaces the minimum, if the highest, it replaces the maximum. Finally, we subtract minimum from maximum to get the subtree range.

Considering vertex 86 once more, pruning at the lower end of superarc $86-55$ gives a subtree minimum of 86 and maximum of 86. We substitute 55 for the minimum, and compute a subtree height of 31. Further in, at the lower end of superarc $30-4$, the maximum is 86 in the upwards subtree and the minimum 30. Replacing 30 with 4, we compute an upwards subtree height of 82.

B. Simplification

Having computed our geometric measures and branch decomposition, simplifying to a threshold amounts to ignoring branches of the contour tree that fail a logical test. For example, suppose we want to ignore all branches that involve less than 1% of the data. This is achieved by testing all superarcs to see whether their volume (or height) is over the threshold, which is trivial to do in parallel. If desired, the "weight" of the pruned branch can be retained by keeping the terminal superarc as an augmenting node in the simplified tree.

IV. Feature Extraction

Once the contour tree has been computed, decomposed and simplified, visualization interfaces extract contours corresponding to selected superarcs. In prior work [9], the user interactively selected contours and manipulated them visually. While this is still possible with the data-parallel contour tree, one goal

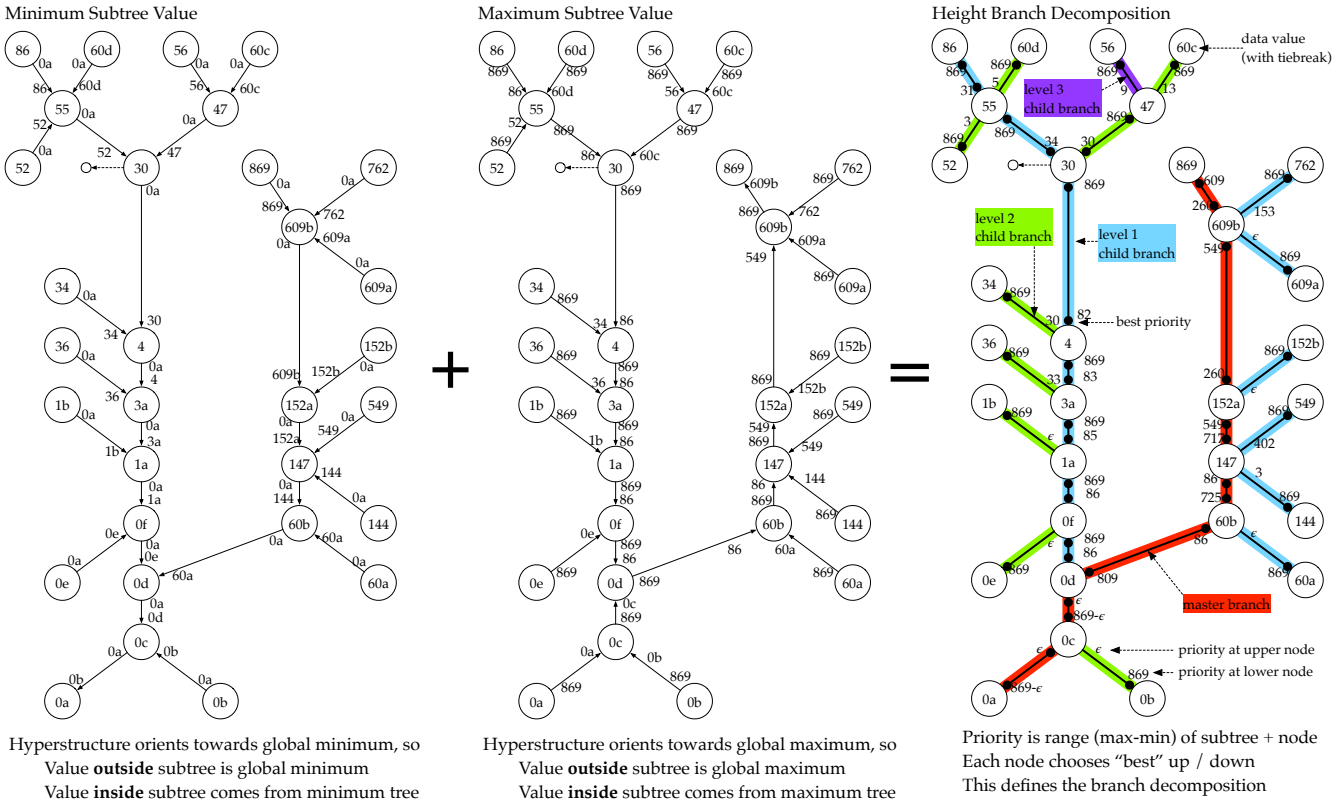


Fig. 5: Computing Branch Decomposition. After computing minima and maxima per subtree (Figure 5), each superarc is annotated with the best available min/max from each end (the first and second contour trees). At each vertex, the incident superarc with highest value is chosen (third tree). For W-free contour trees, the result is guaranteed to be identical to the standard branch decomposition and persistent cancellation. For trees with W-structures, a side tree may have a larger range of values than the obvious choice, so some arcs will differ.

of *in situ* visualization is to defer user interaction until later. We adopt an alternate solution - local contours [9], where we choose a relative isovalue on each branch - normally 50%, or halfway along it.

Previous work [9] adapted the continuation method [32] to extract single contours, but this approach is essentially serial. Instead we extract a contour for a branch using marching cubes and a method based on searching the contour tree [31]. First we use a parallel implementation of marching cubes to extract an isosurface for the isovalue of that branch. Next we filter out the triangles produced by marching cubes that do not belong to the branch.

To determine if a triangle belongs to a branch consider a mesh edge u, v that intersects the triangle. Since the path from u to v in the mesh is monotone there is monotone path from u to v in the contour tree. Therefore along that path there is a superarc whose endpoints' values contain the isovalue for the branch. We search for that superarc with the hyperstructure because it supports efficient search for regular points at logarithmic cost [6]. If that superarc belongs to the branch we keep the triangle, otherwise we discard it.

Each such contour can be extracted in $O(k \lg T)$ time, where k is the size of the entire isosurface, and $O(\lg T)$ is the cost of searching the hyperstructure to find the corresponding superarcs. For a small number of contours (e.g. 10 or 20), we iterate over their superarcs and values to generate them, with the advantage that we will extract them as separate surfaces and can render them accordingly. For large numbers of contours, each mesh cell (or mesh edge) can search for the corresponding path(s) in the contour tree and compare them all at once, but we have not yet implemented this variation.

V. Cinema Integration

To demonstrate integrating our parallel methods into a full visualization pipeline, we developed the "contour visualizer" application prototype. Our goals in developing this application were: (i) to extract a representative set of contours from the scalar field with minimal user interaction; (ii) utilize high-performance computing to handle large-scale data sets; (iii) to use standard scientific visualization libraries for easy integration into existing project.

We implemented the hypersweeps described in Section III as part of the VTK-m project, and integrated them with the existing Cinema database application, using a two stage visualization pipeline. In the first stage, we extract, compress and store features from scalar fields in a Cinema database. In the second stage, we read images from this database, reconstruct features from depth images and visualize them. All of the methods developed have been contributed to the development branch of VTK-m, and are available for use.

Our input is assumed to be a standard VTK image format. While our current implementation works with regular, rectilinear grids, the underlying algorithms employ the topology graph abstraction referred to in Section II, and are valid for any simply connected mesh, subject to writing suitable adaptor classes.

We compute the contour tree of the scalar field, assuming marching cubes connectivity, using the VTK-m [22] implementation of the parallel peak pruning algorithm [6], [11]. Subsequently, we compute the branch decomposition (as described in Section III) either using subtree height or volume as the simplification measure and simplify the branch decomposition to a specified number of branches.

As described in Section IV, we then simplify the contour tree to the top 10 most important branches, and extract one representative contour per branch in local contour mode. At present, we usually choose the 50% isovalue on each branch, but we have also used the 1% isovalue to select contours very close to the critical point: in future we expect to choose multiple contours along each branch.

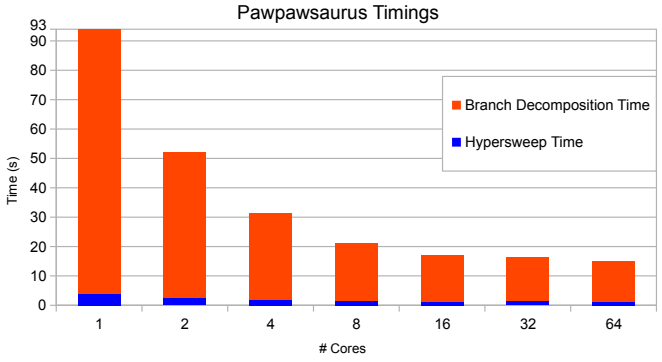


Fig. 6: Subtree height branch decomposition (red) and hypersweep (blue) on Pawpawsaurus. While the scaling plateaus after 8 cores, the performance is overall good especially compared to contour tree computation (see Table 1).

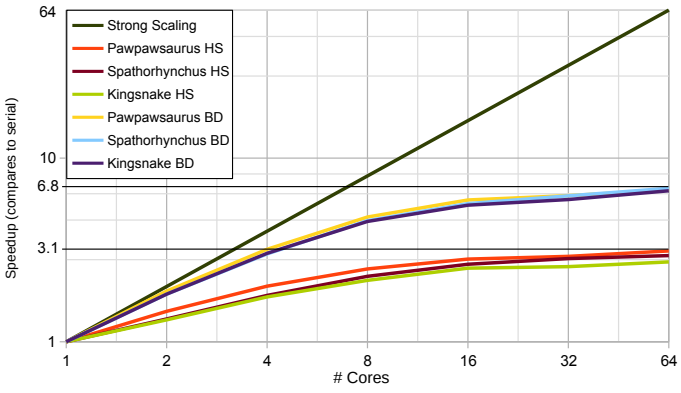


Fig. 7: Scaling of 3D data for up to 64 cores and TBB on Haswell (log/log). The black line shows the ideal strong scaling. Hypersweep (HS) and branch decomposition (BD) are related and have similar scaling patterns: it is possible that the cause is external (VTK-m).

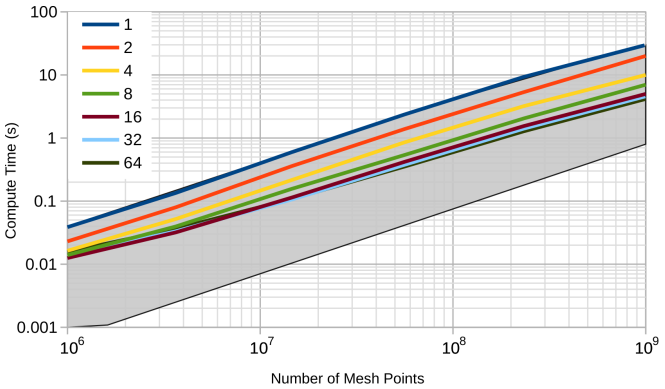


Fig. 8: Scaling using 1 to 64 threads on the 2D Scaled GTOPO Datasets (log/log). The grayed out polygon is perfect weak scaling.

After single contour extraction *in situ*, the first stage is complete, and we save depth images from varying camera positions for later reconstruction [19] based on a TTK [30] implementation in order to avoid saving large meshes of millions of triangles.

The second stage supports post hoc exploration of the data artifacts stored in the Cinema database. We first read all depth images in the cinema database and reconstruct each feature individually using the TTK [30] implementation of the VOIDGA algorithm [19]. The quality of the reconstruction depends on image resolution, camera placement and number of viewpoints in the Cinema database.

As with the Cinema database in general, our project can use different front ends. For some purposes, we use ParaView and TTK, but for others, we implemented a simple web server and web interface to reduce the learning curve for end users. We implemented this using node.js for the server and Three.js for the front end.

The quality of the reconstruction can vary but does not need to be perfect, only good enough for the user to get a general idea of the data. Should the user require the original features they can be retrieved at a higher bandwidth and time cost, and we will explore the best parameter choices for *in situ* visualization in the future.

This visualization pipeline is an improvement upon previous ones such as [3]. Every step in our pipeline is fully data-parallel and it is implemented using popular open source visualization libraries such as VTK-m and TTK. Furthermore our pipeline adds the additional step of reconstructing the depth images in 3D.

VI. Application Example - WarpX

Figure 1 shows the application of the automatic contour selection to the transverse electric field (Ex) of a WarpX laser plasma particle accelerator simulation. Plasma-based accelerators use short ($\leq 100fs$) ultrahigh intensity ($\geq 10^{18}W/cm^2$) laser pulses to drive waves in a plasma. Electrons that become trapped in the plasma (or externally injected electron or positron beams) are then—much like a surfer riding a wave—accelerated by the wave to high energy levels. Understanding the structure of the

electric forces generated by the plasma wave is critical to the design and optimization of plasma-based particle accelerators and understanding of the fundamental physical phenomena. In this context, the difference in function value (i.e., height of arcs in the contour tree) is an important measure of the strength of the electric forces generated by the corresponding feature (i.e., contour) in the electric field. As Fig. 1b shows, using height as importance metric allows us to automatically identify the features with the largest focusing gradients in the transverse electric field Ex, describing the primary structures of the electric field generated by the plasma wave driving particle acceleration. By rendering the contours in situ and storing for each contour a separate depth-image in a Cinema database, users can interactively explore, visualize, and compose the features post-hoc. By storing the additional metrics computed from the contour tree (e.g., volume and persistence of contours) alongside the generated depth images, enables quantitative analysis of the contour-based features and interactive query of the Cinema database to search for relevant contours.

VII. Evaluation

Next we evaluate the compute performance of our implementation (Sec. VII-A) and how well it picks out significant contours (Sec. VII-B).

A. Performance

As noted above, our implementation is freely available in the open source VTK-m library [22]. However there is no implementation of branch decomposition in any other actively maintained visualization library (TTK and VTK). To ensure consistency between methods, we re-implemented branch decomposition in serial. It performed with about the same running time as the parallel branch decomposition on a single core. We have not included those specific running times because our serial branch decomposition was implemented as reference for comparison not with performance in mind.

We ran tests on standard data sets well known in the visualization community or that we have used previously [6], [11], and refer the reader to the appendices of those papers for full details. The Asteroid dataset is freely available courtesy of LANL, the WarpX dataset is not freely available at present.

Our primary test system is the NERSC Cori supercomputer at Lawrence Berkeley National Laboratory, whose Haswell compute nodes have two 16-core Intel® Xeon ™ E5-2698 v3 CPUs with two hyperthreads per core, clocked at 2.3 GHz and with 128GB DDR4 main memory at 2133Mhz. We compiled and used the VTK-m library with Intel's Thread Building Blocks (TBB) threading API.

We first computed the augmented contour tree for each data set using VTK-m's contour tree filter [6]. Next we compute the branch decomposition of every data set with a range of 1, 2, 4, 8, 16, 32 and 64 cores. Finally we compute the branch decomposition over the GTOPO30 data set with 64 cores, but with different scales of the data. This way we can study scaling on a set of related data sets.

In Figure 6, we show timings for the Pawpawsaurus data set. We have chosen Pawpawsaurus because it is one of the largest data sets we have available in terms of regular and super node count. We therefore expect to see the scaling of the hypersweeps, rather than the cost of initialising parallel data structures. Here, we see the most performance gain in going from 1 to 2 cores and then to 4 cores and 8 cores. This is also visible in Figure 7 where the speedup of the hyperswep and the branch decomposition is 3.1 and 6.8 respectively.

Similarly Figure 8 suggests that while the scaling is not linear (gray area in the plot) the performance is still good in practice. This is further supported by Table I where we can clearly see that the hypersweep and branch decomposition are only a small fraction of the computation time of the contour tree. On average the branch decomposition is only 1.76% of the contour tree computation time, so we do not yet see the need for further optimization.

An important reason for the good practical performance of our methods is the topological complexity of the data sets. Remember that our methods do not scale with size of the input mesh, but rather the number of supernodes of the contour tree of the mesh, As we can see in Table I the number of supernodes in most tests is roughly an order of magnitude smaller than the number of regular nodes. Even though a serial method would have sufficed in some tests the need for parallelization will become even more apparent in the future with data sets with more topological complexity or sampling noise.

Furthermore (as pointed out by a reviewer) there are many practical situations, such as time varying domains or ensemble runs, where multiple contour trees need to be computed. For each contour tree we may need multiple branch decompositions if we do not know which geometric measure would be most useful beforehand. Those computations add up and any speedup over a serial implementation with optimal work complexity is valuable. Finally when accelerator devices such as GPUs are used for contour tree computation our parallel implementation allows us to avoid the high cost of inter-device data transfers to CPU for secondary computations.

B. Feature Significance

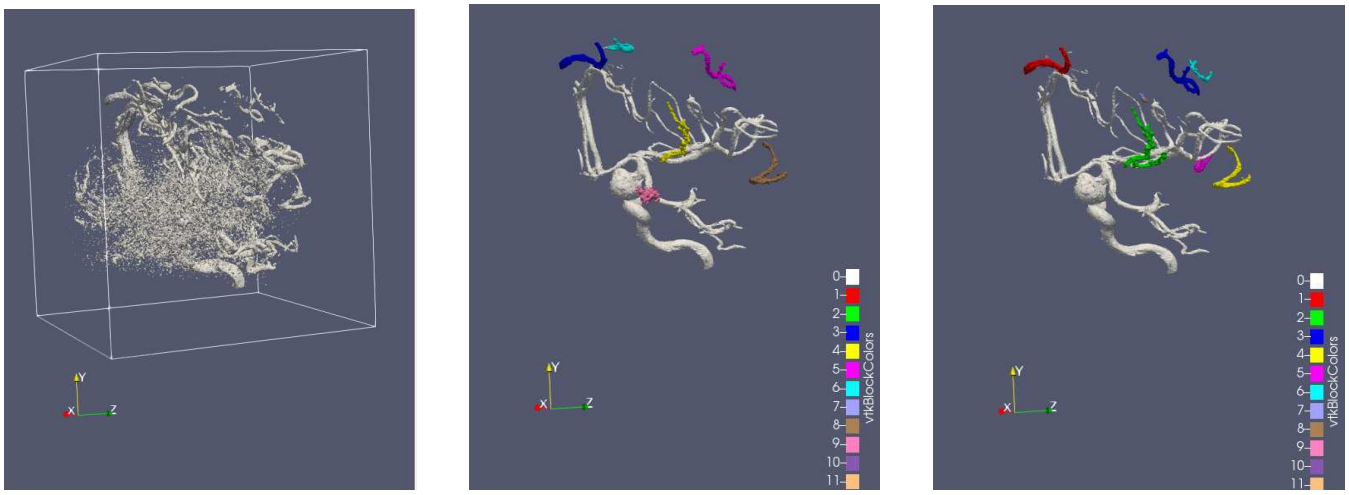
In this subsection we'll consider how the difference between our subtree height decomposition and the standard branch height decomposition impacts feature selection. In Table II the two branch decompositions typically differ in only a small number of branches. Moreover, we know from Section III that the two are identical in contour trees with no W-structures (i.e. those with W diameter of 2 or less. In the table, we see that this is the case, and that in fact, the smallest W-diameter where different decompositions emerge is 5.

Figure 9 shows the effect of choosing the top 10 contours from the aneurism data set, suppressing noise components. Here, both volume and height choose similar top 10 contours. In general, as before [9] volume can be more effective than height, but not always.

In some data sets, the standard branch decomposition is less effective than our new parallel-friendly subtree height decomposition. In Figure 10 we show the result of choosing the top 20 features with the two methods. A large boxy object is visible when subtree height is used, but not when branch

Dataset	Dimensions	Contour Tree Supernodes	Compute Tree seconds	Hypersweep seconds	Branch Decomp seconds	Ratio HS / CT	Ratio BD / CT
Hydrogen Atom	128x128x128	13,038	0.399	0.001	0.025	0.33%	6.47%
Aneurism	256x256x256	65,625	2.793	0.003	0.039	0.12%	1.39%
Bonsai	256x256x256	192,067	3.153	0.007	0.072	0.23%	2.30%
WarpX_E_x	6791x371x371	288,807	317.191	0.005	0.055	0.01%	0.01%
Asteroid	500x500x500	881,831	23.160	0.018	0.258	0.08%	1.11%
Backpack	512x512x373	7,441,922	27.990	0.118	1.431	0.42%	5.11%
Spathorhynchus	1024x1024x750	44,554,912	330.926	0.459	7.299	0.13%	2.20%
Kingsnake	1024x1024x795	55,778,125	268.833	0.589	8.887	0.21%	3.30%
Pawpawsaurus	958x646x1088	89,117,386	352.491	0.979	13.841	0.27%	3.92%
GTopo30 at 0.03125	675x1350	72,276	0.236	0.002	0.014	0.98%	6.21%
GTopo30 at 0.0625	1350x2700	271,772	0.735	0.004	0.036	0.65%	4.95%
GTopo30 at 0.125	2700x5400	991,480	2.571	0.004	0.036	0.18%	1.41%
GTopo30 at 0.25	5400x10400	3,579,117	10.387	0.012	0.108	0.12%	1.04%
GTopo30 at 0.5	10800x21600	12,688,670	44.054	0.038	0.353	0.08%	0.80%
GTopo30 at 1.0	21601x43201	36,912,523	172.301	0.381	3.981	0.22%	2.31%

TABLE I: Once the contour tree and hyperstructure have been computed, hypersweeps to compute secondary properties are highly efficient, adding less than 1% extra time. Our modified branch decomposition, which uses multiple hypersweeps, is a negligible additional cost. Note that the number of supernodes and timings for all data sets differ from the ones reported in [6] because we are using marching cubes connectivity.



(a) Isosurface of the Aneurism data set. (b) Feature extraction based on height. (c) Feature extraction based on volume.

Fig. 9: In an isosurface of the scalar field a) we can observe a lot of sampling noise. To remove the noise we use branch decompositions based on height b) and volume c). Both branch decompositions pick out a similar set of features with varying significance ranking.

height is used. The relevant structures in the contour tree are the six illustrated branches (out of over $3,000,000$ total branches). Notice that the W-structure rooted at $0b$ means that the standard branch decomposition treats this feature as less important, but the new subtree height decomposition, which looks at the height of the entire subtree, displays it.

This does not indicate that the branch height decomposition is invalid, merely that it is imperfect, and that the subtree height is similar and similarly imperfect. However, the new height decomposition is easier to compute in parallel, which is worth having.

VIII. Conclusions and Future Work

We have now completed the first stage of our research into *in situ* topological analysis - the construction and implementation in data-parallel of the full set of algorithms needed to apply topological analysis at scale on a single computer. This involved the initial work on the PPP algorithm [11], the extension to fully augmented contour trees [6], the implementation of geometric measures, simplification and branch decomposition, and of integration with the Cinema database, including single contour extraction.

As part of this, we introduced the *hypersweep* - a data-parallel method for computing properties in contour trees, based on parallel tree contraction. We implemented the hypersweep in the open source VTK-m library and used it to develop a proof of concept *in situ* visualization pipeline using the Cinema database.

Our main research focus in the future, however, will be to continue the task of scaling up topological analysis by developing hybrid algorithms for use on distributed clusters of massively multicore data parallel nodes, such as exemplified by the Summit supercomputer.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National

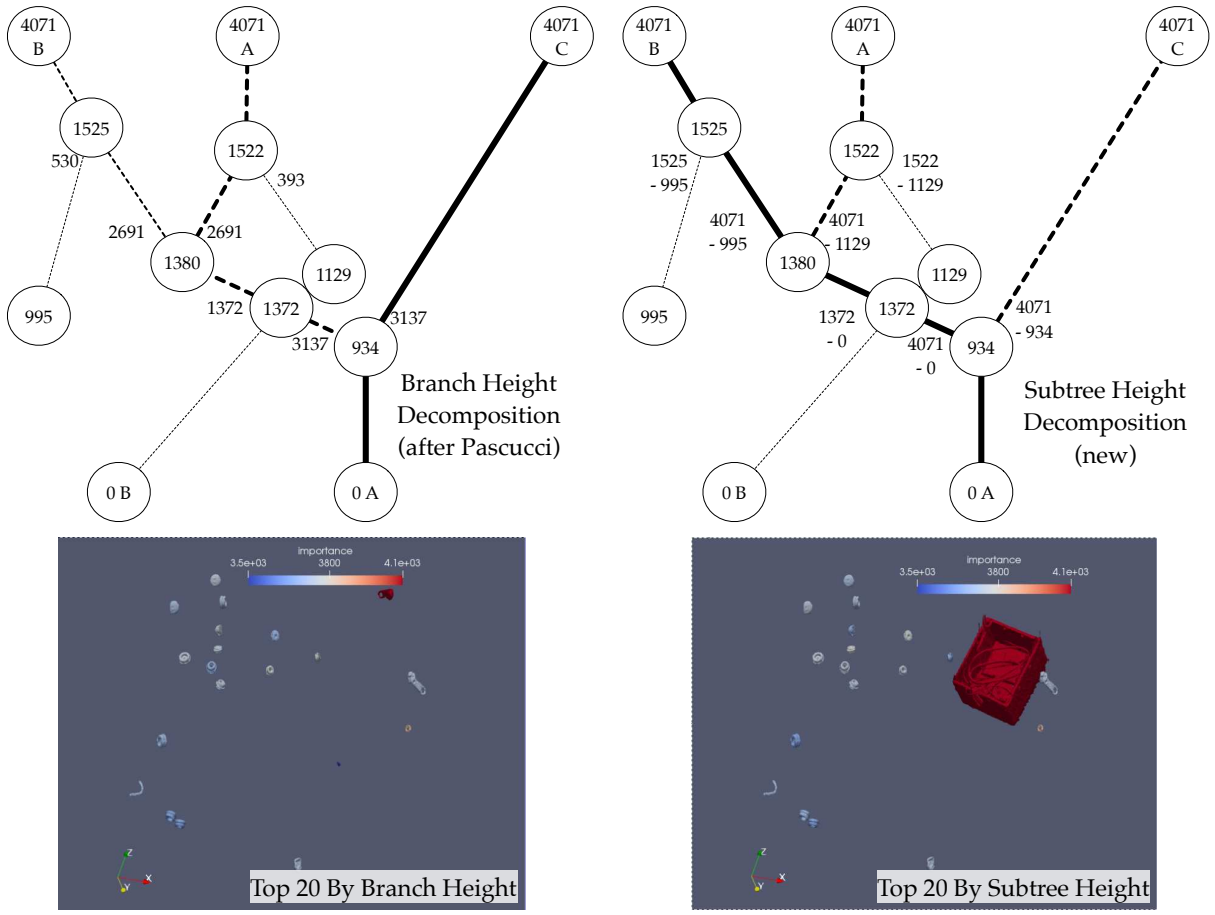


Fig. 10: W-structures in the Backpack data set. Because of a W-structure ending in 0*b*, the left subtree at 934 has a larger overall height than the right subtree, giving a different branch decomposition than Pascucci's [27]. On the right: the top 20 features chosen with each method. While the standard branch decomposition detects the box as feature 39, the subtree height decomposition works better in this instance.

Dataset	Branches	W Diam	Difference	
shockwave	333	3	0	0.0000%
marschner_lobb	810	4	0	0.0000%
neghip	976	4	0	0.0000%
hydrogen_atom	6,532	4	0	0.0000%
aneurism	33,139	4	0	0.0000%
bonsai	96,993	5	8	0.0082%
tooth	151,302	5	4	0.0026%
statue_leg	223,469	6	13	0.0058%
foot	444,616	7	44	0.0099%
mri_ventricles	1,159,963	6	77	0.0066%
skull	1,130,490	7	155	0.0137%
backpack	3,813,085	7	315	0.0098%

TABLE II: Differences from the standard branch decomposition [27]. Both decompositions have the same number of branches, but some leaves can be paired differently. This is due to differences between branch height and persistence in the presence of W-structures [17].

Nuclear Security Administration under Contract No. DE-AC02-05CH11231 to the Lawrence Berkeley National Laboratory and under Award Number 14-017566 to the Los Alamos National Laboratory, and subcontract 7452335 to the University of Leeds. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231. The first author was supported by a scholarship from the School of Computing at the University of Leeds. We thank Jean-Luc Vay and Maxence Thevenet for making the WarpX dataset available.

References

[1] A. Acharya and V. Natarajan. A Parallel and Memory Efficient Algorithm for Constructing the Contour Tree. In *Proceedings of the 2015 IEEE Pacific Visualization Symposium (PacificVis)*, pages 271–278, Apr. 2015.

[2] J. Ahrens, S. Jourdain, P. O'Leary, J. Patchett, D. H. Rogers, and M. Petersen. An Image-Based Approach to Extreme Scale in Situ Visualization and Analysis. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434, 2014.

[3] T. Biedert and C. Garth. Contour Tree Depth Images For Large Data Visualization. In C. Dachsbacher and P. Navrátil, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2015.

[4] A. Bock, H. Doraiswamy, A. Summers, and C. Silva. TopoAngler: Interactive Topology-Based Extraction of Fishes. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):812–821, 2018.

[5] R. L. Boyell and H. Ruston. Hybrid Techniques for Real-time Radar Simulation. In *Proceedings, 1963 Fall Joint Computer Conference*, pages 445–458. IEEE, 1963.

[6] H. Carr, O. Rübel, G. H. Weber, and J. Ahrens. Optimization and Augmentation for Data Parallel Contour Trees. In submission.

[7] H. Carr and J. Snoeyink. Representing Interpolant Topology for Contour Tree Computation. In H.-C. Hege, K. Polthier, and G. Scheuermann, editors, *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 59–74. Springer, 2009.

[8] H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in All Dimensions. *Computational Geometry: Theory and Applications*, 24(2):75–94, 2003.

[9] H. Carr, J. Snoeyink, and M. van de Panne. Flexible Isosurfaces: Simplifying and Displaying Scalar Topology Using the Contour Tree. *Computational Geometry: Theory and Applications*, 43(1):42–58, 2010.

[10] H. Carr, G. H. Weber, C. Sewell, and J. Ahrens. Parallel Peak Pruning for Scalable SMP Contour Tree Computation. In *6th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 75–84, 2016.

[11] H. Carr, G. H. Weber, C. Sewell, O. Rübel, P. Fasel, and J. Ahrens. Scalable Contour Tree Computation by Data Parallel Peak Pruning. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, November 2019.

[12] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 454–463, 2000.

[13] H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[14] J. Gibbons, W. Cai, and D. B. Skillicorn. Efficient Parallel Algorithms for Tree Accumulations. *Science of Computer Programming*, 23(1):1 – 18, 1994.

[15] C. Gueunet, P. Fortin, and J. Jomier. Contour Forests: Fast Multi-threaded Augmented Contour Trees. In *6th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 85–92, Oct 2016.

[16] C. Gueunet, P. Fortin, J. Jomier, and J. Tierny. Task-based Augmented Merge Trees with Fibonacci Heaps. In *2017 IEEE 7th Symposium on Large Data Analysis and Visualization (LDAV)*, pages 6–15, Oct 2017.

[17] P. Hristov and H. Carr. W-Structures in Contour Trees. In *Proc. of TopoInVis*, 2019.

[18] 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 *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031, Nov. 2014.

[19] J. Lukasczyk, E. Kinner, J. Ahrens, H. Leitte, and C. Garth. VOIDGA: A View-Approximation Oriented Image Database Generation Approach. In *8th IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 12–22, 2018.

[20] S. Maadasamy, H. Doraiswamy, and V. Natarajan. A Hybrid Parallel Algorithm for Computing and Tracking Level Set Topology. In *High Performance Computing (HiPC), 2012 19th International Conference on*, pages 1–10. IEEE, Dec. 2012.

[21] G. L. Miller and J. H. Reif. Parallel Tree Contraction Part 1: Fundamentals. In S. Micali, editor, *Randomness and Computation*, pages 47–72. JAI Press, Greenwich, Connecticut, 1989. Vol. 5.

[22] K. Moreland, C. Sewell, W. Usher, L.-T. Lo, J. Meredith, D. Pugmire, J. Kress, H. Schroots, K. L. Ma, H. Childs, M. Larsen, C. M. Chen, R. Maynard, and B. Geveci. VTK-m: Accelerating the Visualization Toolkit for Massively Threaded Architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May 2016.

[23] D. Morozov and G. Weber. Distributed Merge Trees. *ACM SIGPLAN Notices*, 48(8):93–102, 2013.

[24] 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, 2014.

[25] P. O'Leary, J. Ahrens, S. Jourdain, S. Wittenburg, D. H. Rogers, and M. Petersen. Cinema Image-based in situ Analysis and Visualization of MPAS-ocean Simulations. *Parallel Computing*, 55:43 – 48, 2016. Visualization and Data Analytics for Scientific Discovery.

[26] V. Pascucci and K. Cole-McLaughlin. Parallel Computation of the Topology of Level Sets. *Algorithmica*, 38(2):249–268, 2004.

[27] V. Pascucci, K. Cole-McLaughlin, and G. Scorzell. Multi-Resolution Computation and Presentation of Contour Trees. In *Proceedings of the IASTED conference on Visualization, Imaging and Image Processing (VIIP 2004)*, pages 452–290, 2004.

[28] P. Rosen, J. Tu, and L. A. Piegl. A Hybrid Solution to Parallel Calculation of Augmented Join Trees of Scalar Fields in any Dimension. *Computer-Aided Design and Applications*, 15(4):610–618, 2018.

[29] D. Schneider, A. Wiebel, H. Carr, M. Hlawitschka, and G. Scheuermann. Interactive Comparison of Scalar Fields Based on Largest Contours with Applications to Flow Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1475–1482, 2008.

[30] J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics (Proc. of IEEE VIS)*, 2017.

[31] G. Weber, S. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-Controlled Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):330–341, March/April 2007.

[32] G. Wyvill, C. McPheeters, and B. Wyvill. Data Structure for Soft Objects. *Visual Computer*, 2:227–234, 1986.