Lawrence Berkeley National Laboratory
LBL Publications

Title
Distributed Task-Parallel Topology-Controlled Volume Rendering

Permalink
https://escholarship.org/uc/item/69v11647

ISBN
9783030834999

Authors
Sohns, Jan-Tobias
Weber, Gunther H
Garth, Christoph

Publication Date
2021

DOI
10.1007/978-3-030-83500-2_4

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

Distributed Task-Parallel Topology-Controlled Volume Rendering

Jan-Tobias Sohns, Gunther H. Weber and Christoph Garth

Abstract Topology-controlled volume rendering has proven to be a useful tool for exploration of volumetric data by highlighting the global, high-level structure of data sets. However, topological analysis is difficult to parallelize on distributed memory systems – and thus to utilize for in situ visualization – due to the global nature of topological descriptors.

This chapter presents and evaluates a task-parallel formulation of topology-controlled volume rendering applicable to visualization of large scalar field data. It evaluates previous efforts towards parallel topology extraction and introduces a distributed computation schema for augmented contour trees. Through data partitioning into rectilinear blocks, the algorithm is designed to be in-situ suitable. The use of a task-parallel framework aims at latency hiding and dataflow-specific scheduling. It thereby also allows for combining contour tree computation and subsequent volume rendering. The technique divides the scalar field with separate transfer functions according to the branch decomposition of the full data set while each local block only has to keep track of its own vertex augmentation. Beyond describing the approach and its implementation in the task-parallel framework HPX, initial experiments on scaling behaviour are presented.

1 Introduction

Computer simulation techniques have become ubiquitous in the investigation of both scientific and engineering problems. Owing to increased computational ability,

Jan-Tobias Sohns
TU Kaiserslautern, e-mail: j_sohns12@cs.uni-kl.de

Gunther H. Weber
Lawrence Berkeley National Laboratory, e-mail: ghweber@lbl.gov

Christoph Garth
TU Kaiserslautern, e-mail: garth@cs.uni-kl.de

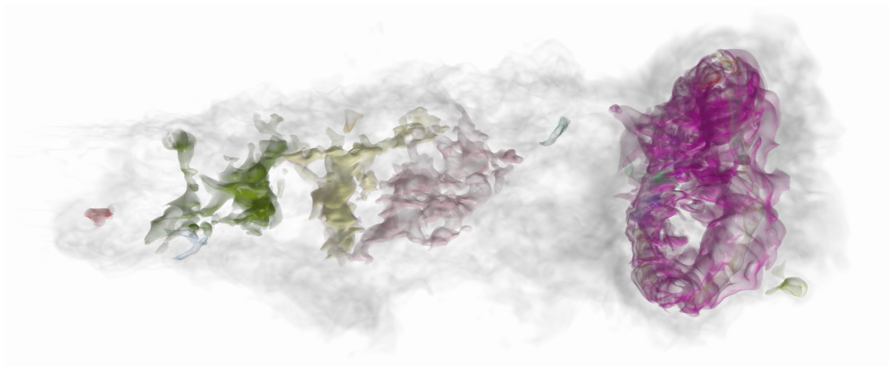


Fig. 1: Distributed topology-controlled volume rendering of the jet data set. Branch pruning minimal persistence is set to 3 (max 16.63).

such simulations have outgrown the ability to retain the data produced by them for post hoc analysis, and thus a re-thinking of established post-processing visualization work flows is necessary. In situ techniques that derive analysis artifacts from the data while it is produced have shown promise in this context. In their most extreme form, visualization images are generated in situ and examined post hoc. Choosing an appropriate set of images allows sufficient flexibility in exploring simulation results [1].

Concurrently, the ever-increasing complexity of data describing real-world problems has necessitated the development of efficient abstraction and reduction techniques. Among these, topological feature extraction relies on a solid mathematical foundation to achieve these goals. In particular, topology-controlled volume rendering [2] can be employed to automatically define transfer functions for volume rendering directly on the structure of the data, as represented by the contour tree [3]. Topology-controlled volume rendering was previously restricted to small data sets, due to the global nature of contour trees that make them hard to compute in a distributed manner.

In this book chapter, we report on a proof of concept distributed pipeline for topology-controlled volume rendering. It is aimed at making bigger, possibly distributed data sets accessible to the technique. To leverage latency-hiding and dataflow-based asynchronous scheduling, our implementation is based on a task-parallel formulation of topological analysis, transfer function generation, and volume rendering. We present early findings on the performance and scalability of our implementation. The experiments are designed to get a first impression of the applicability of topology-controlled volume rendering for bigger data sets. Further investigation in more extensive settings has to be done in the future to get a complete understanding of the practical capability.

The presented work relies heavily on topological descriptors such as the contour tree [4], merge tree [3] and branch decomposition [5]. The reader is referred towards the proposing papers for in-depth information on these concepts. Further on, the augmented version of the trees is used as in [6]. The augmentation is a segmentation of data in the spatial domain, where each data point is affiliated with an arc in the corresponding tree.

The manuscript is structured as follows: After briefly reviewing related work in Section 2, we present our approach in Section 3 and implementation in Section 4. Results and benchmarks on typical datasets are presented in Section 5, and we conclude and reflect on further improvements in Section 6.

2 Related Work

Topology-controlled volume rendering is first presented by Weber et al. [2]. They use the vertex affiliation with branches in the augmented branch decomposition of the contour tree to specify individual user-designed transfer functions for each branch. Weber et al. also realize that a simplification of the topological descriptor is unavoidable to prevent a cluttering of the scene. The visual results are satisfying and promise a powerful analysis technique for 3D scalar data. However, the drawback of their algorithm is the sequential nature of the contour tree computation, which dominates the runtime. It is calculated separately with the algorithm introduced by Carr et al. [3].

Parallel contour tree computation was examined before the practical applications arose. Pascucci et al. [7] describe a parallel algorithm that can also be used in a distributed setting. Based on a divide-and-conquer formulation, it lists a merge routine for merge trees of two adjacent data subsets. This approach still forms the basis for most subsequent algorithms as well as ours. Although vital for many use cases, the initial method does not keep track of the vertex augmentation of the contour tree.

Gueunet et al. [6] present a parallel algorithm that creates an augmented contour tree by dividing the data in range space. Dividing data sets in range space is uncommon in distributed settings, since simulations and measurements are usually allocated in domain space. This makes their algorithm suitable if the data can be divided in range space without too much effort, yet is not fitted for domain space distribution. Nonetheless, it provides a fast parallel computation in a shared-memory system and shows the need of topological simplification again.

Another efficient approach was taken by Carr et al. [8], who demonstrate a highly thread-parallel algorithm to compute contour trees whilst retaining correct augmentation. More fast and efficient parallel algorithms [9] have been proposed recently that successfully focus on shared-memory computation, which this work tries to overcome.

In the wake of the shift towards using graphical processing units for scientific computations, Rosen et al. [10] published an augmented merge tree construction algorithm using OpenCL on a GPU. It outperforms the CPU version by a magnitude in their benchmarks on 2D scalar fields. Unfortunately, it is mentioned in the article that the computational benefits are not necessarily carried over to an extension into 3D.

Morozov et al. [11, 12] specifically target their algorithm on distributed systems and achieved competitive results. They allow for a distributed computation and representation of merge and contour tree, which can handle specific topological requests. These requests do not suffice for the pursued volume rendering and the algorithm does not produce the desired augmented contour tree.

Another important addition to this field was made by Bremer et al. [13], who uses a streaming fashion to gather the merge tree and its augmentation. Their sequential algorithm serves as inspiration for updating the augmentation through a search process and performing on-the-fly simplification on the topological complexity.

Landge et al. [14] focus on the augmentation of very high or low function values and compute only the locally relevant parts of a merge tree on distributed blocks. Their analysis is fast, can be computed in situ and easily highlights smaller features. The focus on local features of a distributed merge tree is advantageous, since it eases the distributed computation and is sufficiently accurate for most analysis purposes.

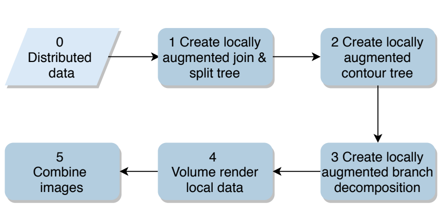
Recent advances in parallel programming have shown that the algorithm of Landge et al. [14] can be implemented efficiently in task-parallel frameworks. Petruzza et al. [15] show that the *Legion* [16] framework is faster for low core counts but does not exhibit good scalability, while using *Charm*++ [17] to schedule the tasks runs and scales just as well as the original data-parallel implementation. Therefore, local merge tree analysis can be done in distributed task-parallel settings.

Some techniques like topology-controlled volume rendering [2] rely on the contour tree and a branch decomposition thereof to assign transfer functions consistently to branches. However, it is not immediately obvious to construct a complete contour tree from distributed merge trees. Hence, both full merge trees, namely join and split tree, are required to cover the whole value range with a branch decomposition. The work in this chapter aims to overcome previous distributed restriction on local merge tree analysis. The full topology-controlled volume rendering pipeline is presented for distributed settings, facilitating the approach for larger data.

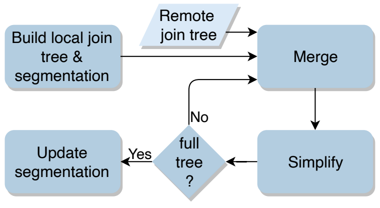
3 System Design

As other works on this topic already recognized, the size of currently produced data sets warrants splitting them up into smaller blocks. In some cases data is already split up by its production process. The global manner of contour trees leads to high communication effort between these distributed data blocks during their computation. We aim to overcome this challenge by using a task-parallel setup that allows for defining complex dependency structures and arbitrary block granularity to suit most simulation outputs.

Due to noise or necessary symbolic perturbation the topology of data sets can be too complex to read useful information out of a volume rendering. In many cases expensive computation and communication costs can be cut down by simplifying



(a)



(b)

Fig. 2: Work flow per data block. (a) complete pipeline, references to Section 3 (b) locally-augmented join tree assembly, explained in sections 4 - 4.3

the tracked topological descriptors in early stages of their calculation without losing essential features. In the previous chapter current methods are described, which struggle either with data size, topological complexity or missing augmentation. Our approach works on arbitrarily distributed blocks of regular data according to the schema shown in Figure 2a.

The key idea is to generate consistent global topological information on all blocks while keeping track of the local augmentation only and reducing the topological complexity early. The data is assumed to already be distributed to blocks (0). On each block the locally-augmented, simplified merge trees are constructed first (1). The merge trees are then combined on each block following the established scheme of Carr et al. [3] to create a locally-augmented contour tree (2). Subsequently, the contour tree is transformed into a locally-augmented branch decomposition covering the topology of the whole scalar field (3). This contour tree is identical over all blocks, whereas each block keeps track of a version with their local augmentation. It is now possible to define global transfer functions that are identical for all blocks, thereby guaranteeing a coherent color mapping.

Assuming the transfer functions are assigned automatically, the blocks can be rendered individually as soon as their precomputation finishes (4). In line with the task-parallel principle, rendering time can be hidden within preprocessing time. To conclude, the resulting images are composited regarding their depth level yielding the topology-controlled volume rendering of a scalar field (5). In the following the algorithm of creating a locally-augmented simplified merge tree will be elaborated in detail.

4 Implementation

Initially, all blocks are assumed to be filled with a coherent fragment of the scalar field with a single layer of ghost cells. For each block, the locally-augmented, simplified merge trees are assembled by first computing the augmented merge trees of the local data. Any sequential algorithm can be used. The established contour tree construction algorithm [3], executed in a later phase, requires coinciding nodes in both merge trees.

Intending to reduce communication, nodes are exchanged mutually inside each block on a local stage. Then, local trees are progressively combined with unaugmented trees of tree-neighboring blocks following a reduction schema extended from [14].

The procedure for join trees is illustrated in Figure 2b, split trees follow analogously. A recurring sequence of merging two trees and simplifying the result is performed. The methods will be called *Merge* and *Simplify* and are repeated on all blocks until the local trees cover the topology of the full data set. After that, the augmentation is corrected accordingly with the *Update* method. The functions as well as the remaining pipeline steps will be explained in chronological order hereinafter.

4.1 Merge

Whenever the merge procedure is called, the currently present unaugmented merge tree is merged with a remote one via the algorithm Landge et al. [14] call *Join Routine*. Their code follows the first proclaimed algorithm for the joining of merge trees [7] closely, yet is more intuitively formulated. The combined tree is iteratively formed by traversing the individual trees in value order starting from the lowest valued leaf.

4.2 Simplify

All nodes of the new tree are checked to decide whether the tree can be simplified by removing them. A node is obsolete when it has become regular through the merge process, recognizable through a degree of two. This was already proposed in the first parallel algorithm [7]. Regular nodes are eliminated without losing topological information.

As mentioned before, we want to reduce the branch decomposition to a distinguishable number of branches for rendering. The *Merge* function's runtime depends linearly on the size of the individual trees. Therefore, the usual chronology of simplifying the topology at the final stage, e.g. the branch decomposition, induces linear overhead in the magnitude of global topology per processor for each merge. However, reducing the tracked topological complexity as early as possible should lower the workload to a small linear overhead of simplified topology per merge. Branch pruning [18, 2] is employed here as a simple and well-established simplification tool that works on both topological structures. To keep the merge tree simplification comprehensible, it has to conducted as if branches were pruned in the branch decomposition.

The idea of branch pruning is removing less relevant branches from the tree. Persistence [19] is chosen as the relevance measure, though other measures work just as well as long as they can be tracked throughout the merging process. Consistent with the branch decomposition the more persistent component is considered to survive if two components merge, while the less persistent component is annexed.

Based on the previous line of reasoning, leaves with a persistence value lower than a predefined threshold are removed from the tree if they are adjacent to a saddle. Exceptions are the most persistent ones in comparison to their siblings, which represent branches that extend above the saddle node. Removing them would prune a branch partly, yet only complete branches are supposed to be eliminated. Further on, boundary nodes regarding the new tree and nodes relevant in opposing join/split tree are retained for stitching in later phases. The conditions hold for regular vertices as well.

To eliminate chains of removable nodes, the simplification routine is looped until no more nodes can be removed. In our benchmarks the single digit loop iterations accrued minuscule overhead.

4.3 Update

After the addition and removal of nodes through joining and simplification, the mapping of local vertices onto arcs can become incorrect. An update method is deployed to achieve the correct augmentation according to the new join tree without recomputing the labels from scratch.

When removing nodes, a new label candidate is saved for every removed node. For regular nodes, the candidate is the predecessor of the node in leaf direction. For leaf nodes, the governing saddle is chosen as the candidate, which corresponds to the parent branch in the branch decomposition. If the candidate is removed as well, the recursive candidates are traversed until an existing node is reached. Path compression is implemented to shorten candidate chains.

4.4 Branch Decomposition Assembly

The contour tree and consequently the branch decomposition can be sequentially assembled for each block individually with any sequential algorithm. With the presented algorithm design both block size and tree sizes can be reduced to low complexity through parameters. Thus, the construction algorithm runs on small data sizes per block and all blocks can run in parallel. Hence, a short runtime is expected for the transformation performed by the *libtourtre* [20] library. The augmentation with local vertices of the merge trees is implicitly carried over to the branch decomposition.

4.5 Transfer Function Assignment

The augmentation of the data points onto branches of the branch decomposition allows assigning a specific transfer function to each branch. The original topology-controlled volume rendering algorithm [2] stopped after computing the topology to allow the user to define custom transfer functions on each branch. Thereby it is possible to create customized functions and yield pleasing visual results. Since fine-tuning the transfer functions requires deep domain knowledge and many iterations, it breaks the algorithm into two separate steps that have to be started individually.

It would be beneficial to the presented method, if these parts could run continuously through forgoing the interactive break. Therefore, a basic automatic transfer function assignment is chosen for exemplary results which imitates colored isosurfaces. A distinguishable [21] color is used per branch with an opacity peak at the saddle value. Overall opacity is increased with branch depth to allow visible inclusion relationships. The main branch is colored in light grey with little opacity to ease perceiving branch location in the volume.

Since we are aware that automatic transfer function assignment is a complex topic with numerous sophisticated solutions, this part leaves room for further work. Exemplary, the implementation also allows user-defined transfer functions as well as the choice of commonly used predefined ones.

4.6 Rendering

Whenever the transfer functions are defined for a block, it can be rendered. To stay in line with the distribution of data blocks, the rendering implementation is drawn upon a task-parallel volume renderer [22] augmented with a branch lookup per sample [2]. The approach divides work in both screen and object space. Each block is rendered individually with the resulting images being composited in depth order afterwards. Additionally, the image is split up into rectilinear tiles allowing further steering of task-size. These tiles are concatenated to create the full resolution image.

5 Results

The following chapter will shine a light on questions arising from this approach. The algorithmic idea will be validated first on shared memory. Then benchmarks are run that examine performance for changing input size as well as resources in distributed settings. In consideration of ever-growing data size, the focus is set on scaling properties.

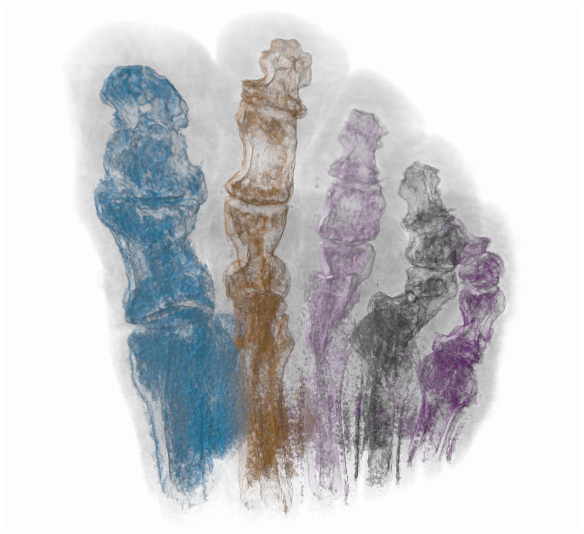


Fig. 3: Distributed topology-controlled volume rendering of the ’CT bones’ data set of TTK [23]. Branch pruning minimal persistence is set to 171 (max 255).

5.1 Experimental Design

The implementation is tested on the Cori supercomputing system of NERSC. Per node two Intel Xeon E5-2698 v3 Haswell processors are provided with a combined total of 32 cores at 2.3GHz and 128GB of memory. For the test cases, a combustion simulation of a jet turbine with native 256x512x256 data points is used. To facilitate increasing data size, the jet data set is super-sampled for higher resolution runs. Resampling a data set is suboptimal for scaling experiments as it theoretically will keep topological complexity near constant, since the number and relative position of features remain roughly the same. However, symbolic perturbation can influence the precise contour tree structure to varying degree and it can be initially examined if topology-controlled volume rendering can be applied for bigger data, even if further experiments are necessary to determine practical applicability.

Before starting with the analysis, a few preliminary considerations are taken first. This algorithm is most useful when analyzing huge distributed data sets such as the results of a distributed simulation. Therefore, it is assumed that the preceding application split up the data in ready to use blocks distributed over the hardware. This is mirrored by assuming that blocks are already loaded into the memory partitions. For now, data is read from disk, though this set-up is a precursor for the in situ case.

Dependencies in the task-parallel framework can be set in a way that allows intermingling of precomputation and rendering phase. However, the speedup through overlapping both phases has shown to be insignificant in comparison to the total runtime and the usual fluctuations of supercomputing environments. Hence, timings are taken separately with global barriers before and after each phase to ensure they don't slow each other down.

5.2 General Observations

As mentioned before, the biggest novelty of this approach lies in the creation of an augmented branch decomposition from distributed data blocks. Hence, the presented benchmarks address the topology computation. Further analysis towards the distributed rendering can be found in the proposing paper [22]. The implementation differences as covered in Section 4.6 induce a near constant overhead through a branch identification per sample. These changes are not expected to lead to changes in scaling behavior.

The implementation is based on assumptions that come up through observation of previous use cases. Figure 1 and Figure 3 convey a clear visualization of topological features while features are still distinguishable. The simplification assumption seems to hold for two contemporary data sets.

5.3 Algorithm Validation

Before thinking about distributed settings, it needs to be determined if the algorithmic idea of subdivision into blocks combined with early topological simplification does behave as expected in the first place. To examine the algorithm's properties, the implementation is tested in a shared memory setting where constant data is divided into an increasing number of blocks (see Figure 4). The used jet data set spans a value range from 0 to 16.63 and branches with less than 3 persistence are pruned. The final branch decomposition contains 25 branches as rendered in Figure 1.

The examined computation, after which all blocks contain a simplified branch decomposition augmented with their local vertices as well as common transfer functions, will be called DAB (Distributed Augmented Branch decomposition) in the following analysis. Only summary timings are provided here, since the task-based paradigm relies on interleaved execution with latency hiding.

The timings shown in Figure 4 reveal that subdivision into blocks amounts for a significant speedup. This can be attributed to a combination of better work distribution among processors for smaller tasks and the on-the-fly removal of short branches early in the construction process. Consequential, fewer arcs have to be processed in the remainder of the algorithm. Considering that computational resources are kept constant, this means less overall operations have to be done even with more merging steps. On the other hand, after a certain point the additional merging overhead outweighs the speedup from early arc removal and the execution time rises.

It has to be noted that the fastest achieved computation time is on par with other shared memory augmented contour tree algorithms [8] which do not rely on

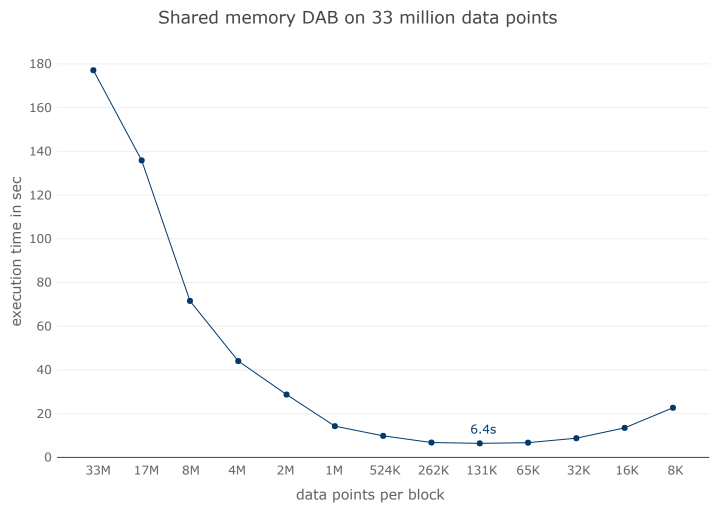


Fig. 4: Computing the DAB on a single Cori Haswell node for the native jet data set with 33M vertices while dividing into different block sizes.

simplification. Therefore, these are preferable for shared memory settings. However, the novel point of this approach is that it allows a subdivision without a shared memory and thus can be applied to much bigger data sets, which will be examined in the following sections.

5.4 Strong Scaling

Strong scaling examines the change in execution time that can be achieved for a fixed problem size by increasing the number of processing units. Multiple measures can be deployed to indicate strong scaling behavior. A test case that inspects strong scaling of the DAB computation was constructed via re-sampling the jet data set to 1024x1024x1024 vertices and benchmarking it on 1 to 128 nodes. With increasing processor count the data set is divided accordingly into more blocks, starting with 128 blocks on a single node.

The tests depicted in Figure 5 showed that increasing hardware availability accelerates the DAB computation in accordance with Amdahl's Law of 95% parallel and 5% sequential parts. If one remembers that the conversion from merge tree to branch decomposition is still done sequentially per block, these results are in expected bounds.

Further on, for a fixed problem size the algorithm does not speed up infinitely with additional resources, but slows down after a certain node count. The overhead through additional merges dominates the cost as in the shared memory test. The

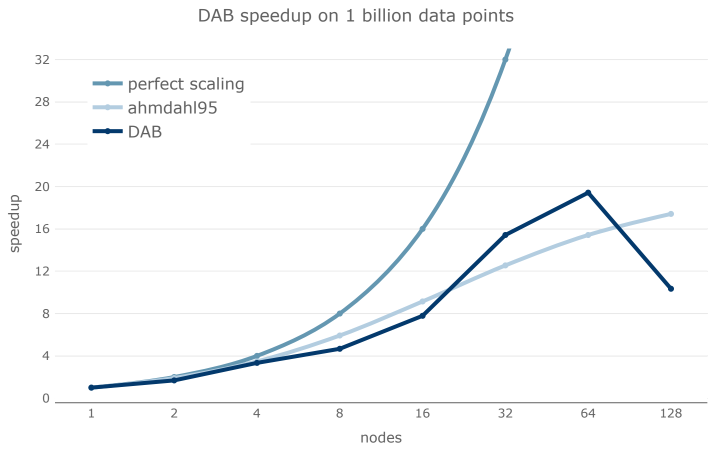


Fig. 5: Speedup for DAB computation on resampled jet data set with 1024x1024x1024 vertices using the single node case as the baseline. Perfect scaling and scaling according to Ahmdahl's Law with 95% parallel program parts is shown in lighter blue.

slowdown through additional blocks and resources is even more extreme here, since network communication is required for each merge of two blocks on different nodes.

5.5 Weak Scaling

One refers to weak scaling when examining the execution time of an algorithm with proportional increases in resources and problem size. Since the algorithm is designed to be applicable to future data sets of increasing size, weak scaling benchmarks are conducted through resampling the jet data set and computing the DAB on increasingly powerful hardware configurations. The baseline is chosen to be a resolution of 64x128x128 data points on a single processor. Processor count and number of data points are doubled in each iteration so that the ratio of approximately 1 million data points per processor remains constant. The last iteration handles 4 billion data points on a 2048x2048x1024 grid. Per processor 8 blocks are created in the single to 16 processors case, 4 blocks for 32 to 256 processors and 2 blocks from 512 processors on. The successive reduction is chosen to minimize communication while still performing local early simplification.

From inspecting Figure 6, it is evident that the execution time increases overall with problem size. The bumps on 32 and 512 processors coincide with stagnant block counts on more resources. Comparing the edge cases, increasing data size 4096-fold while keeping relative resources constant triples the execution time. Communication accounts for significant overhead on rising problem size and resources, yet for the

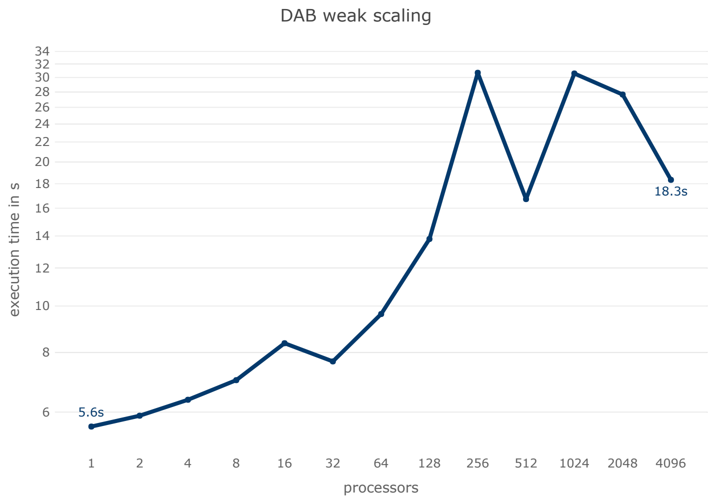


Fig. 6: Weak scaling for DAB computation keeping relative workload per processor constant through resampling the jet data set. Each processor handles ~1 million data points.

examined cases the overhead is sufficiently small to suggest that the algorithm can stay feasible under the weak scaling paradigm.

In all experiments the communication imposed a high runtime penalty. As a general guideline, results were best with block counts per processor of 2 or 4 and block sizes of ~1 million data points. This should be taken as a starting point for further experiments on new data sets.

An aspect not mentioned before is that the studies in the scope of this chapter revealed that contour tree algorithms generally require a substantial amount of memory during the computation. Most papers did not comment on the memory requirements and memory efficient algorithms exist already [13, 24]. However, computing the contour tree or a directly related topological descriptor for a 3D data set of 4 billion data points is challenging without running out of memory for algorithms not specifically designed for it. This chapter presents such an algorithm allowing analysis of huge data sets through on-the-fly topological simplification and minimal data duplication. If the rising communication overhead and the reduction on a small number of topological features is still practical on future data sets is up for future work.

6 Conclusion

The size of contemporary data sets is rising rapidly with no end in sight. Extracting topological features on them get increasingly time-consuming. Simultaneously, the processing power available in high performance compute clusters is growing annually. The idea suggests itself to leverage the distributed systems for preprocessing and rendering of 3D data sets. In this chapter, a task-parallel distribution is chosen to present a possible pipeline.

To put the presented work into perspective, the advantages and drawbacks of current parallel contour tree algorithms are compared first. The analysis revealed that while a plethora of shared memory solutions exist, distributed algorithms are scarce and come with restrictions.

Thereupon, an algorithm is proposed that allows distributed topology-controlled volume rendering. It relies on eliminating small topological features on-the-fly in a distributed construction process. It continues by merging trees so that the full contour tree is known globally while the augmentation is only locally present. Completing the pipeline, a possible distributed rendering technique is provided.

The algorithm was implemented in a task-parallel framework to provide portability and flexible parallelism granularity. To examine the effects of granularity as well as scaling performance, extensive benchmarks were run and analyzed.

The results suggest existing yet limited strong scaling ability. Benchmarks where data size and processor count was raised accordingly revealed that the algorithm can handle larger data sets with small performance decreases. The algorithm's main speedup as well as its limitation stems from the assumption that one is interested in the simplified topology only.

All things considered, this chapter presented a distributed augmented contour tree algorithm that exhibits limited strong and promising weak scaling capabilities. Whether this solution is sufficient for practical use cases or the simplification process can be circumvented in distributed settings is ground for future work.

Acknowledgements This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-AC02-05CH11231 and used resources of the National Energy Research Scientific Computing Center (NERSC), which is a DOE Office of Science User Facility. We thank the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 252408385 - IRTG 2057 for providing accommodation during the research stay at Lawrence Berkeley National Lab (LBNL).

References

1. J. P. 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 *Proc. of the Int. Conf. for High Perf. Comp., Networking, Storage and Analysis*, pp. 424–434, IEEE Press, 2014.
2. G. H. Weber, S. E. Dillard, H. A. Carr, V. Pascucci, and B. Hamann, "Topology-controlled volume rendering," *Trans. Vis. Comput. Graphics*, vol. 13, pp. 330–341, 2007.

3. H. A. Carr, J. Snoeyink, and U. Axen, “Comp. contour trees in all dimensions,” in *Proc. of the Eleventh Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 918–926, Society for Industrial and Applied Mathematics, 2000.
4. R. Oostrum, V. Kreveld, C. Bajaj, V. Pascucci, and D. Schikore, “Contour trees and small seed sets for isosurface traversal,” *13th ACM Symp. on Computational Geometry*, 1999.
5. G. Scorzelli, V. Pascucci, and K. Cole-McLaughlin, “Multi-resolution computation and presentation of contour trees,” *IASTED Conf. on Vis., Imaging and Image Process.*, 2004.
6. C. Gueunet, P. Fortin, and J. Jomier, “Contour forests: Fast multi-threaded augmented contour trees,” *6th IEEE Symp. on Large Data Analysis and Visualization*, pp. 85–92, 2016.
7. V. Pascucci and K. Cole-McLaughlin, “Parallel computation of the topology of level sets,” *Algorithmica*, vol. 38, pp. 249–268, 2003.
8. H. A. Carr, G. H. Weber, C. M. Sewell, and J. P. Ahrens, “Parallel peak pruning for scalable smp contour tree computation,” *6th IEEE Symp. on Large Data Analysis and Visualization*, pp. 75–84, 2016.
9. C. Gueunet, P. Fortin, J. Jomier, and J. Tierny, “Task-based augmented merge trees with fibonacci heaps,” in *7th IEEE Symp. on Large Data Analysis and Visualization*, pp. 6–15, 2017.
10. 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*, vol. 15, pp. 610–618, 2018.
11. D. Morozov and G. H. Weber, “Distributed merge trees,” in *Proc. of the 18th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pp. 93–102, ACM, 2013.
12. D. Morozov and G. H. Weber, “Distributed contour trees,” in *Topological Methods in Data Analysis and Visualization*, 2014.
13. P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. Day, and J. Bell, “Interactive exploration and analysis of large-scale simulations using topology-based data segmentation,” *IEEE Trans. Vis. Comput. Graphics*, vol. 17, pp. 1307–1324, 2011.
14. A. G. Landge, V. Pascucci, A. Gyulassy, J. Bennett, H. Kolla, J. Chen, and P.-T. Bremer, “In-situ feature extraction of large scale combustion simulations using segmented merge trees,” *Proc. of the Int. Conf. for High Perf. Comp., Networking, Storage and Analysis*, pp. 1020–1031, 2014.
15. S. Petruzza, S. Treichler, V. Pascucci, and P. Bremer, “Babelflow: An embedded domain specific language for parallel analysis and visualization,” in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 463–473, 2018.
16. M. Bauer, S. Treichler, E. Slaughter, and A. Aiken, “Legion: Expressing locality and independence with logical regions,” in *SC ’12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–11, 2012.
17. L. V. Kale and S. Krishnan, “Charm++: A portable concurrent object oriented system based on c++,” in *Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA âĂŹ93, (New York, NY, USA), p. 91âĂŞ108, Association for Computing Machinery, 1993.
18. H. A. Carr, J. Snoeyink, and M. van de Panne, “Simplifying flexible isosurfaces using local geometric measures,” in *Proc. of the Conf. on Visualization ’04*, pp. 497–504, IEEE Computer Society, 2004.
19. H. Edelsbrunner, D. Letscher, and A. Zomorodian, “Topological persistence and simplification,” *Discrete & Computational Geometry*, vol. 28, no. 4, pp. 511–533, 2002.
20. S. Dillard, “libtourtre: A contour tree library.” http://graphics.cs.ucdavis.edu/ sdillard/libtourtre/doc/html/, 2008. accessed 2018-11-06.
21. P. Green-Armytage, “A colour alphabet and the limits of colour coding,” *Color: Design & Creativity*, vol. 5, pp. 1–23, 2010.
22. T. Biedert, K. Werner, B. Hentschel, and C. Garth, “A Task-Based Parallel Rendering Component For Large-Scale Visualization Applications,” in *Eurographics Symp. on Parallel Graphics and Visualization*, The Eurographics Association, 2017.
23. J. Tierny, G. Favelier, J. A. Levine, C. Gueunet, and M. Michaux, “The Topology ToolKit,” *IEEE Trans. Vis. Comput. Graphics*, 2017. https://topology-tool-kit.github.io/.
24. A. Acharya and V. Natarajan, “A parallel and memory efficient algorithm for constructing the contour tree,” in *IEEE Pacific Visualization Symp.*, pp. 271–278, 2015.