The Journal of Supercomputing (2019) 75:1565–1579
https://doi.org/10.1007/s11227-018-2690-1

Extended attribute profiles on GPU applied to hyperspectral image classification

Pedro G. Bascoy[1] · Pablo Quesada-Barriuso[1] · Dora B. Heras[1] · Francisco Argüello[1] · Begüm Demir[2] · Lorenzo Bruzzone[3]

Published online: 19 November 2018

Abstract
Extended profiles are an important technique for modelling the spatial information of hyperspectral images at different levels of detail. They are used extensively as a preprocessing stage, especially in classification schemes. In particular, attribute profiles, based on the application of morphological attribute filters to the connected components of the image, have been shown to provide very good results. In this paper we present a parallel implementation of the attribute profiles in CUDA for multispectral and hyperspectral imagery considering the attributes area and standard deviation. The profile computation is based on the max-tree approach but without building the tree itself. Instead, a matrix-based data structure is used along with a recursive flooding (component merging) and filter process. Additionally, a previous feature extraction stage based on wavelets is applied to the hyperspectral image in order to extract the most valuable spectral information, reducing the size of the resulting profile. This scheme efficiently exploits the thousands of available threads on the GPU, obtaining a considerable reduction in execution time as compared to the OpenMP CPU implementation.

Keywords Remote sensing · Hyperspectral · Attribute profiles · Supervised classification · Real-time · GPU

1 Introduction

Classification plays a fundamental role in the analysis of remote sensing images for a number of applications. Researchers have proved that techniques which exploit not only spectral but also spatial information of images significantly improve the classification results using any spectral-based classifier, producing the so-called spectral–spatial classifiers. In particular, mathematical morphology-based classifiers have been very used during the last years [6].

Extended author information available on the last page of the article

Profiles build granulometries in order to get rid of irrelevant spatial details and simultaneously preserve the relevant characteristics of regions. They provide a multi-level characterization of the images using sequential morphological filters. Attribute profiles (APs) [11] consider morphological attribute filters based on one or more attributes such as area, standard deviation, diagonal of the bounding box and moment of inertia. The concept of AP was generalized and applied to multi- or hyperspectral images, known as extended APs (EAPs).

Throughout the literature several works have contributed with efficient algorithms to achieve a fast computation of attribute profiles. L. Vincent proposed in [18] an efficient algorithm based on sequential scanning. The general idea is to successively consider all the regional maxima of the image and process them progressively until its area becomes larger than a threshold λ. The algorithm is queue-based and performs in-place transformations, i.e. it modifies the original image proceeding in a sequential manner after processing each maximum.

A max-tree representation of the image for attribute filtering is proposed by Salembier et al. [16]. The proposal consists in creating a tree of connected components by threshold decomposition. So, at each level h, the components contain only those pixels which present a grey level h. The attribute filters remove the connected components of each image that do not meet a threshold criteria. An efficient algorithm based on the max-tree but without building the data structure is proposed for area opening in [4]. Another approach [19] is based on the union-find method originally proposed by Tarjan [17] and adapted to attribute openings. This approach allows the simultaneous processing of peak components, which are created and merged as needed. A comparison of the approaches for attribute filtering based on pixel-queues, max-trees and union-find can be found in [9].

Parallel algorithms on CPU for attribute filtering are focused on the parallelization of the max-tree data structure [2,8,10,20] on shared-memory architectures. In [20], the image is partitioned into multiple slices and the max-tree of each slice is computed in parallel on different processors using any sequential max-tree algorithm.

In [8], the image is partitioned into one-dimensional rows and one-dimensional trees are processed and merged in parallel. The main bottleneck is represented by the tree construction, consuming about 80% of the application execution time [8]. A comparison of shared-memory parallel implementations of several max-tree algorithms is presented in [2]. In [10], a shared-memory parallel algorithm combining the strategies of flooding and merging max-tree algorithms is proposed and applied to 2D and 3D images. However, even though GPUs provide a cost-efficient solution to carry out on-board real-time processing of remote sensing hyperspectral data [7,13], no GPU proposals can be found in the literature for the attribute profiles computation.

In this paper the first GPU algorithm for computing extended attribute profiles over hyperspectral images based on the max-tree representation is proposed. The proposed scheme does not build the max-tree structure, reducing the memory requirements of the algorithm. On the contrary, following the idea proposed in [4], the algorithm relies on the combination of a flooding and filtering process over a regular matrix structure produced by a connected component labeling algorithm.

A preliminary version of this work considering multispectral images was presented in [15]. In the present paper, the scheme is generalized to consider hyperspectral

images, which implies the need to include a feature reduction method and the extended construction of the attributes. In addition, a detailed study of the GPU implementation and an analysis of several configurations of the algorithm are included in this paper.

2 Attribute profiles

In this section, the concepts and notations from mathematical morphology that are required for understanding morphological attribute profiles are introduced. Then, the attribute opening and attribute filtering are reviewed. Finally, the construction of the extended attribute profile (EAP) is described.

2.1 Definitions

Attribute filters (AFs), also known as attribute openings, are connected operators that process an image according to a given criterion such as the area or the standard deviation of the pixel intensity, removing connected components that do not satisfy that criterion.

A connected component (CC) is a subgraph of an image f in which pixels are connected to each other by paths. AFs keep or merge the CCs based on a logical criterion T if a given attribute is greater/lower than an arbitrary reference, such as $T_{\lambda}^{\mathtt{att}}(\mathrm{CC}) = \mathtt{att}(\mathrm{CC}) > \lambda$, where $\mathtt{att}$ is an attribute, and λ is an arbitrary reference value [5].

The binary-connected opening Γ_x of an image f at a pixel x preserves only the connected component X which contains x. The binary trivial opening Γ_T uses an increasing criterion T to filter X. If the criterion is satisfied, the connected component is preserved, otherwise it is removed according to

$$\Gamma_T(X) = \begin{cases} X & \text{if } \ T(X) = \text{true}, \\ \emptyset & \text{otherwise.} \end{cases} \tag{1}$$

The *binary attribute opening* Γ^T of a binary image f is defined as:

$$\Gamma^T(f) = \bigcup_{x \in f} \Gamma_T[\Gamma_x(f)]. \tag{2}$$

The attribute opening is the union of all the trivial openings $\Gamma_T(f)$ which meet the increasing criterion T. The *greyscale attribute opening* γ^T is given by the maximum grey level of the results of the filtering for each pixel and can be mathematically presented as [11]:

$$\gamma^T(f)(x) = \max\left\{k : x \in \Gamma^T(f_k)\right\}. \tag{3}$$

The attribute filtering can be applied to an image in a sequence of increasing criteria, that is, with progressively higher threshold values, building the AP. Considering a

family of increasing criteria $T = \{T_\lambda : \lambda = 0, \ldots, l\}$, with λ a set of scalar values used as reference in the filtering procedure, and being T_0 true for all the CCs of an image f, the AP is composed of an attribute opening profile Π_{γ^T} and an attribute closing profile Π_{ϕ^T}, which are defined as follows:

$$\Pi_{\gamma^T}(f) = \left\{ \Pi_{\gamma^{T_\lambda}} : \Pi_{\gamma^{T_\lambda}} = \gamma^{T_\lambda}(f), \forall \lambda \in [0, \ldots, l] \right\}, \tag{4}$$

$$\Pi_{\phi^T}(f) = \left\{ \Pi_{\phi^{T_\lambda}} : \Pi_{\phi^{T_\lambda}} = \phi^{T_\lambda}(f), \forall \lambda \in [0, \ldots, l] \right\}, \tag{5}$$

with γ^{T_λ} and ϕ^{T_λ} denoting the morphological attribute opening and closing, respectively. The AP is defined as the concatenation of Eqs. (4) and (5), including the image itself:

$$\mathrm{AP}(f) = \left\{ \Pi_{\phi^T}(f), f, \Pi_{\gamma^T}(f) \right\}. \tag{6}$$

The AP can be extended to multidimensional images by applying Eq. (6) to each band of the image. This approach is known as EAP, and it is defined as follows:

$$\mathrm{EAP}(f) = \{\mathrm{AP}(f)_1, \mathrm{AP}(f)_2, \ldots, \mathrm{AP}(f)_r\}, \tag{7}$$

being r the number of components (bands) of the multidimensional image f.

3 Attribute profiles on GPU

This section describes the implementation details of the attribute profiles on GPU based on a flooding and filtering process. In this sense, the proposal does not build the max-tree and avoids the use of priority queues, such as the direct approaches of [1,18]. In addition, we used an adapted version of the connected component labeling (CCL) algorithm on GPU proposed in [12] for greyscale images. The CCL is used for creating the initial set of connected components used for applying the attribute filters.

Algorithm 1 shows the pseudocode for computing the EAP on GPU proposed in this paper, which requires a hyperspectral image f and a set of scalar values T as input parameters. First, a feature extraction (FE) method based on wavelets [14] is carried out over the input image f, building a feature-reduced image f' which contains as much relevant spectral information as possible concentrated in a reduced number of bands. Then, the algorithm iterates over the set of bands of the feature-reduced image f' to build the attribute opening profile Π_{γ^T} (line 5) and the attribute closing profile Π_{ϕ^T} (line 7) for each band w. Those profiles are finally concatenated with w (line 9), as defined in Sect. 2 by Eq. (6), to build the AP. Iteratively, each AP is finally included in the EAP set conforming the extended attribute profile.

The implementation used for Π_{γ^T} is also used for computing Π_{ϕ^T}, simply complementing the input image f and then complementing the result, lines 6 and 8, respectively.

The pseudocode of the function *AttributeOpeningProfile* in Algorithm 1 is shown in detail in Algorithm 2. The kernels, i.e. the pieces of code suitable to run on a GPU,

Algorithm 1 Extended Attribute Profile on GPU.

Require: Hyperspectral image f and a set of scalar values $T = \{T_\lambda : \lambda = 0, \ldots, l\}$
Ensure: Extended Attribute Profile on GPU: EAP(f)
1: **procedure** EXTENDEDATTRIBUTEPROFILE(f, T)
2: $f' \leftarrow$ FEATUREEXTRACTION(f)
3: EAP $\leftarrow \emptyset$
4: **for each** band w in f' **do**
5: $\Pi_{\gamma^T} \leftarrow$ ATTRIBUTEOPENINGPROFILE(w, T)
6: $w^c = 255 - w$ ▷ Complement the input image
7: $\Pi_{\phi^T} \leftarrow$ ATTRIBUTEOPENINGPROFILE(w^c, T)
8: $\Pi_{\phi^T} = 255 - \Pi_{\phi^T}$ ▷ Complement the result
9: EAP = EAP $\cup \{\Pi_{\phi^T}, w, \Pi_{\gamma^T}\}$
10: **end for**
11: **end procedure**

Algorithm 2 Attribute Opening Profile on GPU.

Require: Input band w and a set of scalar values $T = \{T_\lambda : \lambda = 0, \ldots, l\}$
Ensure: Attribute opening profile on GPU: $\Pi_{\gamma^T}(w) = \gamma^{T_\lambda}(w), \forall \lambda \in [0, \ldots, l]$
1: **procedure** ATTRIBUTEOPENINGPROFILE(f, T)
2: $\Pi_{\gamma^T} \leftarrow w$; $T_\lambda \leftarrow T$ ▷ Initial transfer CPU–GPU
3: H $\leftarrow$ <Histogram>(Π_{γ^T}) ▷ GM
4: i = H.length() − 1; h = H[i]; root = H[0] ▷ Grey levels
5: CCL $\leftarrow$ <ComponentLabeling>(Π_{γ^T}, root) ▷ GM,SM
6: att $\leftarrow$ <InitializeAtt>(CCL) ▷ GM
7: **while** $h >$root **do**
8: att $\leftarrow$ <UpdateAtt>(CCL) ▷ GM
9: $\Pi_{\gamma^T} \leftarrow$ <AttProfile>(Π_{γ^T}, CCL, att, h, T_λ) ▷ Tex,Con,GM
10: **do**
11: <MergeRegions>(Π_{γ^T}, CCL, h, $h-1$) ▷ Tex,GM
12: <FindRoots>(CCL) ▷ GM
13: **while** (oldRoot $\neq$ newRoot)
14: $h \leftarrow$ H[i]; $i = i - 1$
15: **end while**
16: **end procedure**

are placed between <> symbols in the pseudocode. The max-tree traversal procedure done by a regular algorithm is emulated in our kernels *MergeRegions* and *FindRoots*, while the attribute profile is built in kernels *InitializeAtt*, *UpdateAtt* and *AttProfile*. The pseudocode also includes the Tex, Con, GM and SM acronyms to indicate the use of the texture, constant, global and shared memory spaces on GPU, respectively.

First, the image is copied to the global and to the texture memory, and the scalar values T_λ are copied to the constant memory of the GPU. The first three kernels (*Histogram*, *ComponentLabeling* and *InitializeAtt*) are executed only once. The histogram of the input image is computed in global memory (line 3) for extracting its grey levels. The minimum and maximum values would correspond respectively the root and the leaves if the max-tree would have been constructed. We will use this max-tree analogy in the rest of the paper although such structure is not explicitly built.

In line 5, the image is labelled using the CCL algorithm proposed in [12], adapting it by using the root grey value (the histogram minimum) as the background and verifying

that $f(p) = f(q)$ before path compression, instead of assuming that all foreground pixels have the same value. The result is a `CCL` array that has the same dimension as the input data. The attribute of each connected component (line 6 in Algorithm 2) is computed in an array `att`, also of the same size as the image, where the root of the component is used to locate the value of the attribute.

As can be seen in lines 7–15 in Algorithm 2, the `CCL` array traversal is performed. The process begins from the maximum grey level (leaves) and continues to the minimum value of the `CCL` array (root), merging the components at the current grey level h (lines 10–13), and computing the attribute opening at that level (lines 8 and 9). The different kernels used in Algorithm 2 for building the attribute profile are described in the following sections.

3.1 Kernel InitializeAtt

After labelling the connected components, the array to update the attribute of the components (`att`) is initialized in this kernel. Each thread, identified by p in the kernel, adds one to the array `att` (line 3), initializing the area of each component. The atomic operation is necessary to avoid race conditions because multiple threads will update the same global memory location.

Kernel InitializeAtt Initialize the attribute of each connected component.

Require: Connected components `CCL` and attributes `att` in global memory
1: $root \leftarrow$ `CCL`$[p]$
2: **if** ($root \neq background$) **then**
3: atomicAdd(`att`$[root]$, 1)
4: **end if**

3.2 Kernel UpdateAtt

This kernel is executed by each thread on the GPU, updating the attribute of the components that have been recently merged. This approach assumes an increasing criterion in the attribute of area and not increasing for the attribute of standard deviation. All threads processing the same component will atomically increase the attribute in global memory (line 3 in the pseudocode of this kernel), except the thread processing the root of the component, identified by the global position (gid) of that thread within the image.

3.3 Kernel AttProfile

In order to obtain the attribute opening profile, this kernel uses all the scalar values λ in the same kernel call, computing $\Pi_{\gamma^T}(w) = \gamma^{T_\lambda}(w), \forall \lambda \in T = \{T_\lambda : [0, \ldots, l]\}$. The *offset* at line 5 indicates the global memory location on GPU for each attribute opening γ^{T_λ}. We apply the *min* rule for filtering the connected components (CCs) and

Kernel UpdateAtt Update attributes of connected components recently merged.

Require: Connected components `CCL` and attributes `att` in global memory
1: $root \leftarrow \texttt{CCL}[p]$
2: **if** ($\texttt{att}[p] > 0$ and $gid \neq root$) **then**
3: atomicAdd($\texttt{att}[root]$, $\texttt{att}[p]$)
4: $\texttt{att}[p] \leftarrow 0$
5: **end if**

creating the profiles. Each thread filters one pixel of the component at the current level h (line 5) for each λ value used in the attribute profile. Filtering the CCs using one λ at level h corresponds with the attribute opening, as indicated by Eq. (3).

Kernel AttProfile Build the attribute opening profile.

Require: Connected components `CCL`, attributes `att` and current grey level h in global memory. Input image f in texture memory, and T_λ in constant memory.
1: $root \leftarrow \texttt{CCL}[p]$
2: $v \leftarrow \texttt{att}[root]$
3: **for** each $\lambda \in T_\lambda$ **do**
4: **if** ($0 < v < \lambda$ and $\texttt{f}[p] \geq h$) **then**
5: $\Pi_\gamma T[p + offset] \leftarrow h - 1$
6: **end if**
7: **end for**

3.4 Kernel MergeRegions

This pseudocode shows what each thread looks like for a different root in its neighbourhood (lines 2–6) to merge the CCs at a new level h. This operation is performed in texture memory, so we take advantage of the spatial location and the texture cache memory of the GPU. It is unknown if the neighbouring components must be merged until line 7, where the root of the new connected component is updated by an atomic operation.

Kernel MergeRegions Merge the border of connected components at levels h.

Require: Connected components `CCL` and current grey levels h in global memory. Input image f in texture memory.
1: $root \leftarrow \texttt{CCL}[p]$
2: **for** each q neighbor of p **do**
3: **if** ($\texttt{f}[q] \geq h$ and $\texttt{f}[p] \geq h$) **then**
4: $root \leftarrow \min(root, \texttt{CCL}[q])$
5: **end if**
6: **end for**
7: **if** ($root < \texttt{CCL}[p]$) **then**
8: atomicMin($\texttt{CCL}[root]$, $root$)
9: **end if**

3.5 Kernel FindRoots

Kernel FindRoots Find the root of the regions that have been merged.

Require: Connected components CCL and a *flag* in global memory.
1: $root \leftarrow p$
2: $oldRoot \leftarrow \texttt{CCL}[root]$
3: **while** ($\texttt{CCL}[root] \neq root$) **do**
4: $root \leftarrow \texttt{CCL}[root]$
5: **end while**
6: **if** ($oldRoot \neq root$) **then**
7: AtomicExchange($\texttt{CCL}[p], root$)
8: AtomicOr($flag$, 1) ▷ Update a flag indicating a new call to *MergeRegions* is required.
9: **end if**

After merging the borders of adjacent components, they are finally updated by finding the new root for each pixel, as shown in the pseudocode of this kernel. To ensure that the regions are merged and the components are represented by compressed paths, this kernel iterates several times within the GPU (lines 3–5) until the pixel (each thread processes one pixel) gets the new root of the component. Finally, the pixels of the component are updated atomically in global memory only if they have a new root, thus saving write transactions into it. Because the computation is divided by blocks, a *flag* in global memory is updated if the merging of components has not finished, indicating to the CPU that a new call to the *MergeRegions* and *FindRoots* kernels is required.

4 Experimental results

This section presents the results for building the extended attribute profile on GPU using two data sets: a well-known hyperspectral remote sensing image to validate the constructed profile through classification accuracies and a huge data set to push the algorithm beyond the GPU memory limits.

The following lines provide a description of the experimental conditions as well as a discussion of the main results.

4.1 Experimental setup and data sets

The experiments have been executed on a personal computer consisting of a first generation 2.80 GHz Intel Core i7 microprocessor, 16 GB of RAM and a Pascal Nvidia GTX 1070 GPU with 8 GB of global memory, 15 SMs and 128 CUDA cores per SM. The EAPs on GPU are implemented using CUDA C++ and compiled with `nvcc` version 9.1.85. The execution times on GPU include memory allocation, data transformations and data transfers from the host to the device and vice versa.

With reference to the data sets, a commonly used image in the literature, the Pavia Centre image, and an image capturing the surroundings of the city of Santiago de Compostela in Spain have been selected for the experiments. The former was obtained by the ROSIS-03 sensor over the city centre area of Pavia,[1] Italy. The dimensions of

[1] http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

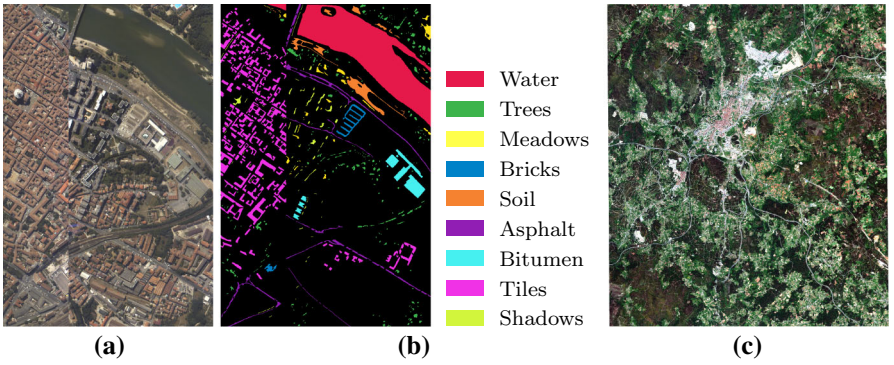

Fig. 1 **a** False colour composition of the Pavia Centre data set. **b** Shows the Pavia's reference data used for assessing the classification accuracy. **c** Corresponds to a colour composition of the Santiago area data set (colour figure online)

the image are 1096×715 with 102 spectral bands. A colour composition of the data set is shown in Fig. 1a, along with its reference data (Fig. 1b), where each colour represents a specific land-cover class.

The data set capturing the Santiago area, Spain, was acquired by the WV110 sensor. Its spatial dimensions are $10,748 \times 12,288$ and 8 spectral bands. The image is available through ESA's World-View 2 data access web-page.[2] The colour composition can be seen in Fig. 1c, and there are no reference data associated with it. Due its large size, it has been selected in order to push the proposed algorithm beyond the limits of the GPU memory, being impossible to fit both the image and the profile within the global memory.

It is worth noting that a feature extraction (FE) method based on wavelets is included in order reduce the number of bands of the Pavia Centre hyperspectral image, as shown in Algorithm 1. The number of bands that remain after the feature extraction (eight) allows building a small profile while keeping most of the spectral information. In the case of the Santiago area data set no FE stage is required. The profile is built over all the available bands of the image. At the same time, the input data for both data sets must be scaled between 0 and 255 for the correct operation of the algorithm.

The attribute profiles built for both data sets are based on the area attribute, being T the following set of scalars: 36, 144, 324, 576, 900, 1296, 1764.

The SVM supervised classification is carried out by the library LIBSVM [3]. Its input is scaled between -1 and 1 for improving classification results. The Gaussian radial basis function (RBF) has been used as the activation function with parameters (C, γ) tuned by fivefold cross validation as: $C = 256$, $\gamma = 1$.

4.2 Performance results

Table 1 shows the execution times for the different stages required to compute the EAP over the Pavia Centre and Santiago area data sets using the area attribute. The column

[2] https://earth.esa.int/web/guest/-/worldview-2-european-cities-dataset.

Table 1 Disaggregated execution times on the computation of the EAP on GPU using the area attribute

T			{36, 144, 324, 576, 900, 1296, 1764}	
#Bands			8	
Profile size			120	
Data set	Pavia Centre		Santiago area	
Connectivity	4		4	
Attributte	Area		Area	
OA accuracy	99.83%		–	
SPEC	EAP	AP per band	EAP	AP per band
Feature extraction	918.74	114.84	–	–
Cuda malloc	7.01	0.88	32.64	4.08
Transfer H2D	3.11	0.39	452.56	56.57
Histogram	12.43	1.55	1,039.73	129.97
CCL	14.92	1.87	1,403.09	175.39
Attribute opening profile	678.44	84.81	84,660.13	10,582.52
Image complement	2.25	0.28	174.82	21.85
Attribute closing profile	637.47	79.68	88,412.92	11,051.62
Transfer D2H	48.50	6.06	8,691.45	1,086.63
Cuda free	8.25	1.03	191.47	23.93
Profile total	1412.37	176.55	185,058.80	23,132.35
Execution total	**2331.11**	**291.39**	**185,058.80**	**23,132.35**
Memory (GM + TM) per band	17.19 MB + 0.75 MB		2896.92 MB + 125.95 MB	
Kernel (block, grid)	32 × 8, 23 × 137		32 × 8, 336 × 1536	

Times are displayed in milliseconds
The line highlighted in bold shows the total execution times

labelled as "EAP" displays the times for the computation of the whole EAP. On the other hand, times belonging to the "AP per band" column correspond to the average of the computation of each AP (one per band on the feature-reduced image by wavelets in the case of the Pavia Centre data set). Note that the Santiago area data set (displayed on the right side of the table) does not need a FE stage due to its reduced number of bands. The row named as "Profile Total" corresponds to the result of summing the execution time for all the other rows above it except "Feature Extraction", which is included in the "Execution Total" one. In addition, the memory requirements for computing every single AP are shown in terms of MB for the global (GM) and texture (TM) memories. The global memory utilization corresponds to $WH(2l+1) + 2WH \times 4$ and to $WH(2l+1) + 2WH \times 4 + 3WH \times 4$ bytes for the area and the standard deviation profiles, respectively. W and H stand, respectively, for the width and height of the input image, while l is the cardinal number of the set T. On the other hand, the TM utilization corresponds to the size of the whole band in bytes: WH.

The last row of the table indicates the launching configuration of the kernels, being the first parameter the block size and the second one the grid size, calculated as $\lceil \frac{W}{32} \rceil \times \lceil \frac{H}{8} \rceil$.

Table 2 Pavia Centre data set: comparison between OpenMP CPU and CUDA GPU execution times for the area attribute profile

#Extracted bands	Profile size	CPU			GPU			Speedup
		FE	EAP	Total	FE	EAP	Total	
4	60	5.62	31.27	36.89	1.33	0.92	2.25	16.40×
8	120	5.48	66.95	125.48	0.92	1.41	2.33	28.73×
16	240	5.10	129.07	245.10	1.02	2.82	3.84	33.61×
32	480	4.42	258.03	484.42	1.15	5.65	6.80	37.95×
64	960	3.02	515.72	963.02	1.39	11.09	12.47	41.36×

Times are displayed in seconds

Table 3 Santiago area data set: comparison between OpenMP CPU and CUDA GPU execution times and memory requirements for the area attribute profile

#Bands	GPU			CPU		Speedup
	Execution time (s)	Memory (MB)		Execution time (s)	Memory (MB)	
		GM	TM			
8	185.06	2896.92	125.95	8899.52	64,488.00	48.09×

It can be observed that the most time-consuming operations correspond to the feature extraction and the attribute opening profile. Regarding the latter, the *InitializeAtt*, *UpdateAtt*, *AttProfile*, *MergeRegions* and *FindRoots* kernels described in the previous section are executed inside the attribute opening profile function (See Algorithm 2). In fact, both opening and closing attribute profile calculations (they are actually the same function as indicated in Algorithm 1) represent 93.17% of the computation time of the profile, which includes the transformation and data transfer between host and device. Although the results for connectivity 8 are not displayed in the table, it is worth noting that the computational times increase 15% on average.

In Table 2, a comparison between a CPU and the proposed GPU implementations is presented. The CPU version of the algorithm matches the steps of the GPU version, i.e. creating the EAP by the usage of the proposed flooding and filtering procedure. In addition, the CPU code was parallelised using OpenMP directives. The performance results are expressed in terms of execution time (in seconds) and GPU speedup over the OpenMP code. The results correspond to the construction of the EAP using the Pavia Centre data set considering the area attribute, a connectivity of four and the same thresholds T shown in Table 1. In addition, different configurations of the number of bands extracted by wavelets are considered, so that the bigger the size of the feature-reduced image, the bigger the profile size and the computational time required. The speedups are calculated using the total computation time (FE+EAP).

It can be noticed that the GPU implementation is more efficient for big profiles, producing high speedups. The biggest speedup, 41.36×, is achieved when the profile built consists of 960 bands.

A more challenging experiment is shown in Table 3, where the profile is built for the Santiago area image. Its size makes it impossible to fit both the whole profile and

Table 4 Pavia Centre data set

Kernel	#	Achieved occupancy (%)	Theoretical occupancy (%)	Threads per block	Registers per thread	SM per block (KiB)	Limiter
FeatureExtraction	1	87.25	100.00	256	20	0.43	None
Histogram	16	40.54	63.64	1024	72	4.00	Registers
ComponentLabeling	16	89.11	100.00	256	23	1.59	None
InitializeAtt	16	80.65	100.00	256	8	0.0	None
UpdateAtt	3066	82.63	100.00	256	8	0.0	None
AttProfile	3082	89.59	100.00	256	31	0.0	None
MergeRegions	7900	89.60	100.00	256	17	0.0	None
FindRoots	7900	83.63	100.00	256	8	0.0	None
Image complement	112	75.64	100.00	256	8	0.0	None

Obtained occupancy for the Pavia Centre data set using connectivity 4 and the area attribute

the image in the GPU global memory (around 23 GB of global memory would be required). To address the problem, a band-wise strategy separately computes the AP of each band on the GPU, moving it to the host once its computation is completed. As the number of bands of the image is reduced (eight), no FE is needed in the processing pipeline, being the profile computed directly from the bands of the data set. Therefore, the execution times shown in the table correspond just to the EAP computation. The global memory (GM) requirements show the maximum amount of memory needed for computing the whole EAP, which corresponds to the amount of memory of a single AP (shown at the beginning of this section) in the case of the developed GPU algorithm. The GPU speedup reaches 48× as compared to the CPU version. Note that the computation of the profile is independently done for each band, so the number of the spectral bands of the input image do not change the GM and TM requirements.

The GPU hardware occupancy of the proposed functions has been analysed using Nvidia `nvprof` tool, which provides metrics at kernel level (Table 4). We have collected the measures using the Pavia Centre data set building an EAP using the area attribute and a connectivity of four. The block sizes correspond to 32×8 for all the kernels presented in Table 4 except for the `Histogram` kernel, which uses the *thrust* C++ STL-style library. Some procedures are not defined at a kernel level but as a set of kernels. In those cases kernel occupancies are estimated by weighting each kernel occupancy by its number of invocations.

Occupancies do not reach the maximum in those kernels whose number of blocks available is not enough in order to hide the memory latency. Accordingly, this behaviour fits for every kernel except for the histogram computation due to the use of the *thrust* library (since it is not possible to configure the launching parameters of it).

5 Conclusions

In this paper a CUDA GPU implementation of the extended attribute profiles for remote sensing hyperspectral images is presented. The implementation is based on a flooding and filtering process similar to those proposed by the max-tree-based algorithms. Nevertheless, the proposal does not build the max-tree explicitly. In fact, the attribute filters are applied to a connected components structure, which holds the necessary information to iteratively build the attribute profile. A wavelet-based feature extraction stage is executed over the hyperspectral image previously to the extended attribute profile computation. Once the profile is completed, a classification stage carried out by SVM is performed assessing the correct behaviour of the developed algorithm. The attributes considered in the experiments are area and standard deviation, although the approach could be extended to others. The experiments have been carried out over Pavia Centre and a huge multispectral data set. Good speedups with a reduction from minutes to a couple of seconds as compared to the corresponding CPU implementation are obtained. Furthermore, a hardware occupancy study has been included to give insights of the correct execution behaviour.

Acknowledgements This work was supported in part by the Consellería de Cultura, Educación e Ordenación Universitaria [Grant numbers GRC2014/008 and ED431G/08] and Ministry of Education, Culture

and Sport, Government of Spain [Grant number TIN2016-76373-P]. Both are co-funded by the European Regional Development Fund (ERDF).

References

1. Breen EJ, Jones R (1996) Attribute openings, thinnings, and granulometries. Comput Vis Image Underst 64(3):377–389
2. Carlinet E, Géraud T (2013) A comparison of many max-tree computation algorithms. In: Hendriks CLL, Borgefors G, Strand R (eds) Mathematical morphology and its applications to signal and image processing. Springer, Berlin, pp 73–85
3. Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol 2:27:1–27:27
4. Darbon J, Akgül CB (2005) An efficient algorithm for attribute openings and closings. In: 2005 13th European Signal Processing Conference, pp 1–4
5. Ghamisi P, Mura MD, Benediktsson JA (2015) A survey on spectral–spatial classification techniques based on attribute profiles. IEEE Trans Geosci Remote Sens 53(5):2335–2353
6. Ghamisi P, Maggiori E, Li S, Souza R, Tarabalka Y, Moser G, De Giorgi A, Fang L, Chen Y, Chi M, Serpico S, Benediktsson J (2018) New frontiers in spectral–spatial hyperspectral image classification: the latest advances based on mathematical morphology, markov random fields, segmentation, sparse representation, and deep learning. IEEE Geosci Remote Sens Mag 6(3):10–43
7. Ma Y, Chen L, Liu P, Lu K (2016) Parallel programing templates for remote sensing image processing on GPU architectures: design and implementation. Computing 98(1):7–33
8. Matas P, Dokládalová E, Akil M, Grandpierre T, Najman L, Poupa M, Georgiev V (2008) Parallel algorithm for concurrent computation of connected component tree. Springer, Berlin, pp 230–241
9. Meijster A, Wilkinson MHF (2002) A comparison of algorithms for connected set openings and closings. IEEE Trans Pattern Anal Mach Intell 24(4):484–494
10. Moschini U, Meijster A, Wilkinson MHF (2018) A hybrid shared-memory parallel max-tree algorithm for extreme dynamic-range images. IEEE Trans Pattern Anal Mach Intell 40(3):513–526
11. Mura MD, Benediktsson JA, Waske B, Bruzzone L (2010) Morphological attribute profiles for the analysis of very high resolution images. IEEE Trans Geosci Remote Sens 48(10):3747–3762
12. Oliveira V, de Alencar Lotufo R (2010) A study on connected components labeling algorithms using GPUs. In: Proceedings: 23rd SIBGRAPI Conference on Graphics, Patterns and Images
13. Plaza A, Du Q, Chang Y, King RL (2011) High performance computing for hyperspectral remote sensing. IEEE J Sel Top Appl Earth Obs Remote Sens 4(3):528–544
14. Quesada-Barriuso P, Argüello F, Heras DB, Benediktsson JA (2015) Wavelet-based classification of hyperspectral images using extended morphological profiles on graphics processing units. IEEE J Select Top Appl Earth Obs Remote Sens 8(6):2962–2970
15. Quesada-Barriuso P, Heras DB, Argüello F, Demir B (2018) GPU computation of attribute profiles for remote sensing image classification. In: Aguiar JV (ed) Proceedings of the 18th International Conference on Computational and Mathematical Methods in Science and Engineering
16. Salembier P, Oliveras A, Garrido L (1998) Antiextensive connected operators for image and sequence processing. IEEE Trans Image Process 7(4):555–570
17. Tarjan RE (1975) Efficiency of a good but not linear set union algorithm. J ACM 22(2):215–225
18. Vincent L (1993) Grayscale area openings and closings, their efficient implementation and applications. In: Proceedings of EURASIP Workshop on Mathematical Morphology and its Applications to Signal Processing, pp 22–27
19. Wilkinson MHF, Roerdink JBTM (2000) Fast morphological attribute operations using Tarjan's union-find algorithm. Springer, Boston, pp 311–320
20. Wilkinson MHF, Gao H, Hesselink WH, Jonker JE, Meijster A (2008) Concurrent computation of attribute filters on shared memory parallel machines. IEEE Trans Pattern Anal Mach Intell 30(10):1800–1813

Affiliations

Pedro G. Bascoy[1] · Pablo Quesada-Barriuso[1] · Dora B. Heras[1] · Francisco Argüello[1] · Begüm Demir[2] · Lorenzo Bruzzone[3]

✉ Dora B. Heras
dora.blanco@usc.es

Pedro G. Bascoy
pedrogonzalez.bascoy@usc.es

Francisco Argüello
francisco.arguello@usc.es

Begüm Demir
demir@tu-berlin.de

Lorenzo Bruzzone
lorenzo.bruzzone@ing.unitn.it

1 Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

2 Faculty of Electrical Engineering and Computer Science, Technische Universität Berlin, Berlin, Germany

3 Department of Information Engineering and Computer Science, University of Trento, Trento, Italy