Generalized Connected Morphological Operators for Robust Shape Extraction

Ouzounis, Georgios Konstantinou

Published as: G. K. Ouzounis and M. H. F. Wilkinson – *"A parallel implementation of the dual-input Max-Tree algorithm for attribute filtering,"* IEEE Transactions on Image Processing, vol. 12, no. 7, pp. 729–739, July 2003.

Chapter 4

A Parallel Implementation of the Dual-Input Max-Tree Algorithm for Attribute Filtering

Let thy speech be short, comprehending much in a few words.

Aprocrypha

Abstract

This paper presents a concurrent implementation of a previously developed Dual-Input Max-Tree algorithm that implements anti-extensive attribute filters based on second-generation connectivity. The paralellization strategy has been recently introduced for ordinary Max-Trees and involves the concurrent generation and filtering of several Max-Trees, one for each thread, that correspond to different segments of the input image. The algorithm uses a Union-Find type of labelling which allows for efficient merging of the trees. Tests on several 3-D datasets using multi-core computers showed a speed-up of 4.14 to 4.21 on 4 threads running on the same number of cores. Maximum performance of 5.12 to 5.99 was achieved between 32 and 64 threads on 4 cores.

4.1 Introduction

ATTRIBUTE filters [11, 65] are a class of shape preserving operators. Their key property is that they operate on image regions rather than individual pixels. This allows image operations without distorting objects, i.e., they either remove or preserve objects intact, based on some pre-specified property. Attribute filters can be efficiently implemented using the Max-Tree algorithm [65], or similar tree structures [35, 89]

Image regions in mathematical morphology are characterized by some notion of connectivity, most commonly 4- and 8-connectivity. This yields an association between connectivity and connected operators which is extensively discussed in [9, 62, 69]. These papers also provide extensions to these basic connectivities known as *second-generation connectivity*. A general framework and algorithm is presented in [57]. The algorithm referred to as the *Dual-Input Max-Tree* supports the mask-based connectivity scheme, for which we give a concurrent implementation in this paper. It is based on the parallel Max-Tree algorithm in [93], which builds individual Max-Trees for image regions concurrently, and merges these trees efficiently.

4.2 Attribute filters

Attribute filters are based on *connectivity openings*. In essence, a connectivity opening $\Gamma_x(X)$ yields the connected component containing the point $x \in X$ and $\emptyset$ otherwise. A connectivity opening is characterized by the following properties; for any two sets $X,\ Y$ it is *anti-extensive* i.e. $\Gamma_x(X) \subseteq X$, *increasing* i.e. if $X \subseteq Y \ \Rightarrow\ \Gamma_x(X) \subseteq \Gamma_x(Y)$, and *idempotent* i.e. $\Gamma_x(\Gamma_x(X)) = \Gamma_x(X)$. Furthermore, for all $X \subseteq E$, $x, y \in E, \Gamma_x(X)$ and $\Gamma_y(X)$ are equal or disjoint.

A general approach in deriving second-generation connectivity openings using arbitrary image operators is given in [57]. A mask-based connectivity opening is defined as:

$$\Gamma_x^M(X) = \begin{cases} \Gamma_x(M) \cap X & \text{if } x \in X \cap M, \quad (4.1a) \\ \{x\} & \text{if } x \in X \setminus M, \quad (4.1b) \\ \emptyset & \text{otherwise.} \quad (4.1c) \end{cases}$$

where M is an arbitrary, binary mask image.

We can define a number of other connected filters based on a connectivity opening that work by imposing constraints on the connected components it returns. In the case of attribute openings such constraints are commonly expressed in the form of binary criteria which decide to accept or to reject components based on some attribute measure.

Attribute criteria Λ are put in place by means of a trivial opening Γ_Λ. The latter yields C if $\Lambda(C)$ is true, and $\emptyset$ otherwise. Furthermore, $\Gamma_\Lambda(\emptyset) = \emptyset$. Attribute criteria are typically expressed as:

$$\Lambda(C) = Attr(C) \geq \lambda \quad (4.2)$$

with $Attr(C)$ some real-value attribute of C, and λ an attribute threshold.

Definition 12. *The binary attribute opening Γ^Λ of a set X with an increasing criterion Λ is given by:*

$$\Gamma^\Lambda(X) = \bigcup_{x \in X} \Gamma_\Lambda(\Gamma_x(X)) \quad (4.3)$$

Many examples are given in [11,65]. Note that if Λ is non-increasing we have an *attribute thinning* Φ^Λ [11] instead. An example is the scale-invariant non-compactness criterion of the form of (4.2), in which

$$Attr(C) = I(C)/V^{5/3}(C), \text{ where } I(C) = \frac{V(C)}{4} + \sum_{\mathbf{x} \in C} (\mathbf{x} - \overline{\mathbf{x}})^2 \quad (4.4)$$

with I the trace of the moment of inertia tensor in 3-D and $V(C)$ the volume of a component C [94]. Attribute filters can be operated on sets characterized by second-generation connectivity by replacing Γ_x with Γ_x^M instead. The proof of this and a more detailed analysis

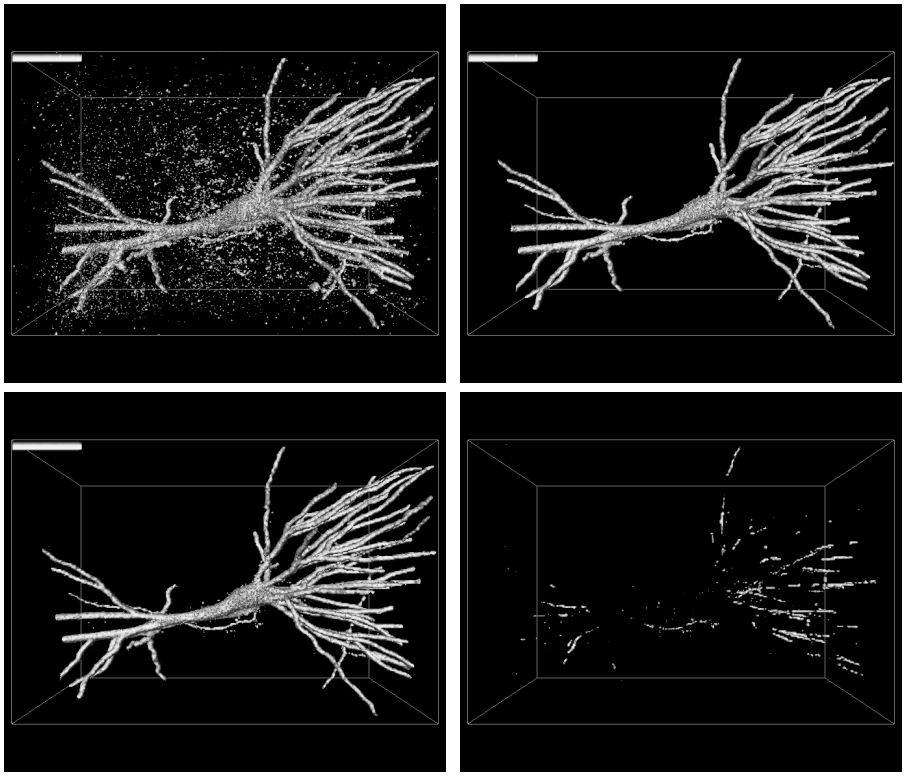

Figure 4.1: *Isosurface projections of a confocal laser scanning micrograph of a pyramidal neuron and the output of the non-compactness filter* (4.4) *based on the 26-connectivity, both at isolevel 1. The first image in the bottom row illustrates the filter's performance using closing-based connectivity and the second shows the difference volumes between two attribute filter results. Various details within the neuron are lost using the 26-connectivity which are preserved by using a second-generation connectivity instead. See [57] for details.*

can be found in [57]. Furthermore, an investigation in optimizing the parameters affecting the performance of these filters is discussed in [56] An example of attribute thinnings using closing-based second-generation connectivity is shown in Figure 4.1.

4.3 The Max-Tree algorithm

The Max-Tree was introduced by Salembier [65] as a versatile structure for computing anti-extensive attribute filters on images and video sequences. It is a rooted, unidirected tree

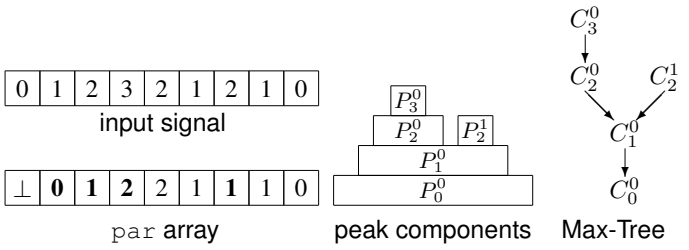

Figure 4.2: *Example of input signal, peak components, Max-Tree and its encoding in a* `par` *array, in which* $\perp$ *denotes the overall root node, and boldface numbers denote the level roots, i.e., they point to positions in the input with grey level other than their own.*

in which the node hierarchy corresponds to the nesting of peak components given a gray-scale image. A peak component P_h at level h is a connected component of the thresholded image $T_h(f)$. Each tree node C_h^k (k is the node index) contains only those pixels of a given peak component which have gray-level h. In addition each node except for the root, points towards its parent $C_{h'}^{k'}$ with $h' < h$. The root node is defined at the minimum level h_{min} and contains the set of pixels belonging to the background.

The algorithm is a three-stage process in which the construction of the tree and the computation of node attributes is independent of filtering and image restitution. During the construction stage every pixel visited contributes to the auxiliary data buffer associated to the node it belongs to. Once a node is finalized, its parent inherits these data and re-computes its attribute. Inheritance in the case of increasing attributes such as area/volume is a simple addition while for non-increasing attributes such as the non-compactness measure of (4.4) the accumulation relies on more delicate attribute handling functions described in [57].

4.4 Including union-find in the Max-Tree

The hierarchical queue-based algorithm given by Salembier [65] cannot be trivially parallellized. In our approach we choose to partition the image into N_p connected disjoint regions the union of which is the entire image domain. Each region is assigned to one of the N_p processors for which a separate tree is constructed. The non-trivial part of this approach is the merging of the resulting trees. It is a process that requires (i) the merging of the peak components P_h^i, (ii) the updating of the parent relationships, and (iii) the merging of the attributes of the peak components. Parallellizing the filtering stage is trivial.

Previously, Najman et al. provided an algorithm to compute the Max-Tree using union-find [52]. Wilkinson et al [93] use a different approach, using Salembier et al's original algorithm [65] and changing the way the labels indicating node-membership of each pixel were chosen. Instead of using arbitrary numbers, Wilkinson et al use the index of the first

pixel of a node as the label. This means that each pixel of a node points to this "canonical element", which is referred to as a *level root*. The level root of a node itself is given the level root of its parent node as its index. These labels (or actually parent pointers in union-find terms) are stored in an array denoted `par`. Thus, if $f(\texttt{par}[x]) \neq f(x)$, x is a level root. In the algorithm in [93], after building a tree using a single thread, each $\texttt{par}[x]$ points directly to a level root: its own if x is not a level root, or to the level root of the parent node. An example is shown in Figure 4.2. Once the results of multiple threads are merged, this is no longer true. Therefore, we implement a function `levroot` to find the level root of any pixel. If $\texttt{levroot}(x) = \texttt{levroot}(y)$ x and y belong to the same node. The implementation of `levroot` also includes path compression as in [79].

4.5 The dual-input mode

As in the sequential case, the structure of the Max-Tree is dictated by the peak components of the mask volume m rather than the original volume f. An example is given in Figure 4.3. The dual-input version of the algorithm in [93] requires a number dummy nodes which assist in the merging of the different trees once all the threads return. To do this we double the size of the `par` array, and place the volumes f and m side by side in a single block of memory. In this way $f(p + volsize) = m(p)$ for all voxels p in the volume domain. For all p for which $f(p) \neq m(p)$ $\texttt{par}(p + volsize)$ will contain a valid reference to a level root.

The flooding function proceeds as described in [93] only we modify the way auxiliary data are handled and add a number of intensity mismatch checks to conform with the dual-input algorithm. After reaching a given level lev(=current level in mask m) and before retrieving any of the pixels available in the queue for that level, we first initialize the auxiliary data variable $attr$. It is set to the attribute count of the node corresponding to the $lero[lev]$. If an attribute count from a node at higher level is inherited through parameter $thisattr$, we update $attr$. A while loop then retrieves sequentially the members of the queue and for each one performs the mismatch check. If $f(p) \neq m(p)$ for a pixel p this signals the case in which p belongs to the current active node at $f(p)$ through the connected component at level $m(p)$, i.e. it defines a peak component at level $f(p)$ to which p in the mask volume is connected. In terms of our parallelizing strategy this means that it already defines a dummy node at $m(p)$ offset by $volsize$. We must then set $\texttt{par}(p + volsize)$ to $lero[lev]$. We must also create a new node at level $f(p)$ if none exists, and add p to the node at level $f(p)$. If $f(p) > m(p)$ p is a singleton (according to (4.1)). This requires finalizing the node which is done by setting its parent to $lero[lev]$, setting its auxiliary data to the unit measure and clearing $lero[f(p)]$. Details are given in Algorithm 1.

Otherwise, if $f(p) = m(p)$, it is necessary to check if the $lero[lev] \geq volsize$, i.e. if it is a dummy node. If this is the case, we update $\texttt{par}[lero[lev]]$ to p, and then set $lero[lev]$ to p, effectively setting the level root to a non-dummy node. The auxiliary data stored in *attr*

Algorithm 1 *The flooding function of the concurrent Dual-Input Max-Tree algorithm.*

```
procedure LocalTreeFlood(threadno, lero, lev, thisattr) =
    Initialize auxilliary attribute data attr and merge with thisattr
    while (QueueNotEmpty(set, lev)) do
        retrieve p from queue at level lev
        if f(p) ≠ lev then
            par[p + volsize] := lero[lev];
            if node at level f(p) exists then
                add p to it; par[p] := lero[f(p)];
            else
                create node at level f(p); lero[f(p)] := p;
            end;
            if f(p) > lev then    (* singleton with parent at lev *)
                finalize node; add p to attr; par[p] := lero[lev];
            end;
        else    (* No mismatch *)
            if lero[lev] ≥ volsize then    (* First pixel at level lev *)
                par[lero[lev]] := p; lero[lev] := p;
            end;
            add p to attr;
        end;    (* No mismatch *)
    end;    (* while *)
    for all neighbours q of p do
        if not processed[q] then
            processed[q] := true; mq := m(q);
            initialize childattr to empty;
            if m(q) ≠ f(q) then newnode := q + volsize;
            else newnode := q; end;
            if lero[m(q)] does not exist then lero[m(q)] := newnode;
            else par[newnode] := lero[m(q)]; end;
            while mq > lev do
                mq := LocalTreeFlood(threadno, lero, mq, childattr);
            end;
            add any data in childattr to attr;
        end;
    end;    (* for *)
    detect parent of lero[lev]
    add auxilliary data in attr to auxilliary data of lero[lev]
    set thisattr to attribute data of lero[lev]
    return level of parent of lero[lev]
end LocalTreeFlood.
```

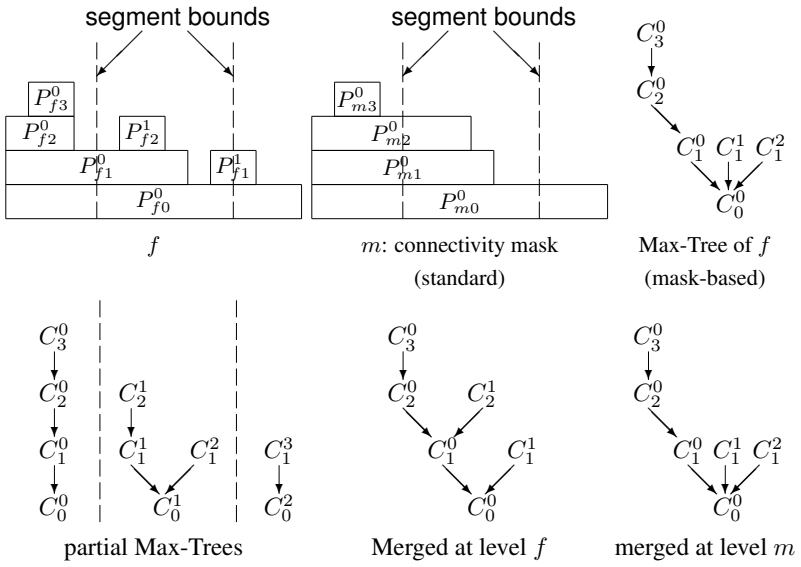

Figure 4.3: *Dual Input Max-Tree of 1-D signal f using mask m: The attributes of C^0_2 and C^1_2 are merged to C^0_2 since all pixels at level $h = 2$ are clustered to a single peak component. Furthermore C^1_1 breaks up to a number of singleton nodes equal to the number of pixels in P^1_{f1}. Bottom row: partial Max-Trees of segments of signal indicated by the dashed lines; merger of partial Max-Trees at level of f at the boundaries yields standard Max-Tree in this case; merging at level of m at the boundary yields correct result.*

are then updated.

For every unprocessed neighbour q of p we determine where to create a new node. If $f(q) = m(q)$ the new node is q, otherwise $q + volsize$. If $lero[m(q)]$ exists, we set $\texttt{par}[newnode]$ to $lero[m(q)]$, otherwise $lero[m(q)]$ is set to $\texttt{par}[newnode]$. If $m(q) \geq lev$ we then enter into the recursion as in [65, 93].

4.6 Concurrent merging of Max-Trees

As in regular connectivities, we must now connect the N_p Max-Trees. In [93], this is done by inspecting the pixels along the boundary between the parts, and performing the `connect` function on adjacent pixels on either side of the boundary. This function is shown in Algorithm 3. A proof of the correctness and a detailed discussion are given in [93]. The key reason why this works efficiently, is that merging two nodes containing x and y, with $f(x) = f(y)$

Algorithm 2 *Concurrent construction and filtering of the Max-Trees, thread p.*

process `ccaf`(p)
  build dual input Max-Tree *Tree*(p) for segment belonging to p
  var $i := 1$, $q := p$;
  while $p + i < K \ \wedge \ q$ **mod** $2 = 0$ **do**
    wait to glue with right-hand neighbor ;
    for all edges (x, y) between *Tree*(p) and *Tree*$(p + i)$ **do**
      if $f(x) \neq m(x)$ **then** $x := x + volsize$;
      if $f(y) \neq m(y)$ **then** $y := y + volsize$;
      `connect`(x, y) ;
    end ;
    $i := 2 * i$; $q := q/2$;
  end ;
  if $p = 0$ **then**
    release the waiting threads
  else
    signal left-hand neighbor ;
    wait for thread 0
  end ;
  $filter(p, lambda)$;
end `ccaf`.

reduces to the assignment

$$\texttt{par}[\texttt{levroot}(y)] := \texttt{levroot}(x). \tag{4.5}$$

This is easily verified as follows: `par`$[$`levroot`$(y)]$ now points to a pixel with the same grey level because $f(x) = f(y)$, and `levroot`$(x) =$ `levroot`(y) after assignment (4.5), so that x and y belong to the same node.

Function `connect` is called by the process *concurrent construction and filter* or `ccaf`(see Algorithm 2), which corresponds to one of the threads of the concurrent merging algorithm. Each thread p first builds a Max-Tree for its own sub-domain V_p.

Process `ccaf` is called after initializing `par`, the auxiliary data functions and preparing the thread data. It starts off by first initializing the level root array $lero$ and hierarchical queue for all gray-levels and finding the minimum voxel values in f and m. Having got the starting voxel of minimum grey value in m it calls `LocalTreeFlood`. If the minimum values in f and m differ, some post-processing as explained in [57] is required.

Algorithm 3 *Merging two Max-Trees*

procedure `connect`$(x, y) =$
  Initialize auxilliary attribute data `temp1` to empty
  $x :=$ `levroot`(x) ; $y :=$ `levroot`(y) ;
  if $f(y) > f(x)$ **then** *swap*(x, y) **end**
  while $x \neq y \ \wedge \ y \neq \perp$ **do**
    $z :=$ `levroot`$($`par`$[x])$
    if $z \neq \perp \ \wedge \ f(z) \geq f(y)$ **then**
      Add data in `temp1` to attribute data of x ;
      $x := z$;
    else
      `temp2` := sum of attribute data of x and `temp1` ;
      `temp1` := attribute data of x ;
      attribute data of x := `temp2` ;
      `par`$[x] := y$; $x := y$; $y := z$;
    end
  end
  if $y = \perp$ **then** (* root of one tree reached *)
    while $x \neq \perp$ **do** (* process remaining ancestors of x *)
      Add data in `temp1` to attribute data of x ;
      $x :=$ `levroot`$($`par`$[x])$;
    end
  end
end `connect`.

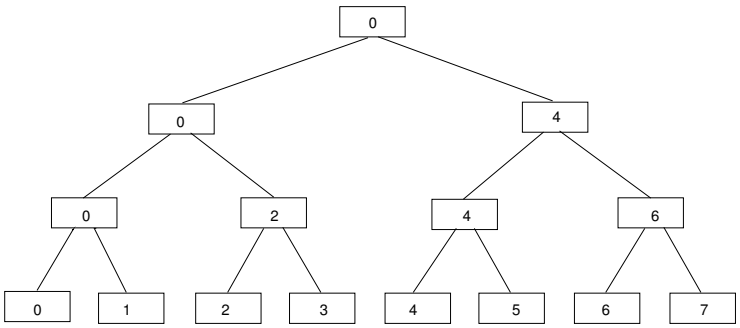

Figure 4.4: *Binary tree used for merging domains.*

After this, the sub-domains are merged by means of a binary tree in which thread p accepts all sub-domains V_{p+i} with $p+i < N_p$ and $0 \leq i < 2^a$, where 2^a is the largest power of 2 that divides p. An example of a binary tree for $N_p = 8$ is shown in Figure 4.4. Note that odd-numbered threads accept no sub-domains. A thread that needs to accept the domain of its right-hand neighbor, has to wait until the neighbor has completed its Max-Tree computation. Because the final combination is computed by thread 0, all other threads must wait for thread 0 before they can resume their computation for the filtering phase. This synchronization is realized by means of two arrays of $N_p - 1$ binary semaphores. The filtering phase is also fully concurrent, and is identical to that described in [93].

For second-generation connectivity, the difference lies not in the implementation of `connect`, but in *which* pixels need to be merged. Suppose x and y are adjacent voxels which lie on different sides of the boundary inspected by `ccaf`. If $f(x) = m(x)$ the node in the Max-Tree at level $f(x)$ is the correct one, as before, otherwise we should start merging at level $m(x)$, as shown in Figure 4.3. At the left-hand segment boundary in this figure, merging at level $f(x)$ ignores the fact that P^0_{f2} and P^1_{f2} are clustered together in node C^0_2 using connectivity based on mask m. By contrast, at the right-hand segment boundary, merging from level $f(x)$ would merge nodes C^2_1 and C^3_1, which are considered singletons in the mask-based connectivity. In the scheme outlined above, this means that we start the merger from x if $f(x) = m(x)$, and from $x + volsize$, otherwise. The same holds for y. Thus the only changes to the `ccaf` function when compared to [93] lies in the statements immediately preceding the call to `connect`.

4.7 Performance testing and complexity

The above algorithm was implemented in C for the general class of anti-extensive attribute filters. Wall-clock run times for numbers of threads equal to 1, 2, 4, 8, 16, 32, and 64 for for two different attributes were determined. The attributes chosen were volume (yielding an attribute opening) and the non-compactness measure (4.4) [94] yielding an attribute thinning.

Timings were performed on an AMD dual-core, Opteron-based machine. This machine has two dual-core Opteron 280 processors at 2.4 GHz, giving a total of 4 processor cores, and 8 GB of memory (4 GB per processor socket). Each timing was repeated 10 times, and the minimum was used as the most representative of the algorithm's performance. Five volume data sets publicly available from `http://www.volvis.org` were used. All volumes were 8 bit/voxel sets, comprising 4 CT-scans and 1 MRI scan. Test were done using volume openings with $\lambda = 100$ and φ_1 with $\lambda = 2.0$ and the subtractive rule. The volume sizes ranged from 22.7 to 128 MB. The speed-up achieved is shown in Figure 4.5. As can be seen, the speed-up is slightly better than linear, as we move from 1 to 4 threads (4.21 $\pm$0.15 for volume openings and 4.14 $\pm$ 0.15 for non-compactness thinning at 4 threads). This may be due to the fact that more than 4 GB of memory is required when processing the larger

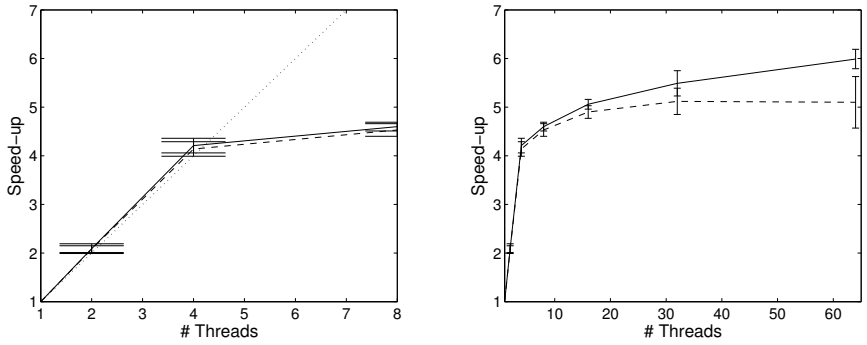

Figure 4.5: *Speed-up for volume openings (solid) and non-compactness thinnings (dashed) as a function of number of threads. The left graph shows the initial, slightly better than linear (dotted-line) speed-up as we move from 1 to 4 threads. The right-hand graph also shows the behavior up to 64 threads.*

volumes in the set, and therefore the processor doing the work requires access to the memory bank of the other socket, resulting in higher latency. As the number of threads exceeds the number of cores, we still obtain more speed-up, up to 5.99 ± 0.2 at 64 threads for volume openings, and 5.12 ± 0.27 at 32 threads for non-compactness thinning. In absolute terms, computing time went from between 20.8 and 128 s down to between 4.66 and 23.4 s.

The complexity of the algorithm is governed by two main parts: the building phase and the merging phase. Assuming a volume of $X \times Y \times Z = N$, in the building phase the time complexity is $O(GN/N_p)$, with G the number of grey levels, and N_p the number of processors. This complexity arises from the $O(GN)$ complexity of Salembier et al's Max-Tree algorithm [47]. If the number of grey levels is large, it may be better to replace this by Najman and Courpie's method [52]. The merging phase has complexity $O(GXY \log N \log N_p)$ if the volume is split up into slices orthogonal to the Z direction. The $\log N$ is due to the fact that we only use path compression, not union-by-rank. Memory requirements are $O(N+G)$.

4.8 Conclusions

The speed-up of the algorithm presented is similar to that of the parallel Max-Tree algorithm in [93]. However, it is about 50% slower in absolute terms. The speed-up if the number of threads exceeds the number of physical processors is due to reduced cache thrashing, as is confirmed by profiling. It also indicates that on machines with more processing cores, a (near) linear speed up beyond 4 CPUs is expected.

Apart from use in 3-D data, the algorithm could be of use in the efficient implementation

of attribute-space connected filters [92], in which the 2-D input image is embedded into a higher-dimensional attribute space, followed by application of a connected filter in that space.

Given the ready availability of multi-core processors, this algorithm is not restricted to supercomputers anymore, but will be of use to many, and in the near future most desktop machines.