cudaIFT: 180x Faster Image Foresting Transform for Waterpixel Estimation using CUDA

Henrique M. Gonçalves[1,3], Gustavo J. Q. de Vasconcelos[1,3], Paola R. R. Rangel[1,4], Murilo Carvalho[2], Nathaly L. Archilha[1] and Thiago V. Spina[1,*]

[1]*Brazilian Synchrotron Light Laboratory, Brazilian Center for Research in Energy and Materials, Campinas, SP, Brazil*

[2]*Brazilian Biosciences National Laboratory, Brazilian Center for Research in Energy and Materials, Campinas, SP, Brazil*

[3]*Institute of Computing, University of Campinas, Campinas, SP, Brazil*

[4]*Institute of Geosciences, University of Campinas, Campinas, SP, Brazil*

Keywords: Image Foresting Transform, GPU, Watershed, Image Segmentation, Superpixels.

Abstract: We propose a GPU-based version of the Image Foresting Transform by Seed Competition (IFT-SC) operator and instantiate it to produce compact watershed-based superpixels (*Waterpixels*). Superpixels are usually applied as a pre-processing step to reduce the amount of processed data to perform object segmentation. However, recent advances in image acquisition techniques can easily produce 3D images with *billions* of voxels in roughly 1 second, making the time necessary to compute Waterpixels using the CPU-version of the IFT-SC quickly escalate. We aim to address this fundamental issue, since the efficiency of the entire object segmentation methodology may be hindered by the initial process of estimating superpixels. We demonstrate that our CUDA-based version of the sequential IFT-SC operator can speed up computation by a factor of up to 180x for 2D images, with consistent optimum-path forests without requiring additional CPU post-processing.

1 INTRODUCTION

Image segmentation is a crucial task for many applications in computer vision and image analysis. It is a challenging process given the wide variety of objects that may be depicted in different imaging modalities (e.g., medical, natural, and geological images). To address this issue, several methods rely on performing a first level of image segmentation by generating *superpixels*. Superpixels divide an image into homogeneous regions such that adjacent pixels with similar intensities/colors are assigned to a same region. Semantic objects are then usually comprised by one or more of those regions. The object segmentation methods aim to classify and/or merge the superpixels to perform the final segmentation and separate the desired objects of interest (Stegmaier et al., 2016; Stegmaier et al., 2018; Rauber et al., 2013).

Superpixel estimation is important because (i) it permits image segmentation methods to aggregate local contextual information about the different regions of the objects (an object may contain very dissimilar parts), and because (ii) it may significantly reduce the computational effort for segmentation, since the amount of estimated superpixels is usually far lower than the number of pixels in the image. Hence, several methods have been proposed to extract superpixels from an image (Achanta et al., 2012; Lotufo et al., 2002; Alexandre et al., 2015; Machairas et al., 2015; Vargas-Muñoz et al., 2018; Felzenszwalb and Huttenlocher, 2004).

Superpixel extraction methods ought to strike a balance between achieving high boundary recall and obtaining compact regions (Vargas-Muñoz et al., 2018). The former evaluates how the superpixels capture the borders of the existing objects in the image, while the latter is concerned with producing regions with shape as regular as possible. Compactness is important because regular shaped regions facilitate the posterior extraction of features for object segmentation. Methods such as the Simple Linear Iterative Clustering (Achanta et al., 2012) (SLIC) have been proposed with compactness in mind, to counterbalance approaches that produce irregular regions such as the watershed transform (Falcão et al., 2004; Roerdink and Meijster, 2000). The latter, however, can usually achieve higher boundary recall at the expense of region compactness.

[*]Corresponding author

Gonçalves, H., Q. de Vasconcelos, G., Rangel, P., Carvalho, M., Archilha, N. and Spina, T.
cudaIFT: 180x Faster Image Foresting Transform for Waterpixel Estimation using CUDA.
DOI: 10.5220/0007402703950404
In *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2019)*, pages 395-404
ISBN: 978-989-758-354-4

More recently, some authors have devised hybrid methods that combine the characteristics of the superpixel estimation by the watershed transform and SLIC (Achanta et al., 2012) to leverage their complementary properties, such as the IFT-SLIC (Alexandre et al., 2015), the Compact Watershed (Neubert and Protzel, 2014), and the Waterpixels (Machairas et al., 2015) approaches. All of the aforementioned techniques rely on variants of the *seeded watershed transform*, which can be efficiently implemented by using the *Image Foresting Transform* (Falcão et al., 2004) (IFT), a tool for the design of image processing and pattern recognition operators based on *optimum connectivity*. The IFT works by promoting a competition among candidate pixels selected as the representatives of the superpixels, henceforth denoted as *seeds*, to conquer the remaining pixels of the image. The resulting regions can be tuned in the aforementioned hybrid approaches to exhibit either the compactness of SLIC superpixels or the superior boundary adherence of the watershed transform.

In this work, we propose a GPU-based version of the IFT by optimum seed competition operator (IFT-SC) and instantiate it to produce Waterpixels (Fig. 1). We aim to address one of the key issues overlooked by the previously stated hybrid approaches: superpixel estimation must be performed in timely fashion. Although linear-time sequential implementations for the IFT-SC exist (Falcão et al., 2004), the required computational time can quickly escalate with the increase in the size of the image. This is particularly true for 3D images, since recent image acquisition techniques (Costa et al., 2017) can easily produce images with *billions* of voxels in *a few seconds* (1500^3 voxel images in about $1-5s$). Hence, even though the dimensionality of the object segmentation problem is reduced when using superpixels, the efficiency of the entire approach can be hindered by the initial process of estimating them (Rauber et al., 2013).

The IFT-SC can be understood by making an analogy with the *ordered formation of communities*. Consider a population in which each individual has the desire of forming a community. Some natural born leaders (seeds) begin the process by offering a reward to their acquaintances to become part of their community. If the reward is higher than the acquaintance's desire of forming his/her own community, or the acquaintance's current reward if it is already part of one, the acquaintance accepts the invitation and changes communities. The communities grow through their members, who start offering rewards that are not higher than their own reward/desire. This process continues following the non-increasing order of reward/desire, until each individual is part of the community that offered the highest reward. In superpixel estimation, the population represents the image, the individuals are the pixels, and each computed community corresponds to a superpixel rooted at the seed pixel (community leader) selected as its representative.

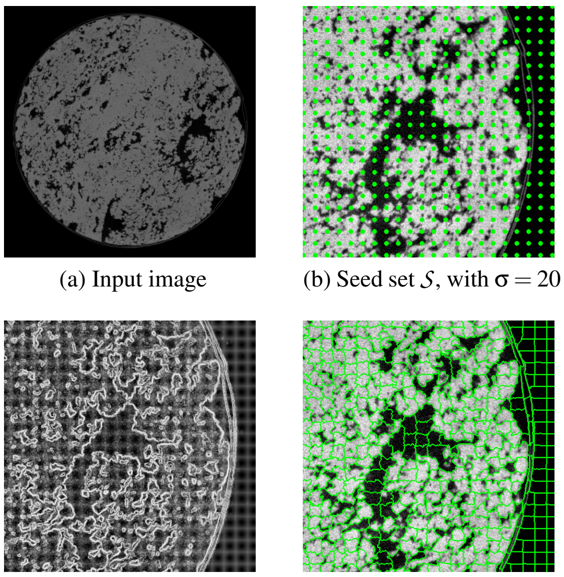
(a) Input image (b) Seed set $\mathcal{S}$, with $\sigma = 20$ (c) Border image B (d) Waterpixels by cudaIFT

Figure 1: Waterpixel computation on a 2D slice of a CT image of dolomite rock sample, using the cudaIFT parallel algorithm with regularly spaced seeds ($\sigma = 20$ and $k = 1000$, see Equations 1-2).

To implement the IFT-SC in GPU, we allow each individual of the population to begin conquering new individuals as soon as they become part of a growing community. Hence, we allow parallel competition between communities to occur, even if the reward that can be offered by their individuals is not yet optimum. In other words, we do not require the order of reward/desire to be respected. We then iterate the competition a few times to ensure that the rewards are as close to the expected optimum value as possible, thereby leading to a parallel version of the IFT algorithm. We have implemented our method using Nvidia's CUDA language and therefore name it *cudaIFT*.

cudaIFT is related to the parallel implementation of the Relative Fuzzy Connectedness algorithms proposed in (Wang et al., 2016; Zhuge et al., 2012). Those works tend to either parallelize the execution considering some order during optimum reward propagation (Zhuge et al., 2012), or heavily rely on global GPU memory with costly access (Wang et al., 2016). cudaIFT thus presents several important contributions:

1. It considerably improves the usage of GPU by making better use of CUDA threads during competition;
2. It makes heavy use of shared block memory instead of global GPU memory to increase throughput;
3. It is a parallel version of the IFT-SC operator, and therefore can be used in settings other than Waterpixel estimation.

We validate cudaIFT for the estimation of Waterpixels in 2D slices of a 3D Computed Tomography image, against the linear-time implementation of the sequential IFT algorithm (Falcão et al., 2004). cudaIFT computes accurate superpixels with consistent optimum-path forest, without requiring further CPU post-processing, while achieving speedups of up to roughly 180x over the CPU version.

The next sections are organized as follows. Section 2 presents an overview of background concepts that are necessary to understand Waterpixel estimation and the IFT algorithm. Section 3 provides the algorithmic details about the cudaIFT approach. We present our experimental validation in Section 4 and draw our final conclusions in Section 5.

2 BACKGROUND

Image segmentation provides high-level knowledge about the content of the image, by representing the objects of interest depicted in the image. An *image* $I : D_I \rightarrow \mathbb{V} \subseteq \mathbb{R}^m$ is a mapping function from the rectangular *image domain* $D_I \subset \mathbb{Z}^d$ to the m-dimensional real domain, such that m scalars are assigned to each pixel in $p \in D_I$. In this work, we are interested in grayscale images ($m = 1$) represented in two dimensional space ($d = 2$), making the array index p represent matricial image coordinates as $p = (x, y)$. The goal of image segmentation is to assign a label $L(p)$ to each pixel p in an image I corresponding to one out of c available objects of interest or the background ($L(p) = 0$).

In the following, we briefly describe some background concepts necessary to understand the superpixel estimation method named Waterpixels, and the sequential Image Foresting Transform algorithm. For simplicity, the aformentioned label assignment $L(p)$ shall refer to the computed c regions, with each region reprepresenting a superpixel.

2.1 Waterpixel Estimation

The Waterpixels approach was proposed in (Machairas et al., 2015) (Fig. 1). In order to strike a balance between compactness and boundary recall, the Waterpixels method performs the watershed transform on a specially crafted border image B, which takes into account the position of the seed pixels in a set $\mathcal{S}$ selected on a regular grid with spacing of σ pixels (Fig. 1b). Image B (Fig. 1c) combines the gradient magnitude of the input image $|\nabla I|$ and a distance map E from the seed set $\mathcal{S}$:

$$B(p) \quad = \quad |\nabla I(p)| + kE(p), \tag{1}$$

in which k is an application-dependent parameter that controls the tradeoff between compactness and boundary adherance, and E is computed as

$$E(p) \quad = \quad \frac{2}{\sigma} \min_{q \in \mathcal{S}} \{d(p,q)\}, \tag{2}$$

with $d(p,q) = ||p - q||$ being the euclidean distance between pixels p and q. If k is small in Eq. 1, the watershed transform is forced to adhere to the original boundaries of image I. When $k \rightarrow \infty$, the watershed transform becomes a Voronoi tessellation. It is worth noting that both the computation of the gradient image ∇I and the seed distance map E can be very efficiently done on the GPU using, in the former case, the Sobel operator for instance. Hence, those optimizations are not the focus of this paper.

2.2 The Image Foresting Transform

The Image Foresting Transform can be seen as generalization of Dijkstra's single source shortest path algorithm (Dijkstra, 1959) for multiple sources and a greater variety of connectivity functions (Falcão et al., 2004; Ciesielski et al., 2018). The IFT algorithm can be used to implement operators such as the watershed transform (IFT-SC (Falcão et al., 2004)), supervised (Papa et al., 2012) and unsupervised (Rocha et al., 2009) pattern classifiers, among others (Miranda and Mansilla, 2014).

The IFT interprets an image as graph, in which the pixels are the nodes and spatially adjacent elements are connected by an arc. The aforementioned ordered community formation analogy is therefore cast into a graph-partitioning problem (Fig. 2). We describe next the main elements required for using the IFT for watershed image segmentation.

2.2.1 Graphs from Images

Let $G = (\mathcal{N}, \mathcal{A}, w)$ denote a weighted graph, in which $\mathcal{N}$ is a set of elements taken as nodes and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$

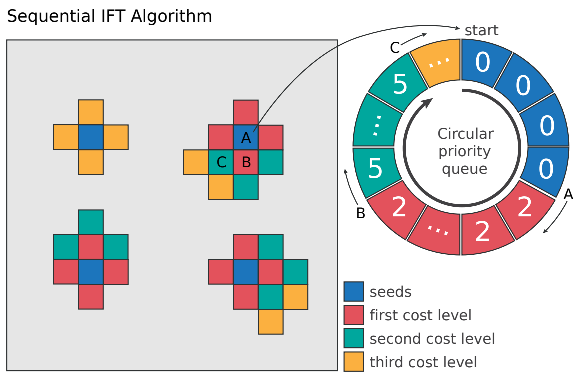

Figure 2: The Image Foresting Transform sequential algorithm diagram, depicting the intermediate result of the community formation procedure. The color code indicates the sequence in which the pixels will be evaluated, according to the final optimum path cost that they will have.

is a binary and *non-reflexive adjacency relation* between elements of $\mathcal{N}$ that form the arcs. We use $q \in \mathcal{A}(p)$ or $(p,q) \in \mathcal{A}$ to indicate that a node $q \in \mathcal{N}$ is adjacent to a node $p \in \mathcal{N}$. Each arc $(p,q) \in \mathcal{A}$ is given a fixed weight $w(p,q)$ that encodes the dissimilarity between adjacent nodes $p,q \in \mathcal{N}$.

For a given graph $G = (\mathcal{N},\mathcal{A},w)$, a *path* $\pi_q = \langle q_1,q_2,\ldots,q \rangle$ with terminus at a node q is simple when it is a sequence of distinct and adjacent nodes. A path $\pi_q = \pi_p \cdot \langle p,q \rangle$ indicates the extension of a path π_p by an arc (p,q) and a path $\pi_q = \langle q \rangle$ is said *trivial*. A connectivity function f assigns to any path π_q in the graph a value $f(\pi_q)$. A path π_q is optimum if $f(\pi_q) \leq f(\tau_q)$ for any other path τ_q in G.

In this work, we are interested in the traditional $f_{\max}$ connectivity function:

$$\begin{aligned} f_{\max}(\langle q \rangle) &= \begin{cases} 0 & \text{if } q \in \mathcal{S}, \\ +\infty & \text{otherwise} \end{cases} \\ f_{\max}(\pi_p \cdot \langle p,q \rangle) &= \max\{f_{\max}(\pi_p), w(p,q)\}. \end{aligned} \quad (3)$$

Function $f_{\max}$ essentially propagates the maximum edge weight along a path. By assigning initial connectivity values of 0 to trivial paths of nodes q belonging to a set of previously selected *seed pixels* $\mathcal{S}$, we ensure that the IFT will force the seeds to compete among themselves to conquer the remaining nodes of the graph $\mathcal{N} \setminus \mathcal{S}$. The IFT algorithm may consider several types of connectivity functions (Miranda and Mansilla, 2013; Miranda et al., 2012), although a profound study about other functions is outside the scope of this work.

2.2.2 Optimum Path Forest

Considering all possible paths with terminus at each node $q \in \mathcal{N}$, an optimum connectivity map $V(q)$ is created by

$$V(q) = \min_{\forall \pi_q \text{ in } (\mathcal{N},\mathcal{A},w)} \{f(\pi_q)\}. \quad (4)$$

The IFT solves the minimization problem above by computing an *optimum-path forest* — a function P which contains no cycles and assigns to each node $q \in \mathcal{N}$ either its predecessor node $P(q) \in \mathcal{N}$ in the optimum path or a distinctive marker $P(q) = nil \notin \mathcal{N}$, when $\langle q \rangle$ is optimum (*i.e.* q is said *root* of the forest). In essence, pixels in the seed set $\mathcal{S}$ become the roots of the forest due to the initialization of function $f_{\max}$ in Eq. 3. We present next the sequential IFT algorithm for solving the minimization problem of Eq. 4.

2.2.3 Sequential IFT Algorithm

Algorithm 1 presents the steps to compute the IFT following the minimization of Eq. 4. It relies on a priority queue Q to propagate optimum path costs in orderly fashion on weighted graph G.

Algorithm 1: Sequential IFT Algorithm.

```
INPUT:      Graph $G = (\mathcal{N},\mathcal{A},w)$ and connectivity function $f$.
OUTPUT:     Optimum-path forest $P$, its connectivity value map $V$
            and its root map $R$.
AUXILIARY:  Priority queue $Q$ and variable $tmp$.

1.  For each $q \in \mathcal{N}$, do
2.      $P(q) \leftarrow nil$, and $V(q) \leftarrow f_{\max}(\langle q \rangle)$.
3.      If $V(q) \neq +\infty$, then insert $q$ in $Q$.
4.  While $Q \neq \emptyset$, do
5.      Remove $p$ from $Q$ such that $V(p)$ is minimum.
6.      For each $q \in \mathcal{A}(p)$, such that $V(q) > V(p)$, do
7.          Compute $tmp \leftarrow f_{\max}(\pi_p \cdot \langle p,q \rangle)$.
8.          If $tmp < V(q)$, then
9.              If $V(q) \neq +\infty$, then remove $q$ from $Q$.
10.             Set $P(q) \leftarrow p$, $V(q) \leftarrow tmp$.
11.             Insert $q$ in $Q$.
```

Steps 1–3 initialize maps for trivial paths. The minima of the initial map V compete with each other and the ones in seed set $\mathcal{S}$ become roots of the forest. They are nodes with optimum trivial-path values, which are inserted in queue Q. The main loop computes an optimum path from the roots to every node p in a non-decreasing order of path value (Steps 4–11). At each iteration, a path of minimum value $V(p)$ is obtained in P when we remove its last pixel p from Q (Step 5). Ties are broken in Q using first-in-first-out policy. The remaining steps evaluate if the path that reaches an adjacent pixel q through p is cheaper than the current path with terminus q and update Q, $V(q)$, and $P(q)$ accordingly.

Seeded watershed segmentation using the IFT algorithm is known as the Image Foresting Transform by Seed Competition (IFT-SC) operator. In this work, it can be straightforwadly derived by considering a graph $G_I = (\mathcal{N}, \mathcal{A}, w)$ with the nodes $\mathcal{N} = D_I$ as the entire domain of image I and a spatial connectivity function $\mathcal{A} = \mathcal{A}_4$ connecting 4-adjacent pixels. To ensure that Waterpixels are computed through IFT-SC, the seed set $\mathcal{S}$ is selected as the superpixel seeds and the arc weights of graph G_I are assigned as $w(p,q) = B(q)$, both of which as previously defined in Section 2.1.

Since arc weights can be normalized to integer values ranging from $\{0, 1, \dots, K\}$, Dial's bucket sorting-based priority queue (Dial, 1969) is usually selected as Q in Algorithm 1. Hence, for sparse graphs as in the case of 4- or 8-connected image-graphs, the algorithm's complexity is $O(K + \mathcal{N})$, being essentially linear for reasonable values of K.

3 THE cudaIFT ALGORITHM

The main challenge of the parallelization of Algorithm 1 lies on the fact that the IFT computes optimum-path costs in greedy fashion, starting from the seeds and extending paths incrementally following the order given by the priority queue. One previously adopted strategy for implementing the IFT algorithm (Zhuge et al., 2012) on the GPU aimed to parallelize the competition respecting the sequence of a virtual priority queue. In essence, competition would take place simultaneously for pixels with the same cost level. Hence, pixels with the same color in the priority queue of Fig. 2 would be allowed to compete to extend their optimum-path trees, as long as their costs matched the current minimum value in the virtual priority queue. This approach severely hinders parallel competition, since the global sequence of the queue is still respected.

cudaIFT instead aims to allow as much competition to take place as possible, by understanding that competition occurs primarily between neighboring regions of the image when computing superpixels. Hence, cudaIFT divides the 2D image into a grid of CUDA blocks, with one CUDA thread per pixel, and keeps track of a pixel activation map. Each CUDA thread verifies if its pixel p is active in the map, and then tries to conquer adjacent pixels $q \in \mathcal{A}(p)$. Seeds are the first nodes that are activated before any competition takes place, and whenever a new pixel is conquered it is immediately activated to continue the optimum-path tree expansion process inside their CUDA thread blocks (Fig. 3).

Due to intrablock synchronization capabilities, we make use of atomic operations and allow competition to occur only inside CUDA blocks first. Then, we resolve interblock propagation of optimum-path costs (bottom of Fig. 3) and iterate the entire algorithm a few times until no pixel is active. This is similar in spirit to the approach in (Wang et al., 2016), although with a number of advantages. First, we make heavy use of the GPU's shared memory to avoid expensive global memory access. Second, we keep CUDA threads continuously checking if their pixel have been activated, until all of the threads in their block have concluded. Third, we compute not only the optimum path cost map V, but also the predecessor map P and label L consistently, with no need for posterior CPU corrections. This is very important, since the Relative Fuzzy Connectedness approach, parallelized in (Wang et al., 2016), actually only requires a consistent cost map V and therefore does not compute a predecessor map P. We detail in the next sections the cudaIFT algorithm.

3.1 Main Steps of cudaIFT

Algorithm 2 presents the main steps of cudaIFT. The algorithm starts by initializing the proper connectivity, predecessor, label, and activation maps by taking into account the seeds in set $\mathcal{S}$ and the connectivity function $f_{\max}$. The initialization kernel is a simple parallelization of a loop over the entire image and seed set. Then, the intrablock competition kernel is invoked to perform seed competition inside each CUDA thread block. In order to propagate the optimum-path costs between CUDA thread blocks, the appropriate kernels are called three times in the loop of Step 4, such that information is propagated between the corners of the blocks as pointed out in Fig. 3 and following (Wang et al., 2016). Finally, all steps are iterated M times to ensure that there are no more active pixels.

With each iteration, more and more pixels become active thereby ensuring that the wavefront of optimum path-costs is propagated even if a CUDA thread block originally has no seeds. Hence, although we focus on superpixel estimation cudaIFT can be used with an arbitrary seed set $\mathcal{S}$. It is important to note, however, that we divide interblock competition into two similar kernels for efficient processing of larger images, since rows and columns at the CUDA thread block borders are evaluated separately.

Algorithm 2: The cudaIFT Algorithm.

```
INPUT:      Seed set $\mathcal{S}$, optimum-path forest $P$, its connectivity value map $V$, and its label map $L$ in global GPU memory.
            Adjacency relation $\mathcal{A}$, and connectivity function $f_{\max}$.
            Number of iterations $M$.
OUTPUT:     Optimum-path forest $P$, its connectivity value map $V$, and its label map $L$ in CPU memory.
AUXILIARY:  Activation map $A$.
```

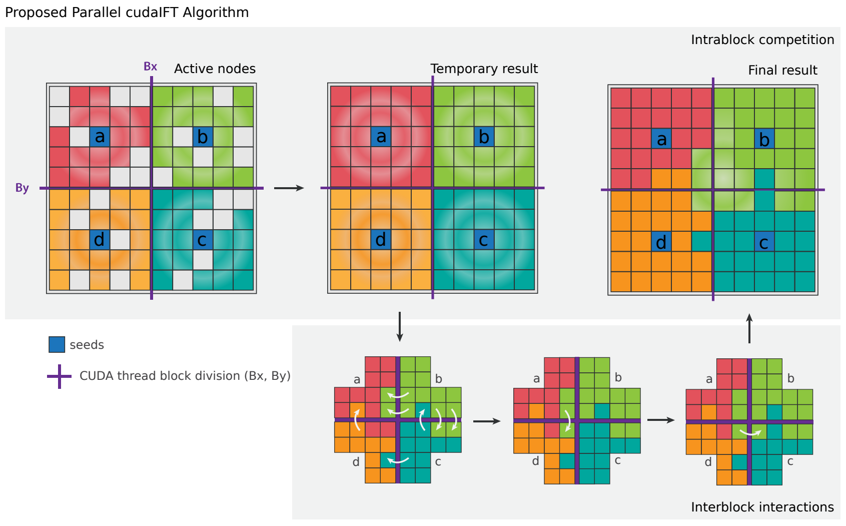

Figure 3: The proposed cudaIFT parallel algorithm diagram. **Top:** intrablock competition performed by active nodes in shared memory, for small CUDA thread blocks (5x5 threads) and a single seed selected in each block. The left image depicts active nodes during some step of the competition. The middle image presents the temporary result in shared memory after the first iteration of competition is done. The right image depicts the final result, after some iterations of intrablock and interblock competition are performed. **Bottom:** interblock competition performed by pixels on the boundary of CUDA blocks that were activated during intrablock competition. The arrows indicate one node conquering another one from a different block. We perform three interblock competition iterations for each intrablock competition to propagate information between the corners of four adjacent blocks (a, b, c, d), as can be seen at the center of the three images.

```
1. Initialization-Kernel($V$, $L$, $P$, $A$, $\mathcal{A}$, $\mathcal{S}$, $f_{\max}$).
2. For $i \in \{1,2,\ldots,M\}$ do
3.    Intrablock-Competition-Kernel($V$, $P$, $L$, $A$, $\mathcal{A}$, $f_{\max}$)
4.    For $j \in \{1,2,3\}$ do
5.       Interblock-H-Competition-Kernel($V$, $P$, $L$, $A$, $\mathcal{A}_h$, $f_{\max}$).
6.       Interblock-V-Competition-Kernel($V$, $P$, $L$, $A$, $\mathcal{A}_v$, $f_{\max}$).
7. Transfer maps $L$, $V$, and $P$ to CPU/main memory.
```

3.2 Intrablock Competition

The intrablock competition kernel is run for each pixel in the image domain $p \in D_I$. CUDA threads are spawned using a 32×32 block size with blocks covering the entire image. Algorithm 3 presents the intrablock competition kernel.

Algorithm 3: Intrablock Competition Kernel.

INPUT:	Global index of pixel p. Adjacency relation $\mathcal{A}$. Activation map A. Optimum-path forest P, its connectivity value map V, and its label map L in global memory. Connectivity function $f_{\max}$.
OUTPUT:	Updated values in global memory of: optimum-path forest P, connectivity value map V, label map L, and activation map A.
AUXILIARY:	Temporary variables *active*, *tmp*. Variables in shared memory: activation map A_s, cost map V_s, predecessor map P_s, label map L_s, mutex lock map K_s, and termination variable *s_done* .

```
1. Copy map values $L(p), V(p), P(p)$ from global to shared memory.
2. syncthreads().
3. While $s\_done \neq False$, do
4.    $V_p \leftarrow V_s(p)$.
5.    $L_p \leftarrow L_s(p)$.
6.    $s\_done \leftarrow 1$.
7.    $active \leftarrow A_s(p)$.
8.    Atomically set $A_s(p) \leftarrow False$.
9.    syncthreads().
10.   If $active = True$, then
11.      For each $q \in \mathcal{A}(p)$ do
12.         Compute $tmp \leftarrow f_{\max}(\pi_p \cdot \langle p,q \rangle)$.
13.         Try-Acquire-Lock($K_s(q)$).
14.         If $tmp < V_p$ or
15.            ($V_s(q) = tmp$, $P_s(q) = p$, $L_s(q) \neq L_p$), then
16.            Set $P_s(q) \leftarrow p, V_s(q) \leftarrow tmp$.
17.            $A_s(q) \leftarrow True, s\_done \leftarrow False$.
18.         Release-Lock($K_s(q)$)
19.      syncthreads().
20.   syncthreads().
21. Copy map values $L_s(p), V_s(p), P_s(p)$ from shared to global memory.
```

First, the current label, cost, and predecessor maps are copied to shared memory. Then, the kernel loops while the pixels of the threads of the corresponding CUDA block are active, given by shared variable $s\_done = True$. While that is true, a snapshot of the label and cost of the current pixel p is done, since they may be altered by other threads during competition, and the pixel p is checked for activation. If p is active (Step 10), then it will attempt to conquer neighboring nodes $q \in \mathcal{A}(p)$. In order to conquer a neighbor q, we need to ensure that only the current thread is modifying the adjacent node's connectivity, label, and predecessor map values.

We have thus created a mutex procedure that essentially tries to lock a map $K_s(q)$ in shared memory. If successful, then the algorithm evaluates if the computed cost value is lower than $V_s(q)$ and, if so, p is able to conquer q (Step 14). However, since competition occurs in parallel and a voxel q is activated as soon as it is reached, then sub-optimum path costs may be propagated inadvertently. This becomes a problem when ties in optimum costs exist, which may lead to inconsistent label maps. Hence, we borrow ideas from the differential IFT-SC operator (Falcão and Bergo, 2004), which is able to perform local corrections after additions/removal of seed pixels, and modify the test in Step 8 of the original IFT Algorithm 1 to take into account ties in the path costs being computed, as presented in Step 14 of Algorithm 3. Since the mutex locking procedure is expensive, as it involves a loop encompassing Steps 13-18 to atomically verify if the state of the locking map $K_s(q)$ is originally 0 and set it to 1 if true, in practice we repeat the test of Step 14 before Step 13 to evaluate only candidate neighbors. Block thread synchronizations are performed to ensure consistency of the competition.

3.3 Interblock Competition

The horizontal interblock competition procedure, given in Algorithm 4, is quite similar to the intrablock kernel. The main difference is that the competition is executed only once for each pixel p on the boundary of the CUDA thread blocks that divided the image into a regular grid. Only the boundary pixels activated during intrablock competition are eligible for interblock competition and to propagate information to adjacent blocks. Note that the vertical competition is essentially identical, what changes is that boundary pixels on the columns are evaluated instead.

Algorithm 4: Interblock Horizontal Competition Kernel.

INPUT:	Global index of pixel p at the boundary of a CUDA thread block from Algorithm 3. Adjacency relation in the horizontal axis $\mathcal{A}_h$. Activation map A, optimum-path forest P, connectivity value map V, label map L in global memory. Mutex lock map K_s in shared memory. Connectivity function $f_{\max}$.
OUTPUT:	Updated values in global memory of: optimum-path forest P, connectivity value map V, label map L, and activation map A.
AUXILIARY:	Temporary variables *active*, *tmp*.

```
1. $V_p \leftarrow V(p)$.
2. $L_p \leftarrow L(p)$.
3. $active \leftarrow A(p)$.
4. syncthreads().
5. If $active = True$, then
6.     For each $q \in \mathcal{A}_h(p)$ do
7.         Try-Acquire-Lock($K_s(q)$).
8.         Compute $tmp \leftarrow f_{\max}(\pi_p \cdot \langle p,q \rangle)$.
9.         If $tmp < V(q)$ or ($V(q) = tmp$, $P(q) = p$, $L(q) \neq L_p$), then
10.            Set $P(q) \leftarrow p, L(q) \leftarrow L_p, A(q) \leftarrow True$.
11.        Release-Lock($K_s(q)$)
```

In the first time that the intrablock competition kernel is executed, all block boundary pixels are activated, but as iterations of the Algorithm 2 occur, this number decreases until none is active. To this end, we actually deactivate boundary pixels once Algorithm 4 is executed both horizontally and vertically after the loop in Step 4 of Algorithm 2 is done, if the boundary pixels are not activated during interblock competition.

4 EXPERIMENTAL RESULTS

We have conducted experiments comparing the sequential linear-time implementation of the IFT-SC algorithm in C language, versus the cudaIFT algorithm for estimation of Waterpixels. We have selected 2D slices of a graycale 3D Computed Tomography image of a carbonate rock sample to perform the comparison, with original size of 1024^3 voxels. For each slice, we carried out the comparison using 4 different seed spacing steps, namely $\sigma = \{5, 10, 20, 30\}$, to understand how seed sparseness affects computation. Parameter k in Eq. 1 was set to 1000 for all experiments. The greater the value of σ the higher the number of iterations (parameter M in Algorithm 2) required for the result of the cudaIFT algorithm to match the result of the sequential IFT-SC. Hence, we executed up to $M = 30$ iterations but stopped as soon as the computed optimum-path cost maps V for both sequential and parallel algorithms matched. The slices were rescaled to encompass four different sizes ($1024^2, 1536^2, 2048^2$, and 4096^2 pixels), to evaluate the speedup factor of the cudaIFT algorithm with increasing amounts of pixels.

It is important to note that, due to the parallel nature of cudaIFT, the forest map P and label map L may differ from the result of the sequential IFT-SC due to the asynchronous propagation of optimum paths. However, the differences in label and predecessor maps only occur in zones where ties occur for the optimum-path costs in map V. Therefore, in practice any result in those regions corresponds to a valid optimum-path forest P and the only thing that should truly match is the optimum-path cost map V. Our verification further ensured the consistency of the optimum-path cost,

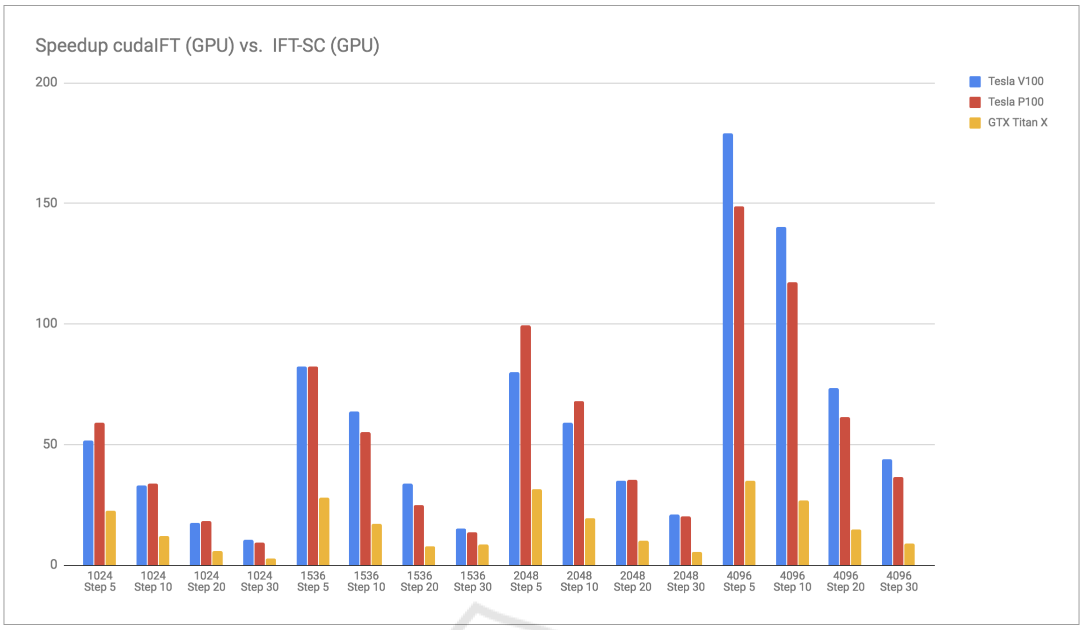

Figure 4: Average speedup factor for computing Waterpixels on 2D slices using cudaIFT vs. CPU IFT. Varying σ spacing (steps) are considered to evaluate how seed sparseness affects the computational times. Different image sizes are also considered ($|D_I| = \{1024, 1536, 2048, 4096\}$), for three different computer configurations with distinct GPUs (Nvidia Titan X, Nvidia Tesla P100, and Nvidia Tesla V100).

forest, and label maps computed by cudaIFT.

We have executed the aforementioned experiments 10 times for each slice and parameter setting, in order to obtain statistics about the time measurements. Our experiments were conducted using three different machines, each with a distinct GPU and configuration to assess the algorithm in different environments. Our first machine is a regular desktop with an Intel Core i7-4790 CPU running at 3.60GHz, 8 GB of memory RAM, Ubuntu 18.04 operating system, and Nvidia GeForce GTX Titan X GPU with 3072 CUDA Cores. The second is a server with an IBM Power 8 CPU running at 2.60GHz, 512 GB of memory RAM, Ubuntu 18.04 OS, and Nvidia Tesla P100 GPU with 3584 CUDA Cores. The final machine is a server with an IBM Power 9 CPU running at 2.3GHz, 512 GB of memory RAM, Ubuntu 18.04 OS, and Nvidia Tesla V100 GPU with 5120 CUDA cores. CUDA 9.2 or CUDA 10 were used in our experiments.

Figure 4 presents the average speedup factor for the previously stated experiments, across all machine configurations, without including CPU-GPU memory transfer as the IBM Power servers possess NVLink buses that could further increase the time difference w.r.t. the used desktop. As expected, cudaIFT performs significantly better for larger images and smaller step sizes. This occurs because smaller step sizes decrease the amount of competition required to segment the images and generate superpixels. The server running Nvidia Tesla V100 outperforms the others for 4096×4096 sized images, which can be explained by the fact that Nvidia's Volta architecture presents superior atomic operations to the Pascal architecture used for GTX Titan X and Tesla P100. In that case, that configuration may reach close to 180x speedup for seeds with step size of $\sigma = 5$, and roughly 140x speedup for $\sigma = 10$. Figure 5 depicts the average raw computational times for the 4096×4096 sized images for cudaIFT and IFT-SC in all three configurations and four step sizes. It can be seen that IFT-SC takes between $3.3s$ and $4.7s$ to execute, while the computational time for cudaIFT is always smaller than $0.5s$, reaching as low as $0.02s$.

For other image sizes, we see a somewhat competitive behavior between the P100 and V100 GPUs, since the clock of the corresponding CPUs is slightly lower for the latter. With respect to the Titan X GPU, the others perform significantly better since the Tesla P100 and Tesla V100 have compute capabilities of 60 and 70, respectively, versus 52 for the GTX Titan X. This implies that the atomic operations for the P100 is also superior, even though it has the same Pascal architecture as the GTX Titan X and only a little over 500 extra CUDA cores. Figure 6 confirms the aforementioned comparisons between GPUs and demonstrates that the cudaIFT runs usually from $2-5x$ faster in Tesla P100 and Tesla V100 than GTX Titan X. Similarly, Titan V100 is up to $1.8x$ faster than Te-

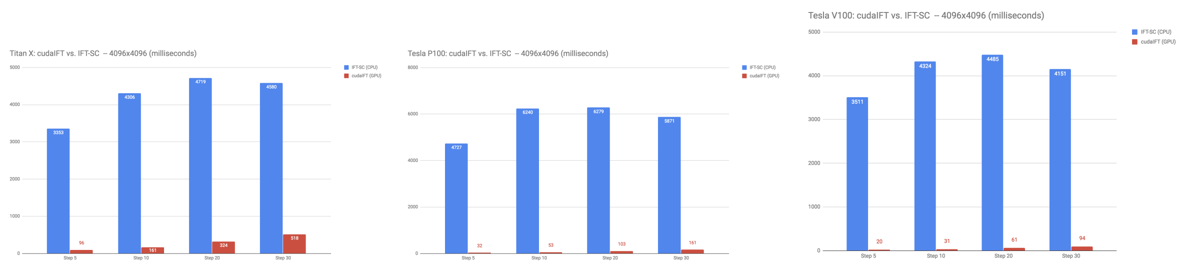

Figure 5: Raw computational times (in milliseconds) between cudaIFT and IFT-SC for image with size 4096×4096, for the three different machines used in this work.

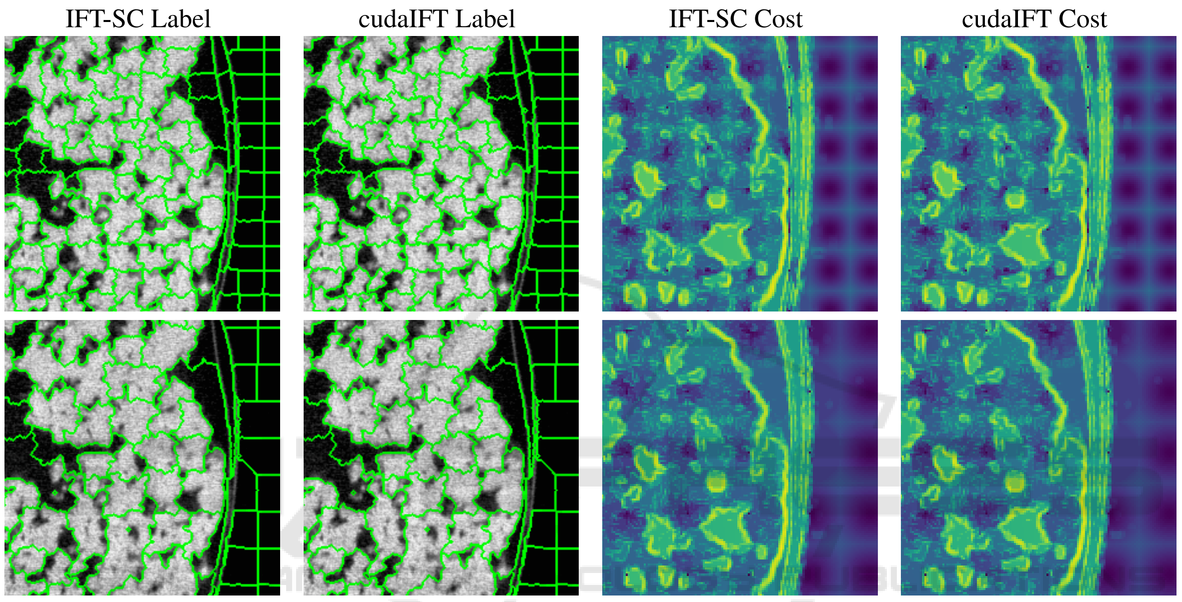

Figure 7: Comparison between the label and cost maps obtained via cudaIFT and IFT-SC, for $\sigma = 10$ (top), $\sigma = 20$ (middle), and $\sigma = 30$ (bottom). While the cost maps are identical, there are virtually no differences between the label maps.

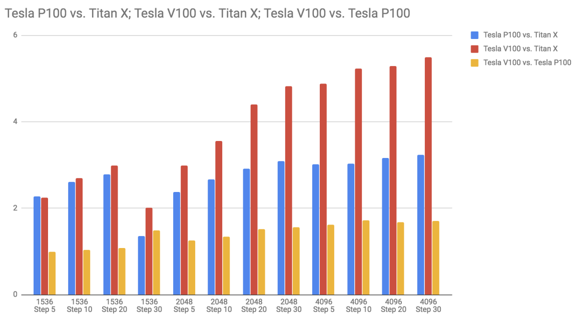

Figure 6: Average speedup factor between GPUs for the computation of cudaIFT.

sla P100. Figure 7 depicts some examples of cudaIFT and IFT-SC results to demonstrate that the cost maps are identical and that there are virtually no differences in label assignment.

5 CONCLUSIONS

We have presented cudaIFT, a parallel version of the IFT-SC operator using Nvidia's CUDA language. cudaIFT is able to achieve up to roughly 180x speedup over the sequential linear-time CPU implementation of the IFT-SC operator, with perfectly consistent calculations of the optimum-path forest. We have evaluated our algorithm to estimate Waterpixels in 2D slices of 3D CT images of rock samples, and demonstrated consistently superior results across three different machine configurations. In the future, we aim to extend the method to work with 3D images and non-planar image-graphs, and to automatically determine the number of iterations M. Our goal is to ensure that large 3D and 4D images being acquired using different imaging modalities with increasingly faster speeds be segmented with the lowest possible processing time. We also intend to extend cudaIFT to handle other types of connectivity functions and evaluate it with more general seed sets.

ACKNOWLEDGEMENTS

The authors would like to thank CNPEM, CAPES, and the Serrapilheira Institute (Serra-1708-16161) for the financial support. The authors would like to thank Dr. Tannaz Pak for the images used in this paper, as well as Prof. Alexandre X. Falcão, Prof. Tiago J. Carvalho, and Prof. Hélio Pedrini for the provided insights and discussions.

REFERENCES

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. (2012). SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2274–2282.

Alexandre, E. B., Chowdhury, A. S., Falcão, A. X., and Miranda, P. A. V. (2015). IFT-SLIC: A general framework for superpixel generation based on simple linear iterative clustering and image foresting transform. In *SIBGRAPI*, Salvador, Brazil.

Ciesielski, K. C., Falcão, A. X., and Miranda, P. A. V. (2018). Path-value functions for which dijkstra's algorithm returns optimal mapping. *J. Math. Imaging Vis.* Accepted.

Costa, G., Archilha, N. L., O'Dowd, F., and Vasconcelos, G. (2017). Automation Solutions and Prototypes for the X-Ray Tomography Beamline of Sirius, the New Brazilian Synchrotron Light Source. In *ICALEPCS*, pages 923–927, Barcelona, Spain. JACoW.

Dial, R. B. (1969). Algorithm 360: Shortest-path forest with topological ordering. *Commun. ACM*, 12(11):632–633.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Falcão, A. X. and Bergo, F. P. G. (2004). Interactive Volume Segmentation With Differential Image Foresting Transforms. *IEEE Trans. Med. Imag.*, 23(9):1100–1108.

Falcão, A. X., Stolfi, J., and Lotufo, R. A. (2004). The Image Foresting Transform: Theory, Algorithms, and Applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(1):19–29.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient Graph-Based Image Segmentation. *Int. J. Comput. Vis.*, 59(2):167–181.

Lotufo, R. A., Falcão, A. X., and Zampirolli, F. A. (2002). IFT-Watershed from gray-scale marker. In *SIBGRAPI*, pages 146–152.

Machairas, V., Faessel, M., Cárdenas-Peña, D., Chabardes, T., Walter, T., and Decencière, E. (2015). Waterpixels. *IEEE Trans. Image Process.*, 24(11):3707–3716.

Miranda, P. and Mansilla, L. (2013). Oriented image foresting transform segmentation by seed competition. *IEEE Trans. Image Process.*, PP(99):1–1.

Miranda, P. A. V., Falcão, A. X., and Spina, T. V. (2012). Riverbed: A novel user-steered image segmentation method based on optimum boundary tracking. *IEEE Trans. Image Process.*, 21(6):3042–3052.

Miranda, P. A. V. and Mansilla, L. A. C. (2014). Oriented Image Foresting Transform Segmentation by Seed Competition. *IEEE Trans. Image Process.*, 23(1):389–398.

Neubert, P. and Protzel, P. (2014). Compact watershed and preemptive slic: On improving trade-offs of superpixel segmentation algorithms. In *ICPR*, pages 996–1001, Stockholm, Sweden.

Papa, J. P., Falcão, A. X., de Albuquerque, V. H. C., and Tavares, J. M. R. (2012). Efficient supervised optimum-path forest classification for large datasets. *Pattern Recognition*, 45(1):512 – 520.

Rauber, P. E., Spina, T. V., Rezende, P., and Falcão, A. X. (2013). Interactive segmentation by image foresting transform on superpixel graphs. In *SIBGRAPI*, Arequipa, Peru.

Rocha, L. M., Cappabianco, F. A. M., and Falcão, A. X. (2009). Data clustering as an optimum-path forest problem with applications in image analysis. *Int J. Imaging Syst. Technol.*, 19(2):50–68.

Roerdink, J. and Meijster, A. (2000). The watershed transform: Definitions, algorithms and parallelization strategies. *Fund. inform.*, 41:187–228.

Stegmaier, J., Amat, F., Lemon, W. B., McDole, K., Wan, Y., Teodoro, G., Mikut, R., and Keller, P. J. (2016). Real-time three-dimensional cell segmentation in large-scale microscopy data of developing embryos. *Dev. Cell*, 36(2):225–240.

Stegmaier, J., Spina, T. V., Falcão, A. X., Bartschat, A., Mikut, R., Meyerowitz, E., and Cunha, A. (2018). Cell segmentation in 3D microscopy images using supervoxel merge-forests with CNN-based hypothesis selection. In *ISBI*, Washington, USA.

Vargas-Muñoz, J. E., Chowdhury, A. S., Barreto-Alexandre, E., Galvão, F. L., Miranda, P. A. V., and Falcão, A. X. (2018). An iterative spanning forest framework for superpixel segmentation. *CoRR*, abs/1801.10041.

Wang, L., Li, D., and Huang, S. (2016). An improved parallel fuzzy connected image segmentation method based on cuda. *BioMed. Eng. OnLine*, 15(1):56.

Zhuge, Y., Ciesielski, K. C., Udupa, J. K., and Miller, R. W. (2012). GPU-based relative fuzzy connectedness image segmentation. *Med. Phys.*, 40(1):011903.