Parallel Attribute Computation for Distributed Component Forests

Gazagnes, Simon; Wilkinson, M.H.F.

Published in:
Proceedings of the 29th IEEE International Conference on Image Processing

Publication date:
2022

PARALLEL ATTRIBUTE COMPUTATION FOR DISTRIBUTED COMPONENT FORESTS

Simon Gazagnes[1]*, Michael H.F. Wilkinson*[2]

[1]University of Texas at Austin, Department of Astronomy,
Austin, Texas, United States
[2]University of Groningen, Bernouilli Institute for Mathematics and Artificial Intelligence,
Groningen, The Netherlands

ABSTRACT

Component trees are powerful image processing tools to analyze the connected components of an image. One attractive strategy consists in building the nested relations at first and then deriving the components' attributes afterward, such that the user can switch between different attribute functions without having to re-compute the entire tree. Only sequential algorithms allow such an approach, while no parallel algorithm is available. In this paper, we extend a recent method using distributed memory techniques to enable posterior attribute computation in a parallel or distributed manner. This novel approach significantly reduces the computational time needed for combining several attribute functions interactively in Giga and Tera-Scale data sets.

Index Terms— Mathematical morphology, Connected filters, Component trees, Image representation, Parallel computing

1. INTRODUCTION

Component Trees [1] are hierarchical structures that encode the nested relations of the *connected components* at all threshold sets of an image. They are well suited to perform attribute filtering operations, which are filters discarding the connected components that do not satisfy a given criterion [2]. Tree techniques were successfully applied for analyzing biomedical [3], remote-sensing [4, 5], and astronomical images [6].

Recently, these tools have been improved to process Giga and Tera-Scale data sets using shared-memory [7, 8] and distributed-memory techniques [9, 10], or a combination of both [11, 12]. These approaches typically build upon a divide-and-conquer approach where the data set is split into several tiles. Component trees are concurrently built for each tile and latter *merged* together to correct the nested relations of the connected components in a parallel manner. Additionally, [7, 10, 12] also presented algorithms to perform attribute filtering or multi-scale analyses in parallel.

The methods mentioned above require choosing the attribute function a priori, such that the components' attributes are computed during the construction of the hierarchical tree representation. This means that using a different attribute function requires to re-build the underlying tree structure. MTDEMO [3] allows sequential computation of the components' attributes on an already existing tree, yet, no algorithm exists to perform the attribute computation in a parallel manner. Such improvement would be valuable for large data sets where re-building the tree is computationally expensive. In this paper, we present a novel approach that enables parallel attribute computation using the Distributed Component Forest (DCF) framework implemented in DISCCOFAN [12].

This paper is organized as follows: Section 2 introduces component trees and attribute filters. Section 3 details the implementation of the parallel attribute computation. Section 4 discusses the performance of this novel approach and Section 5 summarizes our main results.

2. ATTRIBUTE FILTERS AND COMPONENT TREES

Attribute filters are a type of connected filters that remove connected components whose attributes are lower than a given criterion λ. A simple attribute function is the area A (number of pixels) of the connected components [13]. Connected components at a given threshold level are removed if they have $A < \lambda$. Diverse attribute functions exist depending on the applications considered. State-of-the-art component tree methods ([7, 12]) include the *moment invariants* of the components which is useful to extract structures based on their shape properties [3]. In pixel-based processing applications, the filtered image is obtained by applying the attribute filter at all threshold sets and stacking the resulting slices.

However, the use of attribute filters is much more efficient when applied to component tree representations. Component trees are powerful region-based image representations to apply connected filtering or multi-scale analyses. Tree structures encode the relationship between the connected components of an image or volume for all the threshold sets in the data. In a tree structure, connected components are represented by *nodes* which characteristics can be reached through a single element, called the *levelroot* of the node. Constructing the nested relations of the connected components corre-

sponds to building the parent-child relations of all the nodes of the trees by connecting their levelroots. Two typical types of Component Trees are the *Max-Tree* and *Min-Tree*, where leaves represent the bright and dark connected components features, respectively [1].

Performing attribute filtering on the tree representation of a given data set can be done by accessing the levelroots of the different nodes in the tree and removing the ones that do not meet the given criterion. In typical tree construction algorithms, the attribute function and criterion must be chosen beforehand so that the attributes are computed while building the tree [12]. This is limiting for applications that require testing several attribute functions on vast data sets because the tree has to be rebuilt every time. However, the computational load can be lowered by computing the components' attributes in a parallel manner on existing trees. In the next section, we detail how we implemented this approach within the recent DCF method DISCCOFAN [12].

Algorithm 1 Attribute flooding of the local component trees

```
1: procedure TREE_ATTR_FLOODING(ComponentTree tree, long[] ranks, AttrFunction f_attribute)
2:   INIT_ATTR_ARRAY(tree.attr, f_attribute)
3:   visited ← ALLOC_EMPTY_ARRAY(size_tree)
4:   for i ∈ [size_tree − 1, 0] do
5:     node ← ranks[i]
6:     visited[node] ← True
7:     parent ← tree.par[node]
8:     MERGE_ATTR(tree.attr[node], tree.attr[parent])
9:     if tree.gval[node] == tree.gval[parent] and
       visited[parent] then
10:      parent ← tree.par[parent]
11:      MERGE_ATTR(tree.attr[node], tree.attr[parent])
12:    end if
13:  end for
14: end procedure
```

3. DISTRIBUTED COMPONENT FOREST

Component Trees are powerful approaches for image filtering or segmentation but the size of component trees is typically 20 to 50 times the size of the image itself, which becomes prohibitive for dealing with vast data sets. To overcome this limitation, Kazemier et al. [9] introduced the Distributed Component Forest, later extended by Gazagnes & Wilkinson [10, 12]. The image is first divided into tiles assigned to different MPI nodes such that local component trees (with attributes) are built concurrently for all tiles. Because the individual trees only encode the connected components within their tile, the local component trees are corrected for features spanning over the whole image using *Boundary Tree* structures. The latter consists of the subset of nodes in the local tree that characterizes the connected components spanning over the borders of each tile. Boundary Trees are merged and combined two-by-two to correct the parent-child relations and attributes of the spanning components in the whole image. Once all the trees have been merged, the updated information is used to correct the local component trees of each tile. Then, post-processing concurrently the individual local trees (the DCF) yields the same result as processing the entire component tree of the image. Recently, we implemented DISCCOFAN, a DCF-based method, and highlighted its promising results to process vast 2D and 3D, low to high dynamic range, data sets [12]. The next section shows how we extended this framework to perform the parallel attribute flooding.

3.1. Parallel Attribute Flooding

Using several attribute functions successively, in an organized or interactive manner, requires splitting the computation of the DCF into two steps: (1) constructing the correct nested relations of the connected components within the DCF and (2) computing and propagating the components' attributes depending on the attribute function chosen. The first step requires little modification of the original DCF implementation, hence, for space considerations, we only describe the changes related to the second step.

In this new approach, the attributes are propagated on an existing DCF. To do so, we first derive the components' attributes in the local trees within each MPI node. This is done in a concurrent manner using the procedure from Algorithm 1. We initialize the attributes of all the nodes in the tree using a generic function INIT_ATTR_ARRAY that adapts to the choice of the attribute function $f\_attribute$. The computation of the components' attributes (stored in the array $tree.attr$) starts from the leaves of the tree and progresses to the bottom until we reach the root. The array *ranks* holds the mapping of the pixel intensities ($tree.gval$) into an array of prefixes, sorted in decreasing order. Because path compression is guaranteed in the first DCF phase, the attribute of each node that is not a levelroot is directly merged to the levelroot at the same level (found with the array $tree.par$). For the levelroot nodes, their attribute is merged with the levelroot at the lower level. We use an array *visited* to ensure that attributes from the same node are not propagated several times.

Once the attributes have been propagated in the local component trees, they must be corrected in a parallel manner such that filtering these trees yield the same result as filtering the tree of the whole image. We adapted the Boundary tree structures mentioned in Section 3. This procedure, detailed in Algorithm 2, is adapted from Algorithm 3 found in [12]. Since the parent-child relations have already been computed in a previous step, Algorithm 2 is much simpler and enables efficient attribute propagation within the Boundary trees. Hence, for two nodes x and y, belonging to two boundary trees a and b, to be merged, we simply propagate the attribute of x to y and assign y as the new parent of x. Hence, x is no longer

Algorithm 2 Merging two branches with the nodes x and y, belonging to two boundary trees a and b

procedure MERGE_BRANCHES(BoundaryNode x, BoundaryNode y)
  $x \leftarrow$ B_LEVELROOT(x)
  $y \leftarrow$ B_LEVELROOT(y)
  while x is not $\perp$ and $x \neq y$ **do**
    $z \leftarrow$ B_PARENT(x)
    MERGE_ATTR(B_ATTR(x), B_ATTR(y))
    B_PARENT(x) $\leftarrow y$
    $x \leftarrow z$
    $y \leftarrow$ B_PARENT(y)
  end while
end procedure

a levelroot of the merged $a - b$ structure its attribute is only propagated once.

Once all boundary trees have been merged, the components' attributes in the boundary trees are used to update the attributes in the local component trees of each tile. Hence, the DCF can be concurrently processed to perform attribute filtering or multi-scale analysis similarly as in the original DISCCOFAN implementation. We refer the reader to [12] for more details on the DCF post-processing functions.

4. RESULTS

We assess in this section the performance of the approach described in Section 3. All the tests are run on the Peregrine cluster of the Center for Information Technology of the University of Groningen. We use the 7 high memory nodes (1024 or 2048 GB memory) with each 48 Intel Xeon E7 4860v2 2.6 GHz cores used as individual MPI processes during our experiments. The libraries used in all tests are gcc 8.2.0, HDF5 1.10.5, and OpenMPI 3.1.3. The code was compiled with an optimization level O3. We used a large 2D data set of 81503 by 108199 pixels ($\approx$ 9 Gpx), which an ESO astronomical image of the Milky Way taken at the Paranal Observatory in Chile [14]. The image is an RGB data set that we converted into a floating-point data set by creating a luminance image where the intensity of each pixel is given by $L = 0.2126R + 0.7152G + 0.0722B$. Additionally, we derived a lower quantization level image from this data set, by scaling down the intensities between 0 and 255.

Our experiment is defined as follow: for each of the two data sets, we first construct the nested relations of the components of the individual trees of the DCF and then compute the components' attributes for three different characteristics: their area; the area of the minimum enclosing rectangle; and the non-compactness. The first two are a measure of the size of the connected components, and the last one is a scale-invariant measure of the components' deviation in shape from a solid Euclidean disc.

Figure 1 shows the computational time of successively building and correcting the parent-child relations of the local component trees, and flooding and filtering the DCF using the three different attribute functions mentioned previously (referred to as Area, Rect, and Ncomp in the figures), for the two quantized versions of the ESO data set. The right panel shows the corresponding speed-up for both cases, computed as $t(1)/t(N_p)$ where $t(1)$ is the wall-clock time obtained when using 1 process, and $t(N_p)$ the processing time when using N_p processes. We use up to 64 MPI processes in total, and the number of tiles used in each case is equal to the number of processes (*e.g.* for two processes, the original image is divided into two tiles). We do not detail the memory usage of this approach, as no change is expected from the original implementation [12] when both the parents and attributes are computed simultaneously.

The computational time needed to compute the attributes and filter the image varies depending on the attribute function used. This is expected given the different degrees of complexity of these functions. The Area attribute is less computationally expensive given that it only sums the number of pixels belonging to a given connected component. The minimum area of enclosing rectangle, or the non-compactness are more expensive since they include additional operations such as multiplications, MIN or MAX functions. The right panel of Figure 1 shows that this alternative approach has a linear speed-up for both low and moderate/high dynamic range data sets, highlighting that the attribute computation can be done efficiently in a parallel manner.

We also investigate the time gains obtained by enabling only the attributes to being re-computed in parallel rather than the whole DCF. W run the original implementation of DISCCOFAN, which computes both the parent-child relations and the attributes simultaneously, using the three same attribute functions (Area, Rect, and Ncomp). Figure 2 left and middle panels compare the computational time of computing both the tree and attributes (dotted lines) and computing only attributes on already existing trees (solid lines). The right panel shows the corresponding speed-up gains, computed as $t(\text{Attributes} + \text{trees})/t(\text{Attributes})$, for different number of processes used. As expected, computing the attributes on the already built DCF is faster than constructing both simultaneously. We note that for the 8-bit image, the computational gain is a factor of 2 approximately, suggesting that not having to reconstruct the whole tree would divide the timings by 2. For the floating-point case, the performance is slightly lower with a speed-up factor of ~ 1.5. Typically, in the latter case, the local components trees have a larger number of nodes than in the 8-bit case, which impacts the overall size of the boundary tree structures that are merged in parallel. Hence, propagating the attributes takes longer in the latter case.

Figure 2 highlights that these gains are roughly constant as we use more processes, showing that this alternative strategy has a similar scaling relation as the original approach.

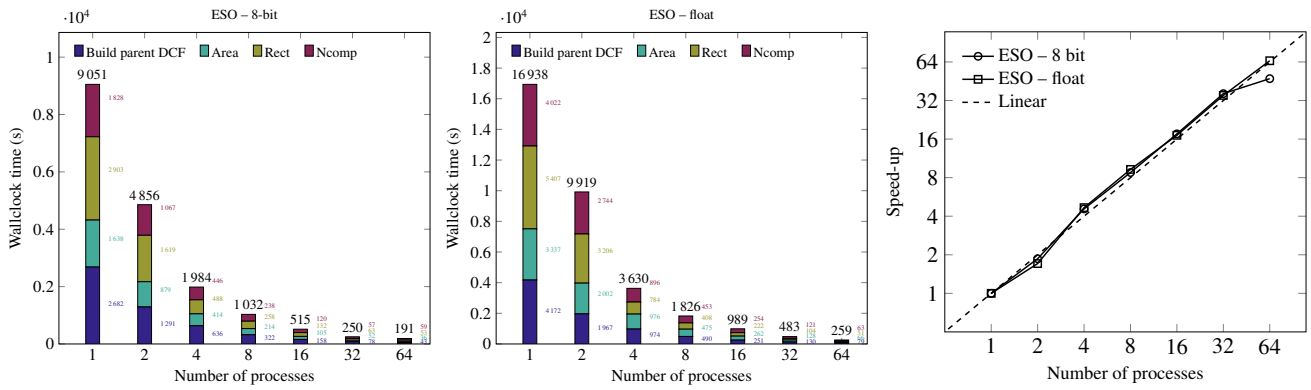

Fig. 1. Computational time of our modified approach on the two quantized version of the ESO image (left and middle panel), and the corresponding speed-up compared to the linear expectation (right panel).

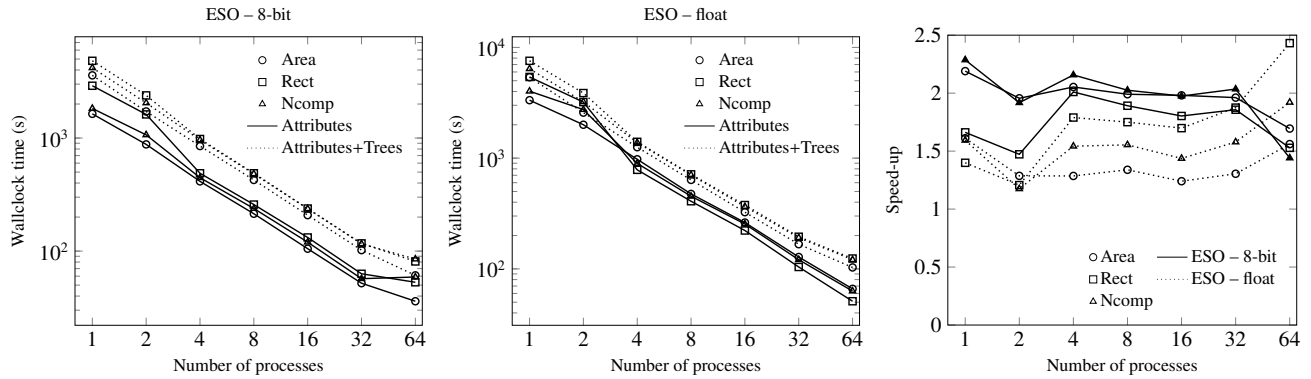

Fig. 2. Left and middle panels: time comparison of computing both the trees and attributes (dotted lines) and computing only attributes on already existing trees (solid lines). Right panel: corresponding speed-up ($t(\text{Attributes} + \text{Trees})/t(\text{Attributes})$) for the two data sets.

The slight decrease in speed-up when using 64 processes in the 8-bit case suggests that we might reach a plateau as using more MPI nodes does not significantly decrease the computation time. For the floating-point case, having smaller chunks further might still improve the performance because the length of each tree decreases. We note that, while the speed-ups reported in Figure 2 might look moderate, the overall speed-up of e.g. performing several attribute filters on the same data set will be significant when one uses a large number of attributes functions.

5. CONCLUSION

In this work, we presented an extension of a recent DCF approach, DISCCOFAN that enables parallel attribute computation once the DCF representation has been built. This novelty allows to choose and switch between several attribute functions without having to recompute the whole DCF structure. This is particularly useful for Giga and Tera-scale applications, where building the individual local component trees can be computationally expensive.

Our results highlight that the scaling performance of the parallel attribute computation is linear. Additionally, computing only the components' attributes in parallel divides the computational time by a factor of 2 and 1.5 for 8-bit and single-precision floating-point data sets, compared to the case where the underlying tree structure is rebuilt. This method could be trivially extended to more complex processing approaches, for example by storing the DCF into a file, which could be read and re-processed with different attribute functions depending on a given application requirement. This would further reduce the cost of building the parent-child nested relations. These techniques could also be used to further develop the MTDEMO software [3] which was suited for interactive visualization of 2D or 3D data sets using a sequential post-attribute computation. Overall, this paper paves the way for complex region-based filtering, visualization applications, or vector-attribute processing techniques, which require combining several attribute functions on large data sets. The code is publicly accessible: `https://github.com/sgazagnes/disccofan`.

6. REFERENCES

[1] P. Salembier, A. Oliveras, and L. Garrido, "Anti-extensive connected operators for image and sequence processing," *IEEE Trans. Image Proc.*, vol. 7, pp. 555–570, 1998.

[2] E. J. Breen and R. Jones, "Attribute openings, thinnings and granulometries," *Comp. Vis. Image Understand.*, vol. 64, no. 3, pp. 377–389, 1996.

[3] M. A. Westenberg, J. B. T. M. Roerdink, and M. H. F. Wilkinson, "Volumetric attribute filtering and interactive visualization using the max-tree representation," *IEEE Trans. Image Proc.*, vol. 16, pp. 2943–2952, 2007.

[4] M. H. F. Wilkinson, P. Soille, M. Pesaresi, and G. K. Ouzounis, "Concurrent computation of differential morphological profiles on giga-pixel images," in *Proc. Int. Symp. Math. Morphology (ISMM) 2011*, 2011, vol. 6671 of *Lecture Notes in Computer Science*, pp. 331–342.

[5] Martino Pesaresi, Guo Huadong, Xavier Blaes, Daniele Ehrlich, Stefano Ferri, Lionel Gueguen, Matina Halkia, Mayeul Kauffmann, Thomas Kemper, Linlin Lu, et al., "A global human settlement layer from optical hr/vhr rs data: concept and first results," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 5, pp. 2102–2131, 2013.

[6] P. Teeninga, U. Moschini, S. C. Trager, and M. H. F. Wilkinson, "Statistical attribute filtering to detect faint extended astronomical sources," *Mathematical Morphology – Theory and Applications*, vol. 1, no. 1, pp. 100—-115, 1 2016.

[7] M. H. F. Wilkinson, H. Gao, W. H. Hesselink, J. E. Jonker, and A. Meijster, "Concurrent computation of attribute filters using shared memory parallel machines," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 10, pp. 1800–1813, 2008.

[8] Ugo Moschini, Arnold Meijster, and Michael H. F. Wilkinson, "A hybrid shared-memory parallel max-tree algorithm for extreme dynamic-range images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, pp. 513–526, 2018.

[9] Jan J Kazemier, Georgios K Ouzounis, and Michael H F Wilkinson, "Connected morphological attribute filters on distributed memory parallel machines," in *Proc. Int. Symp. Math. Morphology (ISMM) 2017*. Springer, 2017, pp. 357–368.

[10] Simon Gazagnes and Michael H. F. Wilkinson, "Distributed component forests in 2-d: Hierarchical image representations suitable for tera-scale images," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 33, no. 11, pp. 1940012, 2019.

[11] Markus Götz, Gabriele Cavallaro, Thierry Géraud, Matthias Book, and Morris Riedel, "Parallel computation of component trees on distributed memory machines," *IEEE Transactions on Parallel and Distributed Systems*, 2018.

[12] Simon Gazagnes and Michael H. F Wilkinson, "Distributed connected component filtering and analysis in 2d and 3d tera-scale data sets," *IEEE Transactions on Image Processing*, vol. 30, pp. 3664–3675, 2021.

[13] L. Vincent, "Grayscale area openings and closings, their efficient implementation and applications," in *Proc. EURASIP Workshop on Mathematical Morphology and its Application to Signal Processing*, Barcelona, Spain, 1993, pp. 22–27.

[14] R. K. Saito, D. Minniti, B. Dias, M. Hempel, M. Rejkuba, J. Alonso-García, B. Barbuy, M. Catelan, J. P. Emerson, O. A. Gonzalez, P. W. Lucas, and M. Zoccali, "Milky Way demographics with the VVV survey. I. The 84-million star colour-magnitude diagram of the Galactic bulge," *"Astronomy and Astrophysics"*, vol. 544, pp. A147, Aug 2012.