Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

A partitioned algorithm for the image foresting transform

Felipe P. G. Bergo and Alexandre X. Falcão

Laboratório de Informática Visual (LIV), Instituto de Computação (IC), Universidade Estadual de Campinas (Unicamp), SP, Brazil
bergo@liv.ic.unicamp.br, afalcao@ic.unicamp.br

Abstract The Image Foresting Transform (IFT) is a powerful graph-based framework for the design and implementation of image processing operators. In this work we present the Partitioned IFT (PIFT), an algorithm that computes any IFT operator as a series of independent IFT-like computations. The PIFT makes parallelization of existing IFT operators easy, and allows the computation of IFTs in systems with scarce memory. We evaluate the PIFT for two image processing applications: watershed segmentation and Euclidean distance transforms.

Keywords: graph algorithms, parallel algorithms, image foresting transform, distance transforms.

1. Introduction

The Image Foresting Transform (IFT) [9] is a graph-based framework for the design and implementation of image processing operators. It reduces image processing operations, such as watersheds [3, 15], morphological reconstructions [8], skeletonization [7] and distance transforms [5], to the computation of a minimum-cost path forest over an implicit graph representation of the image. The IFT runs in linear time, but it does not take advantage of parallel and distributed computer systems. Its data structures also require considerable memory space [10], and this can be a limitation to the processing of large 3D images.

In this work we present the Partitioned IFT, an algorithm that computes any IFT as a set of independent IFTs over partitions of the input image. Both time and memory required to compute the IFT of each partition are proportional to the size of that partition. The minimum-cost path forests of the partitions are merged by fast differential IFTs [6]. This scheme provides the means to take advantage of parallel and distributed computer systems (by assigning each partition's IFT to a different central processing unit (CPU)) and to allow the computation of IFTs with a reduced memory footprint (by computing partition forests sequentially).

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

2. Related works

2.1 Related algorithms

Moga et al. [12] presented two parallel watershed algorithms that treat the image as a graph and perform independent flooding simulations in image partitions. Parallel flooding simulations are repeated while plateaus overflow to adjacent partitions. The same group [13] presented a similar parallel algorithm for the computation of the watershed-from-markers transform. Both works achieve scalable speedups in parallel architectures, but the speedup factor does not scale linearly with the number of processors. Moga et al. [12] achieve speedup factors[1] around 2 for 4-CPU systems, and 3.5 for 8-CPU systems. Bruno and Costa [4] present a distributed algorithm for the computation of Euclidean distance transforms (EDT) based on morphological dilations. Their algorithm achieves a speedup factor of 3.5 on a 4-CPU system.

2.2 The image foresting transform

The IFT algorithm is essentially Dijkstra's algorithm [1], modified for multiple sources and general path cost functions [9]. The image is interpreted as a directed graph whose nodes are the pixels. The edges are defined implicitly by an *adjacency relation* $\mathcal{A}$. Tree roots are drawn from a set $\mathcal{S}$ of *seed nodes* and path costs are given by a *path cost function* f. We use $P^*(s)$ to denote the current path reaching pixel s, $\langle s \rangle$ to denote a *trivial path* containing a single node, and $\langle s, t \rangle$ to denote the edge from pixel s to pixel t. $P^*(s) \cdot \langle s, t \rangle$ is the path that results from the concatenation of $P^*(s)$ and an edge $\langle s, t \rangle$.

The choice of $\mathcal{A}$, $\mathcal{S}$ and f define an IFT operator. The IFT algorithm can compute by ordered propagation any forest property that uses the seed set as reference. Usually, the IFT computes 4 maps: the cost map C stores the cost of the optimal path that reaches each pixel, the predecessor map P stores the predecessor of each pixel in the forest, the root map R stores the root of each pixel's optimal path, and the label map L stores object labels for each pixel. Algorithm 1 below computes the IFT.

Algorithm 1. IFT.

INPUT:	*Image* $\mathbf{I}$*, Path-cost function* f*, Adjacency relation* $\mathcal{A}$*, Seed set* $\mathcal{S}$ *and Seed label map* L_0*.*
OUTPUT:	*Cost map* C*, Predecessor map* P*, Root map* R *and Label map* L*.*
AUXILIARY:	*Priority queue* Q*.*

1. Set $Q \leftarrow \emptyset$.

[1] The speedup factor of a parallel algorithm on an n-CPU parallel system is calculated as $\frac{t_1}{t_N}$, where t_1 is the time required to perform the computation on a single-CPU system, and t_N is the time required to perform the computation on an n-way system.

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

2. **For each** pixel $s \in \mathbf{I} \setminus \mathcal{S}$, **do**
3. Set $C(s) \leftarrow \infty$, $P(s) \leftarrow nil$, $R(s) \leftarrow s$ and $L(s) \leftarrow nil$.
4. **For each** pixel $s \in \mathcal{S}$, **do**
5. Set $C(s) \leftarrow f(\langle s \rangle)$ and $L(s) \leftarrow L_0(s)$.
6. Insert s in Q.
7. **While** $Q \neq \emptyset$, **do**
8. Remove a pixel s from Q such that $C(s)$ is minimum.
9. **For each** t such that $(s,t) \in \mathcal{A}$, **do**
10. Compute $cost \leftarrow f(P^*(s) \cdot \langle s,t \rangle)$.
11. **If** $cost < C(t)$ **then**
12. **If** $t \in Q$ **then** remove t from Q.
13. Set $P(t) \leftarrow s$, $C(t) \leftarrow cost$, $L(t) \leftarrow L(s)$, $R(t) \leftarrow R(s)$.
14. Insert t in Q.

Lines 1–3 set the forest to an initial state where every node's optimum path is a trivial path with infinite cost. Lines 4–6 insert the seed pixels in the priority queue with a trivial path cost computed by f, and initialize seed labels for ordered propagation. The loop of Lines 7–14 uses the priority queue to propagate the optimum paths and conquer the entire image. As long as f is finite and smooth [9], an optimum path with finite cost will be assigned to all pixels connected to $\mathcal{S}$. Once a pixel is removed from the queue (Line 8), it is never inserted again. Therefore, the main loop is repeated $|\mathbf{I}|$ times. For integer path costs with limited increments, Q can be efficiently implemented such that insertions and removals take $O(1)$ time [1]. With small adjacency relations ($|\mathcal{A}| \ll |\mathbf{I}|$) and $O(1)$ queue operations, the IFT algorithm runs in $O(|\mathbf{I}|)$ time [9].

Two common path cost functions for IFT operators are f_{max} and f_{euc}, shown in Equations 1–2 below. Both f_{max} and f_{euc} are *smooth*, as required to ensure the correctness of the IFT [9].

$$f_{max}(\langle s_1, \ldots, s_n \rangle) = \begin{cases} max_{i=1}^{n} \left(I\left(s_i\right) \right) & \text{if } n > 1, \\ h(s_1) & \text{otherwise.} \end{cases} \tag{1}$$

$$f_{euc}(\langle s_1, \ldots, s_n \rangle) = \text{Euclidean distance between } s_1 \text{ and } s_n \tag{2}$$

where $I(s)$ is some value associated to pixel s (such as intensity or gradient intensity) and h is a handicap function for trivial paths. A watershed-from-markers transform can be implemented as an IFT where f is f_{max} (Equation 1), $h = 0$ (for marker imposition), $\mathcal{A}$ is an adjacency with radius between 1 and $\sqrt{2}$ and $\mathcal{S}$ contains the watershed markers [6, 9]. A classical watershed can be implemented using $f = f_{max}$, $h(s) = I(s) + 1$ and $\mathcal{S} = \mathbf{I}$ [11]. Function f_{euc} (Equation 2) allows the computation of distance transforms [5], discrete Voronoi diagrams, skeletonizations and shape saliences [2, 7, 9, 14].

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

2.3 The differential image foresting transform

The differential IFT [6] (DIFT) was motivated by interactive 3D image segmentation applications where the user interactively selects the seed pixels. It is quite common for the user to add new seeds and remove previous ones based on the visualization of the segmentation result. The first IFT is computed by Algorithm 1 as usual, from a seed set $\mathcal{S}_0$. The maps C, P, R and L must be initialized to a forest of trivial paths with infinite costs before the first DIFT is computed. Given a set $\mathcal{S}'$ of seeds to be added and a set $\mathcal{S}''$ of tree roots to be removed, the DIFT computes the optimum path forest for the effective seed seet $\mathcal{S}_1 = (\mathcal{S}_0 \setminus \mathcal{S}'') \cup \mathcal{S}'$. The DIFT processes only pixels affected by the seed set editing, and runs in sublinear time. Instead of providing $\mathcal{S}''$ directly, the DIFT takes a set $\mathcal{M}$ of removal markers, and $\mathcal{S}''$ is computed as the set of roots of the pixels in $\mathcal{M}$. Algorithm 2 below is the main DIFT algorithm. The DIFT-TreeRemoval subroutine referenced in Line 2 visits all pixels that belong to removed trees, sets their optimum paths to trivial paths with infinite costs (forcing their recalculation by Algorithm 2), and builds the set $\mathcal{F}$ of frontier pixels.

Algorithm 2. DIFT.

INPUT:	*Image* $\mathbf{I}$*, Cost map* C*, Predecessor map* P*, Root map* R*, Label map* L*, Path-cost function* f*, Adjacency relation* $\mathcal{A}$*, Set* $\mathcal{S}'$ *of new seed pixels, Set* $\mathcal{M}$ *of marking pixels, Seed label map* L_0*.*
OUTPUT:	C*,* P*,* R *and* L*.*
AUXILIARY:	*Priority queue* Q*, Frontier set* $\mathcal{F}$*.*

```
1.  Set Q ← ∅.
2.  (C, P, F) ←DIFT-TreeRemoval(C, P, R, L, A, M).
3.  F ← F \ S'.
4.  While S' ≠ ∅, do
5.      Remove any t from S'.
6.      If f(⟨t⟩) < C(t) then
7.          Set C(t) ← f(⟨t⟩), R(t) ← t, L(t) ← L0(t), P(t) ← nil.
8.          Set F ← F ∪ {t}.
9.  While F ≠ ∅, do
10.     Remove any t from F and insert t in Q.
11. While Q ≠ ∅, do
12.     Remove a pixel s from Q, such that C(s) is minimum.
13.     For each t such that (s, t) ∈ A, do
14.         Compute cost ← f(P*(s) · ⟨s, t⟩).
15.         If cost < C(t) or P(t) = s then
16.             If t ∈ Q then remove t from Q.
17.             Set P(t) ← s, C(t) ← cost, R(t) ← R(s), L(t) ← L(s).
18.             Insert t in Q.
```

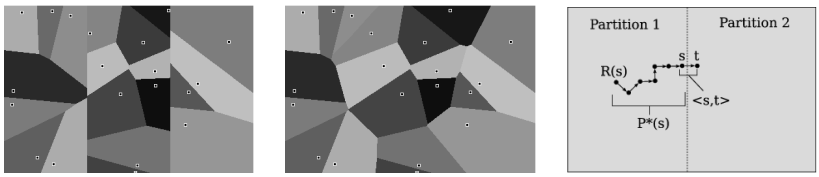
Lines 2–3 compute a set $\mathcal{F}$ of frontier pixels that belong to non-removed trees but share edges with pixels in removed trees. Lines 4–10 insert the

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

new seeds and the frontier pixels in the queue. Lines 11–18 are very much like the main loop of the IFT Algorithm (Algorithm 1), except for the condition $P(t) = s$ in Line 14, which forces the update of all pixels that had their optimum paths modified. The result of the DIFT is an optimum path forest for the "effective seed set" $\mathcal{S}_1 = (\mathcal{S}_0 \setminus \mathcal{S}'') \cup \mathcal{S}'$.

3. The partitioned image foresting transform

In the Partitioned IFT (PIFT), we split the input image and seed set in N_P *partitions*. The number of partitions can be chosen to match the number of available processing nodes, or so that the computer system has enough memory to run the IFT algorithm on each image partition. Partitions do not need to be equally sized. We compute independent IFTs on each partition. At this point, we have an optimum forest that ignores the inter-partition edges of the graph. Figure 1(a) shows an example of this partial result for the EDT using a set of random pixels as seeds and 3 partitions. To allow propagation through the inter-partition graph edges, we consider the paths obtained by the concatenation of each edge $\langle s, t\rangle$ to $P^*(s)$ (Figure 1(c)). When $f(P^*(s) \cdot \langle s, t\rangle)$ is less than the current cost of t, or the edge was part of the destination pixel's previous optimal path, the endpoint is added as seed in a differential IFT so that it can be propagated. If more than one inter-partition edge share a same endpoint t, the one that provides the lower path cost $P^*(t)$ is propagated. A new iteration of differential IFTs is computed for each partition. The PIFT halts when no inter-partition edge satisfies the criteria for addition. Figure 1(b) shows the complete EDT, obtained after 2 iterations over the 3 partitions.



(a) (b) (c)

Figure 1. Labels of an EDT with the Partitioned IFT: (a) Partial result after the first iteration and (b) final result after the second iteration. (c) PIFT notation: $\langle s, t\rangle$ is an inter-partition edge, $P^*(s)$ is the optimum path assigned to s, and $R(s)$ the root of $P^*(s)$.

The differential IFTs used in the Partitioned IFT always have an empty set of removal markers. The Partition-IFT algorithm below (Algorithm 3) computes the IFT within a partition. It is essentially the differential IFT algorithm without tree removal, and with special treatment of inter-partition edges.

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

Algorithm 3. PARTITION-IFT.

INPUT: *Image partition* $\mathbf{I}'$*, Cost map* C*, Predecessor map* P*, Root map* R*, Label map* L*, Path-cost function* f*, Adjacency relation* $\mathcal{A}$*, Set* $\mathcal{S}$ *of seed pixels, Seed label map* L_0*, Set* $\mathcal{E}_I$ *of incoming inter-partition edges.*
OUTPUT: *Maps* C*,* P*,* R*,* L *and Set* $\mathcal{E}_O$ *of outgoing inter-partition edges.*
AUXILIARY: *Priority queue* Q*.*

```
1.  Set Q ← ∅, E_O ← ∅.
2.  If S ≠ ∅ then
3.      For each pixel s ∈ I', do
4.          Set C(s) ← ∞, P(s) ← nil, R(s) ← s and L(s) ← nil.
5.      For each pixel s ∈ S, do
6.          Set C(s) ← f(⟨s⟩) and L(s) ← L_0(s).
7.          Insert s in Q.
8.  For each edge ⟨s,t⟩ ∈ E_I, do
9.      Compute cost ← f(P*(s) · ⟨s,t⟩).
10.     If cost < C(t) or P(t) = s then
11.         Set C(t) ← cost, P(t) ← s, R(t) ← R(s) and L(t) ← L(s).
12.         Insert t in Q.
13. While Q ≠ ∅, do
14.     Remove a pixel s from Q, such that C(s) is minimum.
15.     For each t such that (s,t) ∈ A, do
16.         If t ∈ I' then
17.             Compute cost ← f(P*(s) · ⟨s,t⟩).
18.             If cost < C(t) or P(t) = s then
19.                 If t ∈ Q then remove t from Q.
20.                 Set P(t) ← s, C(t) ← cost, R(t) ← R(s), L(t) ← L(s).
21.                 Insert t in Q.
22.         Else Insert ⟨s,t⟩ in E_O.
```

The DIFT is unable to tell whether the algorithm is on the first iteration, therefore the initial state of the forest must be set before the first iteration. In the PIFT, the seed set $\mathcal{S}$ will only be non-empty in the first iteration. We use this property to initialize the partition's forest to trivial paths with infinite costs in Lines 2–4. Lines 5–7 queue and initialize the seed pixels in the same way the IFT does. Lines 8–12 process the incoming inter-partition edges $\mathcal{E}_I$. Edges that offer lower costs to their endpoints or belonged to the previous forest are queued for propagation. If multiple edges in $\mathcal{E}_I$ reach the same endpoint, the edge that provides the lower cost for the endpoint takes precedence. The main loop in Lines 13–22 is very similar to the main loop of the DIFT, with the addition of the partition test $t \in \mathbf{I}'$ in Line 16. Edges within the current partition are processed normally. Inter-partition edges are added to the outgoing edge set $\mathcal{E}_O$ (Line 22). Note that the cost computation in Line 9 may require additional information about $P^*(s)$, which can contain pixels of several partitions. All path information required to compute $f(P^*(s) \cdot \langle s,t \rangle)$ must be passed along with the set $\mathcal{E}_I$.

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

For f_{max}, only $C(s)$ is required. For f_{euc}, only $R(s)$ is required. Since $L(s)$ may be propagated in Line 11, it must also be part of the input. Passing each element of $\mathcal{E}_I$ as $\{s, t, C(s), R(s), L(s)\}$ is enough to compute the PIFT with either f_{max} or f_{euc}. The PIFT algorithm (Algorithm 4) that computes the IFT of an image $\mathbf{I}$ from its partitions is shown below.

Algorithm 4. PARTITIONED IFT.

INPUT: *Image* $\mathbf{I}$*, Path-cost function* f*, Adjacency relation* $\mathcal{A}$*, Set* $\mathcal{S}$ *of seed pixels, Seed label map* L_0*, Number of partitions* N_P*.*
OUTPUT: *Cost map* C*, Predecessor map* P*, Root map* R*, Label map* L*.*
AUXILIARY: *Edge sets* $\mathcal{E}$*,* $\mathcal{E}'$*,* $\mathcal{E}''$ *and* $\mathcal{E}'''$*, Seed set* $\mathcal{S}'$*.*

1. Set $\mathcal{E} \leftarrow \emptyset$.
2. Split $\mathbf{I}$ in N_P partitions $\mathbf{I}[1] \dots \mathbf{I}[N_P]$.
3. **For** $i = 1$ **to** N_P, **do**
4. Set $\mathcal{S}' = \{s \mid s \in \mathcal{S} \wedge s \in \mathbf{I}[i]\}$.
5. Set $(C[i], P[i], R[i], L[i], \mathcal{E}') \leftarrow$ PARTITION-IFT$(\mathbf{I}[i], C[i], P[i], R[i], L[i], f, \mathcal{A}, \mathcal{S}', L_0, \emptyset)$.
6. Set $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}'$.
7. **Repeat**
8. Set $\mathcal{E}''' \leftarrow \emptyset$.
9. **For** $i = 1$ **to** N_P, **do**
10. Set $\mathcal{E}'' = \{\langle s, t\rangle \mid \langle s, t\rangle \in \mathcal{E} \wedge t \in \mathbf{I}[i]\}$.
11. Set $(C[i], P[i], R[i], L[i], \mathcal{E}') \leftarrow$ PARTITION-IFT$(\mathbf{I}[i], C[i], P[i], R[i], L[i], f, \mathcal{A}, \emptyset, nil, \mathcal{E}'')$.
12. Set $\mathcal{E}''' \leftarrow \mathcal{E}''' \cup \mathcal{E}'$.
13. Set $\mathcal{E} \leftarrow \mathcal{E}'''$.
14. **Until** $\mathcal{E} = \emptyset$.
15. Set $C \leftarrow \cup_{i=1}^{N_P} C[i]$, $P \leftarrow \cup_{i=1}^{N_P} P[i]$, $R \leftarrow \cup_{i=1}^{N_P} R[i]$ and $L \leftarrow \cup_{i=1}^{N_P} L[i]$.

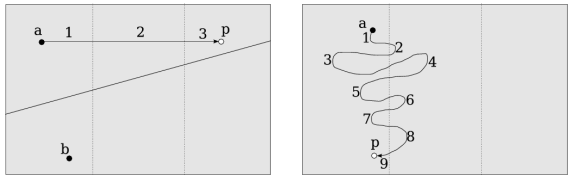
Lines 1–2 initialize the inter-partition edge set $\mathcal{E}$ and split the input image in N_P partitions. The loop in Lines 3–6 run the first IFT iteration on each partition. All inter-partition edges are accumulated in the set $\mathcal{E}$. The loop in Lines 7–14 run the remaining IFT iterations on the partitions, until no propagation occurs and the set $\mathcal{E}$ of inter-partition edges is empty (Line 14).

For parallel architectures, both loops (Lines 3–6 and 7–14) can be done in parallel. For distributed systems, the executions of Partition-IFT (Algorithm 3) can be performed as remote procedure calls. Note that the partitioned maps ($C[i]$, $P[i]$, $R[i]$ and $L[i]$) are only needed at the end of the algorithm, to compose the final IFT maps. In a distributed implementation, these maps can be kept on the remote processing nodes and do not need to be transferred at each call to Partition-IFT, as they are not modified by the caller.

Performance Considerations. The overall number of pixels processed by the PIFT is larger than $|\mathbf{I}|$. After the loop of Lines 3–6 of Algorithm 4,

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

the PIFT has already processed $|\mathbf{I}|$ nodes. However, the number of pixels processed by the loop of Lines 7–14 decreases at each iteration, and the algorithm converges rapidly to the optimum path forest. The number of PIFT iterations — i.e., one iteration of the loop of Lines 3–6 plus the number of iterations of the loop of Lines 7–14 — is bounded by the maximum number of inter-partition edges contained by an optimum path, plus one. Each inter-partition edge postpones the resolution of the optimum path to the next PIFT iteration. Figure 2 illustrates some examples. In an Euclidean distance transform (Figure 2(a)), all paths flow away from the roots, and a path may cross at most $N_P - 1$ partition boundaries, requiring at most N_P PIFT iterations. For path cost functions like f_{max}, there is no restriction to the shape of optimum paths, and cases like the one in Figure 2(b) can occur. However, as the number of iterations increases, the number of pixels processed by each iteration tends to decrease, and the PIFT converges more rapidly to the optimum forest.



(a) (b)

Figure 2. Partition crossings and PIFT iterations: In the PIFT-EDT, paths cross at most $N_P - 1$ partition boundaries. In (a), $P^*(p)$ crosses 2 boundaries to reach p from a. The numbers are the iteration in which the path segment is propagated. (b) For general path-cost functions, a path may cross partition boundaries several times.

4. Experimental results

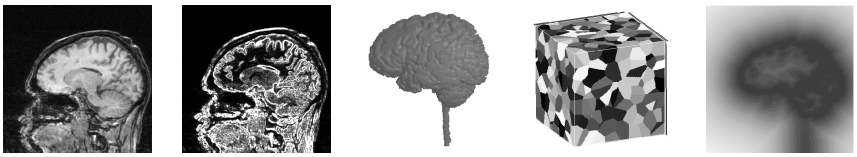
We implemented the PIFT as a client-server system, with a simple TCP stream-based protocol for communication between the master client that executes Algorithm 4 and the distributed servers that execute Algorithm 3. In our implementation, the image is always split in equal-sized partitions, using the x coordinate to separate partitions (such as in Figure 1(a)). We chose 3 applications to evaluate the PIFT:

1. WS-BRAIN: Watershed-based segmentation of a 3D MR image of the brain, using $f = f_{max}$ and $\mathcal{A}$ =6-neighborhood adjacency. Seeds were selected interactively in the background and in the brain. The gradient intensity was computed by a Gaussian enhacement filter followed by morphological gradient computation [6]. The size of the image is $356 \times 356 \times 241$, with a voxel size of $0.70mm^3$ (Figure 3(a–c)).

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

2. EDT-RND: Euclidean distance transform of 1000 random points within a 256^3 volume, using $f = f_{euc}$ and $\mathcal{A}$ =26-neighborhood (Figure 3(d)).

3. EDT-BRAIN: Euclidean distance transform using the border of the brain object (segmented in the first application) as seed set ($|\mathcal{S}| = 355,556$). $f = f_{euc}$, $\mathcal{A}$ =26-neighborhood and volume size is $356 \times 356 \times 241$ (Figure 3(e)).



(a) (b) (c) (d) (e)

Figure 3. Images from the evaluation applications: (a) Slice from the WS-BRAIN input image. (b) gradient intensity of (a). (c) 3D renderization of the WS-BRAIN result. (d) Visualization of the discrete Voronoi diagram, result of the EDT-RND. (e) Slice from the distance map computed in EDT-BRAIN.

First, we measured the processing overhead of the PIFT as the number of partitions (N_P) increases. We computed the 3 applications with the PIFT, using from 1 to 10 partitions. Table 1 and Figure 4 present the number of nodes processed in each case and the upper bound for the speedup factor. These results indicate that a 10-way parallel system may be able to offer a speedup factor of 6.60 to the EDT computation, and a factor of 2.34 to the Watershed transform on these instances of problems.

The EDT computations required at most 4 iterations before halting. PIFTs based on f_{max} are less efficient, since they allow free-form paths that can traverse several partitions. This can be noticed by the irregularity and increased slope of the plot in Figure 4(b), as compared to Figure 4(a). The WS-BRAIN PIFTs required at most 23 iterations to converge. The number of processed nodes grows linearly with the number of partitions. In real data with non-uniform distributions (WS-BRAIN and EDT-BRAIN), bad choices of partition boundaries may increase the number of processed nodes, such as in the $N_P = 4$ and $N_P = 8$ cases of EDT-BRAIN and $N_P = 5$ of WS-BRAIN.

In a second set of experiments we used the PIFT to compute EDT-RND, EDT-BRAIN and WS-BRAIN in two parallel systems: a PC with 2 CPUs (Athlon MP 1800+@1150 MHz) and 2 GB of RAM, and a Compaq AlphaServer GS140 6/525 with 10 CPUs (Alpha EV6@525 MHz) and 8 GB of RAM. Table 2 presents the results. On the EDT applications, we achieved speedup factors very close to the measured upper bounds (Table 1) for $N_P = 2$ and $N_P = 4$. On other hand, there was little or no speedup for

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

Table 1. Number of processed nodes and upper bound for the speedup factor in each application, using up to 10 partitions.

N_P	**WS-BRAIN**		**EDT-RND**		**EDT-BRAIN**	
	Nodes	Speedup	Nodes	Speedup	Nodes	Speedup
1	30.5×10^6	1.00	16.8×10^6	1.00	30.5×10^6	1.00
2	48.6×10^6	1.25	17.3×10^6	1.94	31.5×10^6	1.93
3	59.0×10^6	1.55	17.7×10^6	2.84	34.2×10^6	2.67
4	62.7×10^6	1.94	18.2×10^6	3.69	39.6×10^6	3.08
5	76.7×10^6	1.98	18.6×10^6	4.51	37.8×10^6	4.03
6	75.1×10^6	2.43	19.2×10^6	5.25	38.7×10^6	4.72
7	92.2×10^6	2.31	19.7×10^6	5.96	42.8×10^6	4.98
8	98.1×10^6	2.48	20.1×10^6	6.68	46.8×10^6	5.21
9	106.5×10^6	2.57	20.5×10^6	7.37	42.2×10^6	6.50
10	130.2×10^6	2.34	21.0×10^6	8.00	46.2×10^6	6.60

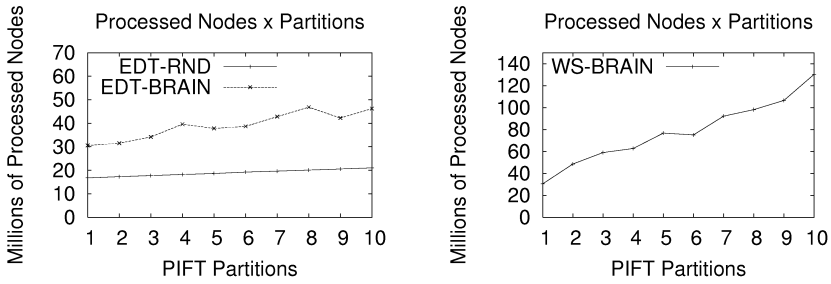


Figure 4. Number of processed nodes vs. number of partitions for (a) EDT-RND, EDT-BRAIN and (b) WS-BRAIN.

the watershed application. Our prototype implementation uses a naive communication protocol with no data compression. Besides that, the edge set transfers of Lines 5 and 11 of Algorithm 4 were implemented in a sequential way, and instances with a large number of partitions and/or a large number of PIFT iterations (such as WS-BRAIN with $N_P = 10$) performed poorly because the CPUs remained idle while waiting for the client to complete the sequential edge set transfers.

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10–13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

Table 2. PIFT performance on two parallel computer systems. Times are given in seconds.

System	N_P	WS-BRAIN		EDT-RND		EDT-BRAIN	
		Time	Speedup	Time	Speedup	Time	Speedup
Dual Athlon	1	258.1	1.00	195.9	1.00	459.5	1.00
	2	242.9	1.06	106.2	1.84	246.3	1.87
10-CPU GS140	1	280.6	1.00	228.8	1.00	611.4	1.00
	2	284.3	0.99	126.6	1.81	324.2	1.89
	4	226.2	1.24	73.0	3.13	274.3	2.23
	8	249.3	1.13	49.3	4.64	214.1	2.86
	10	336.4	0.83	47.9	4.78	197.7	3.09

5. Conclusion and future works

We introduced the Partitioned Image Foresting Transform, an algorithm that computes minimum-cost path forests as a set of independent DIFTs [6, 9] in partitions of the input image. The PIFT is useful for taking advantage of parallel computer systems and for computing IFTs in computer systems with limited memory, such as handhelds and embedded systems. The PIFT is applicable to any IFT-based operator, and therefore can be readily employed to parallelize morphological reconstructions [8], watershed transforms [2,3,6,15], distance transforms [5,9] and skeletonizations [7,14], among other operators. It is a trend in microprocessor technology to compensate CPU speed limitations by producing multi-core CPUs. The PIFT is an important contribution that allows existing image processing applications to use modern hardware efficiently with minimum effort.

We implemented a prototype PIFT system with a simple client-server architecture built on top of TCP streams. Even with no data compression and with some inneficient network operations, we achieved speedup factors very close to the expected upper bounds for EDT operations. PIFT-based watershed segmentation performed poorly due to the inneficiency of edge set transfers in our prototype. With a better protocol, the PIFT should be able to reach speedup factors closer to the upper bounds in Table 1.

Future works include: development of better protocols for implementation of the PIFT in parallel systems, evaluation of the speedup bounds for specific operators – such as the watershed transform – and investigation of enhancements to the PIFT such as partitioning schemes and iteration scheduling among nodes.

Proceedings of the 8th International Symposium on Mathematical Morphology,
Rio de Janeiro, Brazil, Oct. 10 –13, 2007, MCT/INPE, v. 1, p. 425–436.
http://urlib.net/dpi.inpe.br/ismm@80/2007/04.13.17.56

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice-Hall, 1993.

[2] F. P. G. Bergo and A. X. Falcão, *Fast and automatic curvilinear reformatting of MR images of the brain for diagnosis of dysplastic lesions*, Proc. of 3rd Intl. Symp. on Biomedical Imaging (April 2006), 486–489.

[3] S. Beucher and F. Meyer, *The morphological approach to segmentation: The watershed transformation*, Marcel Dekker, 1993.

[4] O. M. Bruno and L. F. Costa, *A parallel implementation of exact Euclidean distance transform based on exact dilations*, Microprocessors and Microsystems **28** (April 2004), no. 3, 107–113.

[5] P. E. Danielsson, *Euclidean Distance Mapping*, Computer Graphics and Image Processing **14** (1980), 227–248.

[6] A. X. Falcão and F. P. G. Bergo, *Interactive Volume Segmentation with Differential Image Foresting Transforms*, IEEE Trans. on Medical Imaging **23** (2004), no. 9, 1100–1108.

[7] A. X. Falcão, L. F. Costa, and B. S. da Cunha, *Multiscale skeletons by image foresting transform and its applications to neuromorphometry*, Pattern Recognition **35** (April 2002), no. 7, 1571–1582.

[8] A. X. Falcão, B. S. da Cunha, and R. A. Lotufo, *Design of connected operators using the image foresting transform*, Proc. of SPIE on Medical Imaging **4322** (February 2001), 468–479.

[9] A. X. Falcão, J. Stolfi, and R. A. Lotufo, *The Image Foresting Transform: Theory, Algorithms, and Applications*, IEEE Trans. on Pattern Analysis and Machine Intelligence **26** (2004), no. 1, 19–29.

[10] P. Felkel, M. Bruckschwaiger, and R. Wegenkittl, *Implementation and Complexity of the Watershed-from-Markers Algorithm Computed as a Minimal Cost Forest*, Computer Graphics Forum (Eurographics) **20** (2001), no. 3, C26–C35.

[11] R. A. Lotufo and A. X. Falcão, *The ordered queue and the optimality of the watershed approaches*, Mathematical Morphology and its Applications to Image and Signal Processing **18** (June 2000), 341–350.

[12] A. N. Moga, B. Cramariuc, and M. Gabbouj, *Parallel watershed transformation algorithms for image segmentation*, Parallel Computing **24** (December 1998), no. 14, 1981–2001.

[13] A. N. Moga and M. Gabbouj, *Parallel Marker-Based Image Segmentation with Watershed Transformation*, Journal of Parallel and Distributed Computing **51** (May 1998), no. 1, 27–45.

[14] R. S. Torres, A. X. Falcão, and L. F. Costa, *A graph-based approach for multiscale shape analysis*, Pattern Recognition **37** (June 2004), no. 6, 1163–1174.

[15] L. Vincent and P. Soille, *Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations*, IEEE Trans. on Pattern Analysis and Machine Intelligence **13** (June 1991), no. 6.