

GPU Framework for Change Detection in Multitemporal Hyperspectral Images

Javier López-Fandiño¹  · Dora B. Heras¹ ·
Francisco Argüello¹ · Mauro Dalla Mura²

Received: 15 September 2017 / Accepted: 6 December 2017 / Published online: 16 December 2017
© Springer Science+Business Media, LLC, part of Springer Nature 2017

Abstract Nowadays, it is increasingly common to detect land cover changes using remote sensing multispectral images captured at different time-frames over the same area. A large part of the available change detection (CD) methods focus on pixel-based operations. The use of spectral–spatial techniques helps to improve the accuracy results but also implies a significant increase in processing time. In this paper, a Graphic Processor Unit (GPU) framework to perform object-based CD in multitemporal remote sensing hyperspectral data is presented. It is based on Change Vector Analysis with the Spectral Angle Mapper distance and Otsu’s thresholding. Spatial information is taken into account by considering watershed segmentation. The GPU implementation achieves real-time execution and speedups of up to $46.5\times$ with respect to an OpenMP implementation.

Keywords Hyperspectral change detection · Segmentation · Spectral Angle Mapper · Change Vector Analysis · GPU · CUDA

✉ Javier López-Fandiño
javier.lopez.fandino@usc.es

Dora B. Heras
dora.blanco@usc.es

Francisco Argüello
francisco.arguello@usc.es

Mauro Dalla Mura
mauro.dalla-mura@gipsa-lab.grenoble-inp.fr

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

² GIPSA-lab, Institute of Engineering, CNRS, Grenoble INP, Université Grenoble Alpes, 38000 Grenoble, France

1 Introduction

Hyperspectral imaging is achieved by acquiring reflectance values for each pixel over a large number (e.g., hundreds to thousands) of narrow and contiguous spectral channels [3, 35], known as spectral bands. The further sensor technology advances, increasing the number of pixels and spectral bands, the greater the necessity for efficient techniques to deal with the huge amount of data collected by hyperspectral sensors. The detailed spectral information present in these images is exploited in many remote sensing applications that usually address problems like segmentation, classification, target detection, or change detection (CD), among others [12, 34]. The availability of multitemporal hyperspectral images (i.e., images corresponding to different time-frames over the same region) makes it possible to apply techniques to automatically recognize the significant changes that occurred in a region through time [4, 32].

One possible classification of CD techniques is based on the type of fusion of the multitemporal data (see Table 1): at feature level or at decision level [4]. In the former the information of both images is combined before performing the CD. The combination is usually based on algebraic operations over both images, such as image differencing, image ratioing, application of normalized difference vegetation index (NDVI) [1] or Change Vector Analysis (CVA) [20]. CVA is a technique that relies on the computation of distances and angles between pairs of pixels (each pixel is represented by a vector of values) corresponding to the same position in both images. The original CVA technique [33] is based on the computation of Euclidean distances. Several papers indicate that the Spectral Angle Mapper (SAM) is a better metric for comparing spectra in hyperspectral datasets [10]. Its independence of the number of spectral components allows comparing images of different spectral dimensionality. Besides, it is invariant to scale changes making it insensitive to variations in illumination [10, 18].

Other techniques, based on the fusion of images at the feature level, perform feature extraction over the image resulting from the fusion using, for example, Principal Component Analysis (PCA) [6, 21]. Calculating morphological profiles, such as attribute profiles, is another very well-known method in the remote sensing field in order to extract spatial features from the images [11]. Context based approaches such as Markov Random Fields (MRF) [14] or Expectation-Maximization-based Level Sets (EMLS)

Table 1 Classification of CD methods regarding the fusion level of the multitemporal data

Fusion at the feature level	
Pixel-based operations	Image differencing, ratioing, NDVI [1], CVA [20]
Feature extraction	PCA [6, 21]
Morphological profiles	Attribute profiles [11]
Context based approaches	MRF [14], EMLS [15]
Fusion at the decision level	
Post-classification	[17]
Direct multirate classification	[36]

[15] are applied in some papers. Regarding the thresholding process that is necessary in order to identify the changes, the most commonly used approaches are statistical methods such as Expectation-Maximization (EM) [5] but these techniques are computationally inefficient as they are based on slow iterative methods. Instead, Otsu's method [24] allows obtaining suitable thresholds with a much lower computational complexity since it relies on histograms that can be efficiently computed on a Graphic Processor Unit (GPU).

Regarding the approaches where the fusion of the information provided by both images is carried out at the decision level, we can quote some post-classification based techniques that compute the changes as the differences in the classification of the two images [17]. Other techniques perform a direct multivariate classification [36].

Remote sensing hyperspectral applications are computationally demanding and, therefore, good candidates to be projected onto high performance computing (HPC) infrastructures such as clusters or specialized hardware devices [9,26]. Some remote sensing techniques have been executed in Field Programmable Gate Arrays (FPGA) [37,38] or GPUs [19,26,28,31] to reduce their execution times. In this context, we consider real-time operation as that which allows us to fully process one image before the next one can be delivered by the sensor. This time depends on the sensor used. For example, for the AVIRIS sensor used in this paper, a track line of 512 pixel vectors is collected in 8.3 ms [27]. GPUs provide a cost-efficient solution to carry out onboard real-time processing of remote sensing hyperspectral data. For instance, [2] shows the advantages of performing hyperspectral unmixing on GPU, and [16] proposes a framework for atmospheric cloud filtering in remotely sensed images. Some papers in the literature explore the benefits of the GPU projection of CD techniques. [8] introduces a nearest neighbor hierarchical CD framework for crop monitoring where only the most computationally expensive step is executed in GPU. A CD technique based on image differencing and fuzzy clustering on AMD GPUs is presented in [39].

A framework to detect binary object-based changes in multitemporal hyperspectral datasets is presented in this paper. The proposed framework is designed to be efficient on both multi-core architectures and commodity GPUs. It considers fusion at the feature level while the spatial information is extracted through a segmentation process. The segmentation is carried out independently for each image before combining the features of the two images. CVA using the SAM distance [10] is used as an improvement to the CVA presented in [33]. The use of Otsu's method [24] to threshold the changes is proposed as an alternative to the EM approach. Finally, an iterative spatial regularization [19] is applied to reduce the presence of noise and unconnected pixels in the final output. The framework presented in this paper allows the efficient integration of these techniques as well as the real-time execution on GPU.

2 GPU Hyperspectral Framework for Change Detection

This section is devoted to the introduction of some Compute Unified Device Architecture (CUDA) fundamentals (Sect. 2.1) as well as to the CUDA GPU implementation (Sect. 2.2) of the framework.

2.1 CUDA GPU Programming Fundamentals

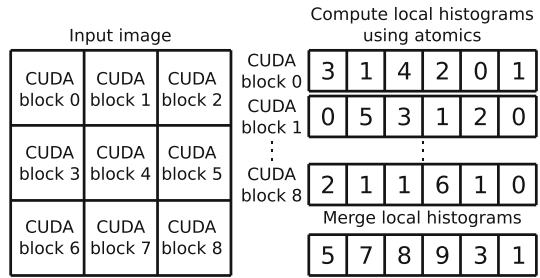
GPUs provide massively parallel processing capabilities thanks to their data parallel architecture. The CUDA platform allows NVIDIA GPUs to execute programs invoking parallel kernels that simultaneously execute in many parallel threads (one instance per thread, following a SIMD programming model) [23]. The threads are organized into blocks forming a grid that is mapped to a hierarchy of CUDA cores in the GPU. They can access data from multiple memory spaces: Private local memory, registers, a shared memory per block whose lifetime equals that of the block, and a shared global memory for all threads that is persistent across kernel launches. The NVIDIA PASCAL GPU architecture used in this paper [23] implements a two level cache hierarchy including a configurable L1 cache for each streaming multiprocessor (SM) and a unified L2 cache shared among all SMs. The architecture provides 96 KB of shared memory for each SM with a block limit of 48 KB.

The threads run in groups of 32 called warps. The occupancy of the GPU is defined as the number of active warps over the maximum number of warps supported per SM. Therefore, the performance will be better when the hardware occupancy is higher. The occupancy can be limited by three factors: the number of registers employed, the amount of shared memory required, and the maximum number of threads per block.

In order to improve performance, some optimization techniques have been considered:

1. *Minimize the use of global memory and the number of data transfers between host and device memories* The use of the global memory is minimized using in-place computation whenever possible, i.e., using the memory reserved for the inputs to store the outputs. All the computations are carried out in the GPU memory so that data transfers between host (CPU) and device (GPU) are reduced to copying the inputs and returning the outputs. It is also essential to minimize the data transfers between global and shared memory.
2. *Efficient use of libraries for common operations* The cuBLAS library [22] is used to efficiently compute matrix operations.
3. *Search for the best kernel configuration* The block size is tuned for each kernel to minimize the execution time. The factors involved in the block size selection are the number of registers and the shared memory used by the program. In our application the selection of the block size depends on the GPU model whereas the dimensions of the image are not relevant for the decision. Given that each SM can have a maximum number of active blocks, larger blocks are preferable. The selected block size is a multiple of 32 (the warp size) to avoid divergences among threads and thus maximize parallelism. A size of 512 or 1024 threads per block was selected for most of the kernels. This size is large enough to exploit the memory bandwidth of the device. Additionally, if a computation involves steps requiring different numbers of threads, it is split into consecutive kernels allowing the optimization of the resources and, therefore, maximizing efficiency.
4. *Avoid writing collisions* When several threads in a kernel need to atomically write in the same structure, it is more efficient to perform independent partial results per block and combine these results in a second kernel. This can be applied, for

Fig. 1 Example of histogram calculation using 9 blocks and 6 histogram levels



instance, in the calculation of a histogram, as shown in Fig. 1. In the first kernel, each CUDA block calculates the histogram of a spatial portion of the input image. Then, the second kernel merges all the partial results into the global histogram.

2.2 CUDA GPU Framework for Binary Change Detection

The details of the proposed spectral–spatial GPU framework for CD are provided in this section. Some pseudocodes are introduced where each process executed in GPU is placed between <> symbols and may involve one or more kernels. The pseudocodes also include the GM and SM acronyms to indicate kernels executed in global memory or shared memory, respectively.

The flowchart of the framework is displayed in Fig. 2 and its stages are detailed in Fig 3. The inputs to the framework are two co-registered hyperspectral images acquired on two different time-frames over the same area. Figure 4 shows the data structures employed along with the thread activity.

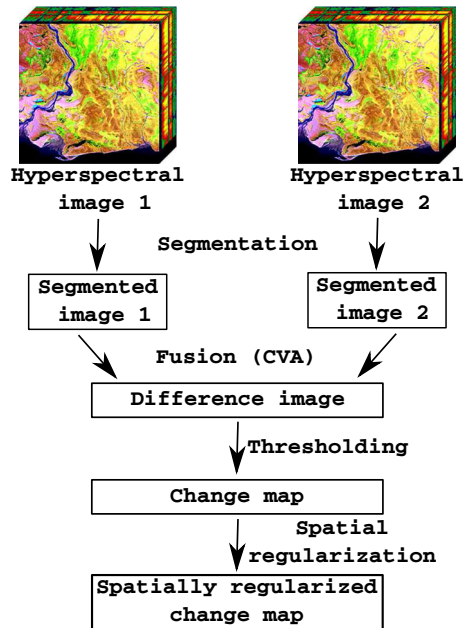
2.2.1 Segmentation Stage on GPU

As shown in Figs. 2, 3, and 4, each image is independently segmented using the watershed transform. The segmentation stage involves a three-step processing: Calculation of a gradient, creation of a watershed-based segmentation map, and averaging of the pixels belonging to the same regions.

The first step consists in the calculation of the magnitude of a gradient. This gradient reduces the hyperspectral image to one band as shown in Fig. 5. For each pixel, and for each band of the pixel, the horizontal and vertical terms of the gradient are calculated separately. Then, the norm of these terms is computed and the value of the gradient for each band is accumulated to produce the final gradient value for each pixel. This process is computed in shared memory (optimization technique 1 in Sect. 2.1) and requires a load operation in shared memory for each band of the image.

Then, a segmentation map is calculated through a watershed technique based on a cellular automaton [30] called CA-watershed. The watershed segmentation fits this approach particularly well because it produces over-segmented region maps, minimizing the probability of including more than one semantic object in the same region. The input is the one-band image obtained by the previous gradient (See Fig. 6). The

Fig. 2 Change detection flowchart in the spectral–spatial framework



- (I) **Segmentation.**
Input: two co-registered hyperspectral images.
Output: two segmented hyperspectral images.
 (1) Generate the segmented images through watershed.
- (II) **Fusion.**
Input: two segmented hyperspectral images.
Output: difference image.
 (1) Apply CVA with SAM distance to obtain a difference image.
- (III) **Thresholding.**
Input: difference image.
Output: raw change map.
 (1) Obtain the range of the data in the image.
 (2) Calculate the histogram of the data.
 (3) Calculate threshold through Otsu’s method.
 (4) Generate binary image according to the calculated threshold.
- (IV) **Spatial regularization.**
Input: raw change map.
Output: spatially regularized change map.
 (1) Apply neighbor-based spatial regularization to the raw change map.

Fig. 3 Work-flow of the GPU-based spectral–spatial framework for change detection

CA-watershed is implemented asynchronously (exploiting optimization technique 2 in Sect. 2.1), being up to five times faster than the synchronous CUDA implementation [29]. It includes intra-block asynchronous updates computed in shared memory and inter-block synchronous updates computed in global memory to efficiently deal with the fact that the CA technique needs data from the neighbors of each pixel.

This implementation presents the advantage of reusing information within a block to efficiently exploit the shared and cache memories of the device (optimization 1 in

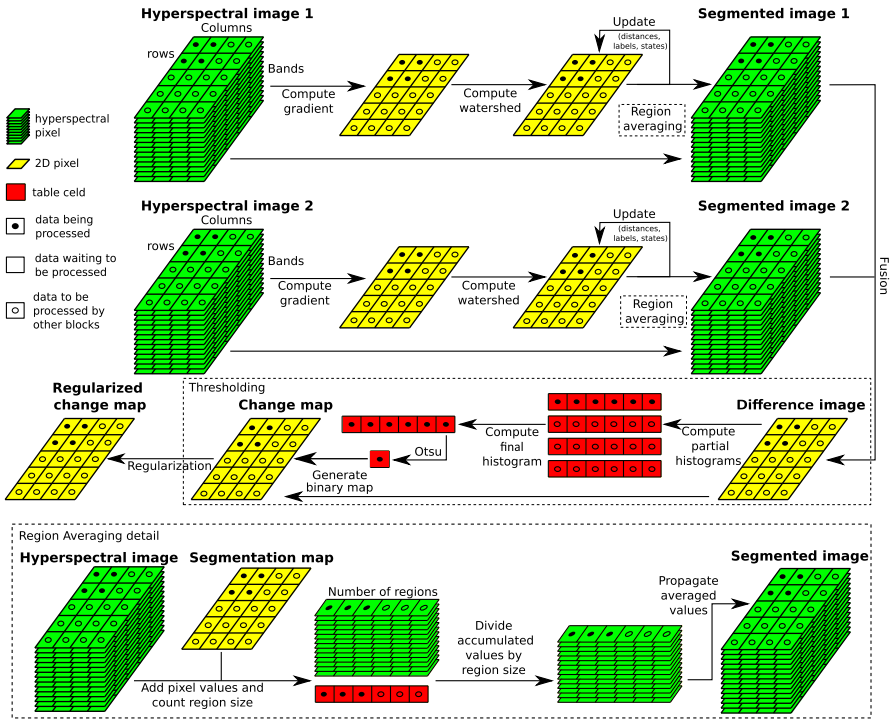


Fig. 4 GPU computation diagram, showing the data structures and the activity of each thread at a given instant of time. The data structures stored in global or shared memory are classified as hyperspectral data, 2D data and table (all the structures that do not correspond to the first two categories). In a given instant of time, data may be computed by a thread of the current block, by a thread of another block or it may be waiting to be processed

```

Gradient on GPU.
Input: hyperspectral image  $X$ .
Output: single band image.
1: for each band  $k$  of  $X$  do                                ▷ Distances step (1)
2:   <Load band  $k$  in shared memory>                            ▷ SM-GM
3:   for each pixel  $i, j$  in band  $k$  do
4:     <Compute the x term of the gradient as                    ▷ SM
        $grad_x(i, j) = I(i + 1, j) - I(i - 1, j)$  >
5:     <Compute the y term of the gradient as                    ▷ SM
        $grad_y(i, j) = I(i, j + 1) - I(i, j - 1)$  >
6:     <Accumulate the gradient module as                        ▷ SM
        $grad(i, j) += \text{sqrt}(grad_x(i, j)^2 + grad_y(i, j)^2)$  >
7:   end for
8:   <Synchronize threads within the block>                      ▷ SM
9: end for
10: <Write  $grad(i, j)$  to global memory>                          ▷ SM-GM
    
```

▷ GM: Global Memory, SM: Shared Memory

Fig. 5 Pseudocode for the gradient kernel required by the watershed-based segmentation corresponding to step (I) in Fig. 3

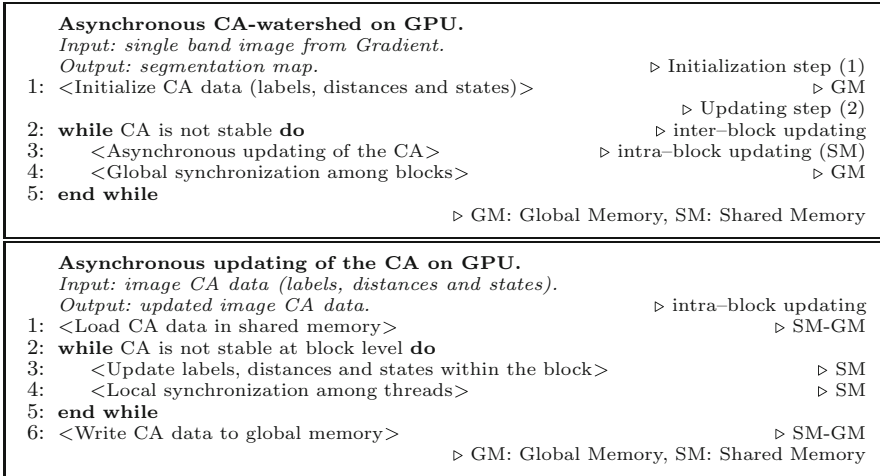


Fig. 6 Pseudocode (main routine and detail of the asynchronous updating function) for the watershed segmentation corresponding to step (I) in Fig. 3

Sect. 2.1). It involves two kernels, implementing the initialization and updating steps of the CA-watershed. They are configured to work in two-dimensional thread blocks as shown in Fig. 4, so that each thread operates over a single pixel. The updating step is an iterative process that lasts until no modifications are made to the available data inside the region (lines 2–5 in Fig. 6). This implementation generates a segmentation map where the pixels are connected so that every pixel in the same region has the same label.

Finally, in the third step of the segmentation, the process to average the spectra of the pixels belonging to the same segmentation region starts (see Figs. 4 and 7). It involves three consecutive kernels (optimization 4 introduced in Sect. 2.1) that are executed in global memory because the size of the regions is unknown and there is no data reuse. The first kernel accumulates the values of each pixel belonging to a region. Each thread atomically adds the spectral values of a pixel to the corresponding region values and also increments a counter for the number of pixels in the region. The second kernel divides the accumulated pixel values for each region by the number of pixels of the region, thus using as many threads as regions in the image. The final kernel propagates the calculated average values to every pixel belonging to each region, needing again as many threads as pixels in the image. In the three kernels each thread computes all the bands of the corresponding pixel. The computation is carried out in-place (optimization technique 1).

After the segmentation stage, the images will have no intra-region variability and a longer inter-region variability. This allows us to detect the changes more easily and to do it at the object level instead of at the pixel level. By assigning the same value to all the pixels in the region, that region (i.e., corresponding to an object or a part of it) will be considered as a whole in the thresholding step.

Region averaging on GPU.
Input: hyperspectral image, watershed segmentation map.
Output: region-averaged hyperspectral image.

- 1: <Add pixel values in each watershed region> ▷ GM
 –Check the region of the pixel in the segmentation map.
 –Atomically add 1 to the corresponding region counter.
 –Atomically add each band of the pixel to the corresponding accumulator.
- 2: <Divide accumulated values by number of pixels in each region> ▷ GM
- 3: <Propagate the averaged value to each pixel of each region> ▷ GM
▷ GM: Global Memory

Fig. 7 Pseudocode for the averaging process in the segmentation of step (I) in Fig. 3

Image fusion on GPU.
Input: two segmented images.
Output: difference image.

- 1: <Apply CVA with SAM distance> ▷ GM
 –Accumulate the product for each band between correspondent pixels.
 –Accumulate the square of each pixel for each band for both images.
 –Calculate distance as $(2/\pi) * \arccos(\text{product}/(\sqrt{\text{firstSquare}} * \sqrt{\text{secondSquare}}))$.
 –Denormalize the distance by a scale factor.
▷ GM: Global Memory

Fig. 8 Pseudocode for the fusion stage corresponding to step (II) in Fig. 3

2.2.2 Fusion Stage on GPU

The following step combines the segmented images into a difference one. This is usually handled by Change Vector Analysis (CVA) processing [33] using Euclidean distance. Our proposal is to create this difference image by using the Spectral Angle Mapper (SAM) distance,

$$\alpha_{i,j} = \frac{2}{\pi} \cos^{-1} \left(\frac{S_j \cdot S_i}{\|S_j\| \|S_i\|} \right) \in [0, 1], \quad (1)$$

where $\alpha_{i,j}$ is the spectral angle between the spectrum at pixel i (S_i) and the one at j (S_j), and i, j are the two pixels under consideration.

The fusion of the two segmented hyperspectral images through CVA is computed by a kernel (Fig. 8) where each thread computes the SAM distance between the two pixel-vectors corresponding to the same spatial location in the two segmented images. This distance is then denormalized by applying a scale factor that allows the subsequent computation of a histogram of the difference image. The distance computation is carried out in global memory generating the difference image in Fig. 4.

2.2.3 Thresholding Stage on GPU

A thresholding process is needed to obtain the change map of the images. In our approach, it is performed following Otsu's algorithm [24]. To do this, a histogram of the previously calculated difference image is required. In order to preserve as much detail as possible, this histogram is created with as many bins as gray levels in the difference image.

Thresholding on GPU.	
<i>Input: difference image.</i>	
<i>Output: raw change map.</i>	
1: <Search data interval in the difference image>	▷ SM-GM
2: <Calculate partial histograms of the data>	▷ GM
3: <Merge partial histograms>	▷ GM
4: <Calculate threshold by Otsu's method>	▷ GM
5: <Generate binary image regarding the calculated threshold >	▷ GM
	▷ GM: Global Memory, SM: Shared Memory

Fig. 9 Pseudocode for the thresholding stage corresponding to step (III) in Fig. 3

This process (see pseudocode in Fig. 9) starts by calculating the range encompassing the values of the difference image. This can be achieved efficiently through the cuBLAS library [22] (optimization strategy 3 in Sect. 2.1). After this, the histogram of the data is efficiently calculated through two consecutive kernels (optimizations 3 and 4 in Sect. 2.1). First, n partial histograms are obtained (being n the number of blocks of threads) as shown in Fig. 4, in a kernel where each thread processes a pixel of the difference image. In the second kernel, the partial histograms are combined in order to obtain the final histogram using as many threads as bins in the histogram. This way each thread writes in a different position of the array.

The next task is to calculate the global value of the threshold using Otsu's algorithm. It assumes two classes of pixels in the histogram, and calculates, in an iterative process, the optimal threshold as the one that provides the maximum inter-class variance. This process is intrinsically sequential (it iterates through the possible thresholds and selects the one resulting in a larger inter-class variance), thus it is computed by a single thread. This is not a problem as long as it is an extremely quick step.

Finally, the CD map is directly created through a process where all the pixels in the difference image whose value is greater than the calculated threshold are identified as changed pixels. The same number of threads as pixels of the difference image are launched in this kernel. The output of the thresholding stage (the change map) is stored over the same memory that contains the magnitude of change for each pixel (optimization technique 1).

2.2.4 Spatial Regularization on GPU

The previous processing can produce some unconnected pixels or noise in the CD map. This is corrected by applying an iterative spatial regularization processing similar to the one introduced in [19].

This stage is computed by one kernel, as shown in Fig. 10. The information of the closest neighborhood of each pixel is exploited in an iterative process. The value of each pixel in the CD map is computed by a thread. A global flag is activated if any pixel is updated, indicating that the iterative process must last one more iteration. The process is computed in-place (optimization technique 1 in Sect. 2.1) in global memory in order to guarantee that the values of the neighbors of a pixel are always correctly updated.

```

Spatial regularization on GPU.
Input: raw change map.
Output: spatially regularized change map.
1: while Global flag > 0 do
2:   <Global flag = 0 >                                ▷ GM
3:   <Apply neighborhood-based spatial regularization>    ▷ GM
4:   if Any pixel value changes then
5:     <Global flag ++ >                                ▷ GM
6:   end if
7: end while

```

▷ GM: Global Memory

Fig. 10 Pseudocode for the spatial regularization stage corresponding to step (IV) in Fig. 3

2.2.5 Comparison to the Sequential and OpenMP CPU Implementations

Some stages of the method present different implementations in the CPU and GPU versions in order to fully exploit each architecture. In particular, the region averaging step of the segmentation stage (Fig. 7) and the fusion stage (Fig. 8) are performed in pixel-vector order (i.e. the image is stored in memory so that the components of a pixel are contiguous) in the CPU implementations. The GPU version works in band-vector order. This aims to maximize the exploitation of the available cache memories in the CPU and maximize the number of threads being executed in parallel in the GPU, respectively. Furthermore, the histogram calculation of the thresholding stage (Fig. 9) follows a two-step processing in GPU, as explained in Sect. 2.2.3, while this is not necessary in the CPU version of the code, that computes it in only one step.

3 Experimental Results

In this section we provide experimental results over hyperspectral images obtained from the AVIRIS sensor. The results will be analyzed in terms of accuracy in the binary detection as compared to the reference detection data and in terms of computational performance of the GPU code. The datasets and the experimental conditions are described in Sect. 3.1. The accuracies achieved and the performance results are detailed in Sect. 3.2.

3.1 Hyperspectral Datasets and Experimental Set-Up

The proposed framework has been evaluated both in CPU and GPU. The specifications of the hardware are detailed in Table 2. The CPU version of the code has been compiled using the gcc compiler version 4.9.4 with OpenMP 3.0 support under Linux. This is, at the moment of running the experiments presented in this paper, the most recent gcc version compatible with CUDA. Optimization level 3 (`-O3` compiler flag) is used in every case. The CUDA code has been compiled using the nvcc compiler with version 8.0 of the toolkit under Linux.

The accuracy of the proposed framework is evaluated in terms of correctly detected changes and total error, which represents the sum of missed alarms (MA) and false

Table 2 CPU and GPU hardware specifications

Hardware	# of cores	Core clock (MHz)	RAM (GB)	L1 (KB)	L2 (KB)	L3 (MB)	GFLOPS (Double)	Bandwidth (GB/s)
Intel Core i5-3470	4	3200	24	64	256	6	102.4	54/26 ^a
TITAN X	3584	1417	12	48	3072	–	317.4	3/3/342 ^b

^aSequential read and sequential write, respectively

^bHost to device, device to host, and device to device transfers, respectively

Table 3 Distribution of the pixels in the available reference data for the CD of the Santa Barbara and Bay Area datasets

	Santa Barbara		Bay Area	
	Absolute	Percentage	Absolute	Percentage
Changed pixels	52,134	7.16	38,425	12.81
Unchanged pixels	80,418	11.04	34,211	11.40
Unlabeled pixels	595,608	81.80	227,364	75.79
Total	728,160	100.00	300,000	100.00

alarms (FA) [5]. These metrics are expressed in absolute and percentage terms. The performance results are expressed in terms of execution time (in seconds) and speedup (the number of times the GPU implementation runs faster with respect to a baseline implementation) compared to an optimized OpenMP version of the framework executed on a multicore CPU using 4 threads. The performance results represent the average of ten independent executions.

The tests were run over two airborne co-registered hyperspectral images taken by the AVIRIS sensor:

- The Santa Barbara dataset (years 2013 and 2014) over the Santa Barbara region (California). Its dimensions are 984×740 pixels \times 224 spectral bands.
- The Bay Area dataset (years 2013 and 2015) in Patterson (California). Its dimensions are 600×500 pixels \times 224 spectral bands.

The images were co-registered using HypeRview [13], a desktop tool that performs registration based on the computation of the multilayer fractional Fourier transform [25]. A reference map of the object-based areas that change between the two images considered from each dataset was constructed by visual inspection. The reference data includes areas with changes and without them, in order to avoid incorrect accuracy results corresponding to misclassified unchanged areas. Table 3 summarizes the number of pixels assigned to each class over the images.¹

3.2 Accuracy and Performance Results

This section presents the accuracy and performance results obtained by the proposed framework over the experimental data.

Table 4 (for the Santa Barbara dataset) and Table 5 (for the Bay Area dataset) show the accuracies obtained using the previously introduced framework (SAM + WAT + Otsu) with CVA based on SAM distance and Otsu's thresholding, along with other configurations included for comparison purposes. EUC + EM can be used as a reference to compare the other methods, as it is frequently done in the bibliography [5]. It includes an Euclidean distance CVA and Expectation-Maximization

¹ The datasets along with the reference maps created and some experimental results can be downloaded from: <https://wiki.citius.usc.es/hiperespectral:cva>.

Table 4 Accuracy results for CD (correctly classified pixels, MA, FA, and total error), in terms of number of pixels and percentages, for the Santa Barbara dataset

Configuration	Correct	MA	FA	Total error
EUC + EM	110,065 (83.04%)	7990	14,497	22,487 (16.96%)
EUC + Otsu	110,348 (83.25%)	9866	12,338	22,204 (16.75%)
SAM + Otsu	125,598 (94.75%)	4147	2807	6954 (5.25%)
EUC + WAT + Otsu	115,101 (86.83%)	7842	9609	17,451 (13.17%)
SAM + WAT + EM	126,389 (95.35%)	1765	4398	6163 (4.65%)
SAM + WAT + Otsu	128,523 (96.96%)	3011	1018	4029 (3.04%)

Best result for each column in bold

Table 5 Accuracy results for CD (correctly classified pixels, MA, FA, and total error), in terms of number of pixels and percentages, for the Bay Area dataset

Configuration	Correct	MA	FA	Total error
EUC + EM	64,530 (88.84%)	4245	3861	8106 (11.16%)
EUC + Otsu	61,745 (85.01%)	8940	1951	10,891 (14.99%)
SAM + Otsu	68,952 (94.93%)	2272	1412	3684 (5.07%)
EUC + WAT + Otsu	62,944 (86.66%)	8476	1216	9692 (13.34%)
SAM + WAT + EM	68,548 (94.37%)	868	3220	4088 (5.63%)
SAM + WAT + Otsu	70,411 (96.94%)	2011	214	2225 (3.06%)

Best result for each column in bold

thresholding. EUC + Otsu includes CVA with Euclidean distance and Otsu based thresholding. SAM + Otsu substitutes the Euclidean distance by the SAM distance. EUC + WAT + Otsu adds the spatial processing based on watershed segmentation plus region averaging and spatial regularization. SAM + WAT + EM and SAM + WAT + Otsu add the same spatial processing to the configuration that uses SAM distance.

The proposed framework provides very good CD results, achieving up to 96.96 and 96.94% of correctly classified pixels for the Santa Barbara and Bay Area datasets, respectively (See Tables 4 and 5). It can be seen that, in general, the configurations based on SAM distance (SAM + Otsu, SAM + WAT + Otsu, SAM + WAT + EM) greatly improve the accuracies obtained by those using Euclidean distance based CVA (all the remaining configurations), achieving up to 11 more percentage points. It is also observed that the spatial information included improves the results in both the EUC and SAM cases. The configurations replacing Otsu's threshold with the Expectation-Maximization (EM) technique (EUC + EM, SAM + WAT + EM) show that Otsu's approximation provides a better thresholding of the difference data and, therefore, a better change map.

The change maps achieved by the different configurations are shown in Figs. 11e–g and 12e–g. The figures also include a color composition of the input images (a, b), the reference data of the changes (c) and the hit map of changes obtained by the best configuration (SAM + WAT + Otsu) in terms of accuracy with respect to the reference

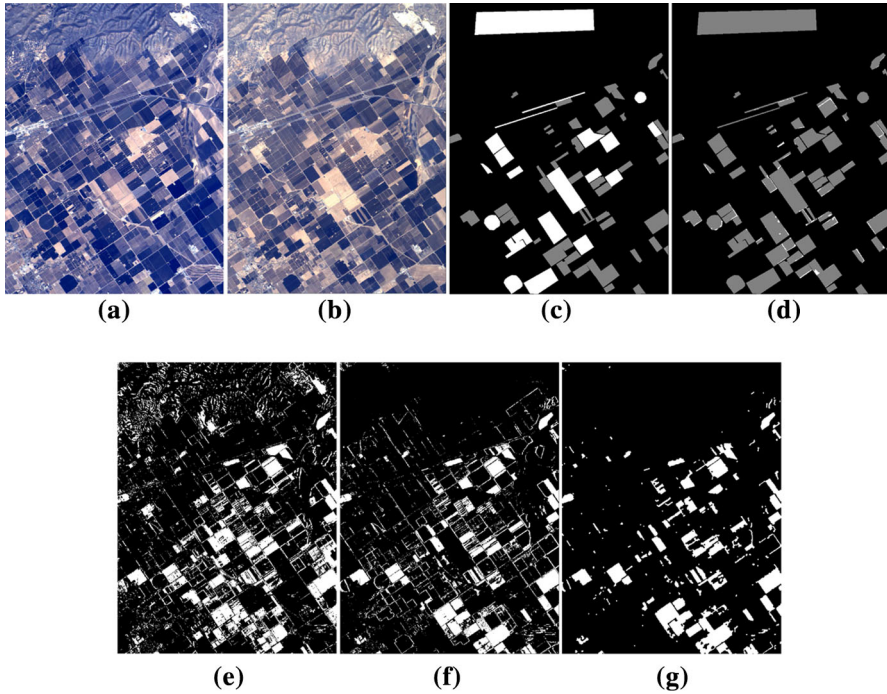


Fig. 11 Color composition of the input images (a, b), reference data of changes {white = no change, gray = change} (c), hit map for SAM+WAT+Otsu {white = miss, gray = hit} (d), EUC+Otsu CD map (e), SAM+Otsu CD map (f), SAM+WAT+Otsu CD map (g) for the Santa Barbara dataset

data (d). As it can be seen in Figs. 11c and 12c, the reference data cover regions with different sizes and shapes throughout the image. Figures 11d and 12d show that the framework correctly detects the presence or absence of changes in a large percentage of the reference data (96.96 and 96.94%, respectively). Figures 11g and 12g show that the spatial information removes most of the noisy pixels present in Figs. 11e, f and 12e, f, improving the final accuracy.

The execution time results obtained in the sequential CPU, the OpenMP CPU using 4 threads, and the CUDA GPU versions of the framework are detailed in Table 6. The OpenMP CPU version of the framework achieves a reasonable speedup when compared to the sequential one, taking account of the number of threads employed. A speedup of up to $46.5\times$ is achieved with the GPU implementation as compared to the fastest CPU version (the one with OpenMP). It can be seen that all the GPU algorithms in the framework achieve high speedups. The stages involving a heavier computational load, i.e., the gradient calculation, the region averaging step, and the spatial regularization stage achieve large speedups ($65.4\times$, $32.4\times$, and $65.6\times$, respectively, for the largest dataset). The smallest speedups correspond to the fusion and thresholding stages ($17.3\times$ and $7.0\times$ for the Santa Barbara dataset and $16.7\times$ and $3.0\times$ for the Bay Area dataset) but they are stages involving low computational load and, therefore, do not significantly affect the total speedup. Finally, it is worth con-

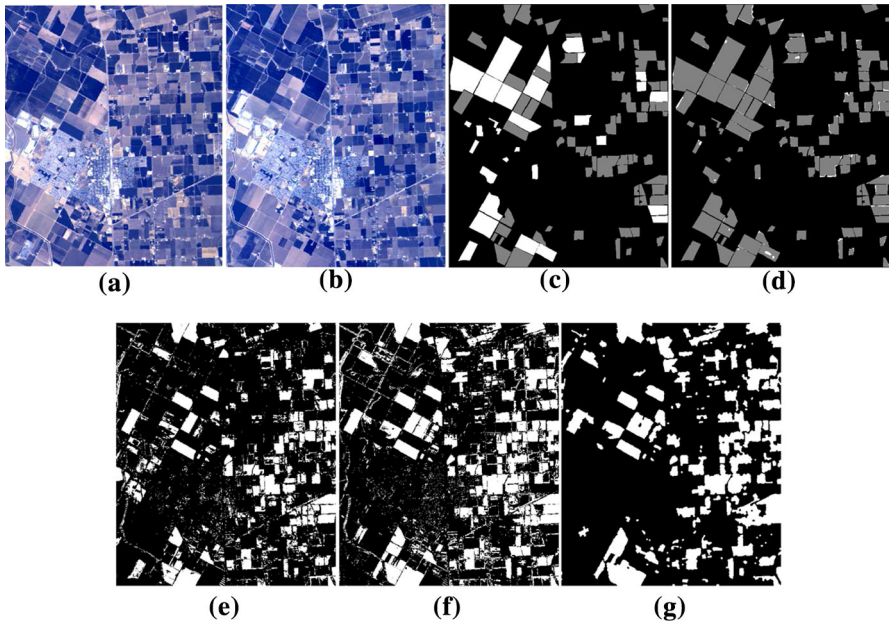


Fig. 12 Color composition of the input images (a, b), reference data of changes {white = no change, gray = change} (c), hit map for SAM+WAT+Otsu {white = miss, gray = hit} (d), EUC+Otsu CD map (e), SAM+Otsu CD map (f), SAM+WAT+Otsu CD map (g) for the Bay Area dataset

sidering that the CPU-GPU transfer time is 0.917 s for the Santa Barbara dataset and 0.353 s for the Bay Area dataset. This means that even if the images have to be loaded into the GPU memory only for this calculation, which is not usually the case in real applications, the total execution time of the GPU version would be faster than the best CPU version. In order to visualize the computational requirements of the framework, it is worth noting that the maximum memory required in GPU at a given instant of time is approximately 2 times the size of one of the images of the dataset. This situation corresponds to the step when the regions are averaged and to the fusion stage.

The hardware occupancy in GPU achieved for the most relevant kernels was analyzed using the *nvprof* tool and it is shown for the Santa Barbara dataset in Table 7 together with the GFLOPS achieved by each stage of the algorithm. The most costly kernels in terms of execution time are those involved in the calculation of the region averaging process (See Fig. 7). These are executed in 512-thread blocks. The second kernel, that divides the accumulated values, achieves 41.2% occupancy because the computational load is too small to hide the operation latency. The kernel for the gradient calculation (See Fig. 5) uses bi-dimensional blocks of threads of size 32×4 and achieves an occupancy of 49.4%, limited by the amount of shared memory available (this information is provided by *nvprof*). The next most relevant kernel is the one devoted to the regularization of the change map (Fig. 10) because it is an iterative kernel executed 46 times for the Santa Barbara dataset in this example. This kernel uses blocks of 512 threads achieving an occupancy of 82%. The limiting factor for

Table 6 Performance results in terms of execution time (in seconds) and speedups of the GPU framework for CD with the SAM + WAT + Otsu configuration

Dataset Stage	Santa Barbara				Bay Area			
	CPU	OpenMP	GPU	Speedup	CPU	OpenMP	GPU	Speedup
Gradients	4.838	1.441	0.022	65.4×	1.887	0.587	0.010	58.7×
Watershed	0.311	0.136	0.004	34.0×	0.229	0.095	0.003	31.6×
Average regions	2.651	0.937	0.029	32.3×	1.068	0.384	0.014	27.4×
Fusion	0.224	0.121	0.007	17.3×	0.094	0.050	0.003	16.7×
Thresholding	0.007	0.007	0.001	7.0×	0.003	0.003	0.001	3.0×
Regularization	1.352	0.984	0.015	65.6×	0.287	0.222	0.004	55.5×
Total	9.383	3.626	0.077	46.5×	3.568	1.341	0.034	38.3×

CPU sequential implementation, *OpenMP* optimized parallel implementation using 4 threads
Speedup represents the factor by which the GPU version is faster than the OpenMP one

Table 7 Performance results in terms of execution time (in seconds), achieved GFLOPS, memory utilization, and occupancy of the GPU framework for CD with the SAM + WAT + Otsu configuration for the Santa Barbara dataset

Stage	Sub-stage	Time	GFLOPS	Memory utilization (%)	Achieved occupancy (%)	Theoretical occupancy (%)
Gradients		0.022	89	35	49.4	50
Watershed		0.004	–	35	95.8	100
Average regions	Addition	0.017	–	85	98.5	100
	Division	0.002	187	25	41.2	50
	Propagation	0.010	–	65	99.5	100
Fusion		0.007	148	85	97.9	100
Thresholding		0.001	–	75	65.9	100
Regularization		0.015	7	95	82.0	100

this kernel, obtained by *nvprof*, is the memory bandwidth of the L2 cache memory. The kernel devoted to the fusion (See Fig. 8) achieves an occupancy of 97.9%.

As it can be seen in Table 7, most of the kernels are limited by the memory utilization (memory utilization above 75% in the table), which prevents the full exploitation of the computing resources. The kernels with enough computational load and not limited by the memory utilization are the ones that achieve more GFLOPS (the division kernel in the region averaging and the fusion kernel). It is important to note that some of the kernels execute only integer or single precision operations.

3.3 Multi-GPU Projection

The framework presented includes several steps performed independently over each one of the input images. This makes it a good candidate to be projected into a multi-GPU system in order to achieve better execution times.

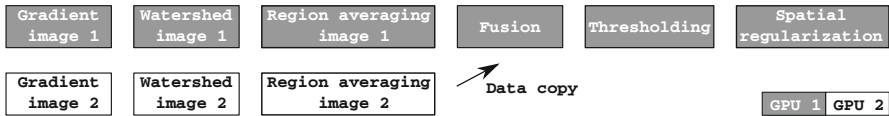


Fig. 13 Multi-GPU change detection flowchart

Table 8 Single and dual GPU performance results in terms of execution time (in seconds) and speedups of the GPU framework for CD with the SAM + WAT + Otsu configuration

Stage	1 GPU	2 GPUs	Speedup
Gradients	0.076	0.038	2.0×
Watershed	0.014	0.007	2.0×
Average regions	0.053	0.030	1.8×
Memory Movements	–	0.004	–
Fusion	0.014	0.014	1.0×
Thresholding	0.002	0.002	1.0×
Regularization	0.022	0.022	1.0×
Total	0.181	0.117	1.5×

We have developed a version of the framework following the flowchart of Fig. 13. The gradient calculation, watershed segmentation and region averaging processes for both images are executed in parallel in two independent GPUs, whereas the fusion, thresholding and spatial regularization steps are performed in a single GPU. This flowchart was designed aiming to minimize the data movements among GPUs. Once the region averaged images are generated, the fusion process dramatically reduces the data size, decreasing the computational cost of the last stages of the algorithm. As a consequence, if the code were executed using both GPUs, the GPU computational load would not be enough to hide the access latency to the data. For this reason, the last 3 processes of the CD are performed in a single GPU.

To carry out the experiments in a multi-GPU system we have used the FT2 cluster at CESGA [7]. In this cluster, each computing node includes an Intel Xeon E5-2680 v3 processor at 2.50 GHz with 128 GB of RAM. A three-level cache hierarchy is available, with 30 MB of shared L3. Four of the nodes also include a NVIDIA Tesla K80 device. The K80 has a dual-GPU design with Kepler architecture including 2496 cores at 0.82 GHz and 12 GB of memory per GPU. Each GPU also incorporates a L1 / L2 cache hierarchy of 1.5 MB. The L1 cache is available only for the threads running in the same SM, the L2 cache is shared among all the threads. The code was compiled using the nvcc version 8.0, and OpenMP is used to manage the two GPUs in parallel. The experiments were run over the Santa Barbara dataset.

Table 8 shows the comparison between the execution times corresponding to a single GPU execution versus a dual GPU execution in the previously presented system. The single GPU version is slower than the one presented in Table 6 because the NVIDIA Tesla K80 cores achieve 0.82 GHz whereas the TITAN X cores achieve 1.4 GHz. We obtain a speedup of 1.5× using two GPUs. The steps that have been parallelized represent 80% of the computation time in the single GPU version. Furthermore, the

data loads are also $1.5\times$ faster when two GPUs are employed because each input image can be loaded in parallel into the GPU memory.

4 Conclusions

An efficient GPU framework for CD in remote sensing multitemporal hyperspectral images is presented in this paper. In the framework, the fusion of multitemporal data is carried out at the feature level through CVA based on SAM distance. The decision on the changes is based on Otsu's thresholding, which improves on the Expectation-Maximization technique in terms of computational burden. The framework also involves spatial processing based on averaging the pixels of the input images belonging to the same spatial region in a watershed-based segmentation map. Finally, a spatial regularization of the change map is applied to remove disconnected pixels. The proposed framework achieves accuracies of up to 96.96% in hyperspectral datasets from the AVIRIS sensor. The GPU projection of the method reaches a speedup of $46.5\times$ as compared to the OpenMP CPU version, requiring less than 0.08 s for 984×740 pixel images including 224 spectral bands. A multiple GPU version of the framework has also been implemented and tested offering good speedups of up to $1.5\times$ for the same images. As future work, the GPU framework will be extended with new algorithms for object-based CD.

Acknowledgements This work has received financial support from the Ministry of Science and Innovation, Government of Spain, co-funded by the FEDER funds of the European Union, under Contracts TIN2013-41129-P and TIN2016-76373-P; Xunta de Galicia, Programme for Consolidation of Competitive Research Groups Ref. 2014/008; the Consellería de Cultura, Educación e Ordenación Universitaria (Accreditation 2016-2019, ED431G/08); and the European Regional Development Fund (ERDF).

References

1. Bannari, A., Morin, D., Bonn, F., Huete, A.: A review of vegetation indices. *Remote Sens. Rev.* **13**(1–2), 95–120 (1995)
2. Bernabé, S., Sánchez, S., Plaza, A., López, S., Benediktsson, J.A., Sarmiento, R.: Hyperspectral unmixing on GPUs and multi-core processors: a comparison. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **6**(3), 1386–1398 (2013)
3. Bioucas-Dias, J., Plaza, A., Camps-Valls, G., Scheunders, P., Nasrabadi, N., Chanussot, J.: Hyperspectral remote sensing data analysis and future challenges. *IEEE Geosci. Remote Sens. Mag.* **1**(2), 6–36 (2013)
4. Bovolo, F., Bruzzone, L.: The time variable in data fusion: a change detection perspective. *IEEE Geosci. Remote Sens. Mag.* **3**(3), 8–26 (2015)
5. Bruzzone, L., Prieto, D.F.: Automatic analysis of the difference image for unsupervised change detection. *IEEE Trans. Geosci. Remote Sens.* **38**(3), 1171–1182 (2000)
6. Celik, T.: Unsupervised change detection in satellite images using principal component analysis and means clustering. *IEEE Geosci. Remote Sens. Lett.* **6**(4), 772–776 (2009)
7. CESGA: Finis Terrae II Quick start guide (2017). <https://cesga.es/en/paginas/descargaDocumento/id/231>
8. Chen, Z., Vatsavai, R.R., Ramachandra, B., Zhang, Q., Singh, N., Sukumar, S.: Scalable nearest neighbor based hierarchical change detection framework for crop monitoring. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 1309–1314. IEEE (2016)
9. Christophe, E., Michel, J., Inglada, J.: Remote sensing processing: from multicore to GPU. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **4**(3), 643–652 (2011)

10. Dennison, P.E., Halligan, K.Q., Roberts, D.A.: A comparison of error metrics and constraints for multiple endmember spectral mixture analysis and spectral angle mapper. *Remote Sens. Environ.* **93**(3), 359–367 (2004)
11. Falco, N., Mura, M.D., Bovolo, F., Benediktsson, J.A., Bruzzone, L.: Change detection in VHR images based on morphological attribute profiles. *Geosci. Remote Sens. Lett. IEEE* **10**(3), 636–640 (2013)
12. Fauvel, M., Tarabalka, Y., Benediktsson, J.A., Chanussot, J., Tilton, J.C.: Advances in spectral–spatial classification of hyperspectral images. *Proc. IEEE* **101**(3), 652–675 (2013)
13. Garea, A.S., Ordóñez, A., Heras, D.B., Argüello, F.: HypeRvIEW: an open source desktop application for hyperspectral remote-sensing data processing. *Int. J. Remote Sens.* **37**(23), 5533–5550 (2016)
14. Ghosh, A., Subudhi, B.N., Bruzzone, L.: Integration of Gibbs Markov random field and Hopfield-type neural networks for unsupervised change detection in remotely sensed multitemporal images. *IEEE Trans. Image Process.* **22**(8), 3087–3096 (2013)
15. Hao, M., Shi, W., Zhang, H., Li, C.: Unsupervised change detection with expectation-maximization-based level set. *IEEE Geosci. Remote Sens. Lett.* **11**(1), 210–214 (2014)
16. Ke, J., Sowmya, A., Guo, Y., Bednarz, T., Buckley, M.: Efficient GPU computing framework of cloud filtering in remotely sensed image processing. In: 2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA), pp. 1–8. IEEE (2016)
17. Kempeneers, P., Sedano, F., Strobl, P., McInerney, D.O., San-Miguel-Ayanz, J.: Increasing robustness of postclassification change detection using time series of land cover maps. *IEEE Trans. Geosci. Remote Sens.* **50**(9), 3327–3339 (2012)
18. Keshava, N.: Distance metrics and band selection in hyperspectral processing with applications to material identification and spectral libraries. *IEEE Trans. Geosci. Remote Sens.* **42**(7), 1552–1565 (2004)
19. López-Fandiño, J., Quesada-Barriuso, P., Heras, D.B., Argüello, F.: Efficient ELM-based techniques for the classification of hyperspectral remote sensing images on commodity GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **8**(6), 2884–2893 (2015)
20. Malila, W.A.: Change vector analysis: an approach for detecting forest changes with landsat. In: LARS Symposia, p. 385 (1980)
21. Nielsen, A.A.: Kernel based orthogonalization for change detection in hyperspectral image data. In: 6th EARSeL Workshop on Imaging Spectroscopy (2013)
22. Nvidia: CUBLAS Library User Guide (2013)
23. Nvidia: NVIDIA Tesla P100. The Most Advanced Data Center Accelerator Ever Built. Featuring Pascal P100, the Worlds Fastest GPU (2016)
24. Otsu, N.: A threshold selection method from gray-level histograms. *Automatica* **11**(285–296), 23–27 (1975)
25. Pan, W., Qin, K., Chen, Y.: An adaptable-multilayer fractional Fourier transform approach for image registration. *IEEE Trans. Pattern Anal. Mach. Intell.* **31**(3), 400–414 (2009)
26. Plaza, A., Du, Q., Chang, Y.L., King, R.L.: High performance computing for hyperspectral remote sensing. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **4**(3), 528–544 (2011)
27. Plaza, A., Plaza, J., Paz, A., Sanchez, S.: Parallel hyperspectral image and signal processing. *IEEE Signal Process. Mag.* **28**(3), 119–126 (2011)
28. Quesada-Barriuso, P., Argüello, F., Heras, D.B.: Computing efficiently spectral–spatial classification of hyperspectral images on commodity GPUs. In: Tweedale, J., Jain, L. (eds.) *Recent Advances in Knowledge-Based Paradigms and Applications*, pp. 19–42. Springer, Berlin (2014)
29. Quesada-Barriuso, P., Heras, D.B., Argüello, F.: Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata. *Comput. Electr. Eng.* **39**(8), 2638–2655 (2013)
30. Quesada-Barriuso, P., Heras, D.B., Argüello, F.: Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), pp. 79–86. IEEE (2012)
31. Sánchez, S., Ramalho, R., Sousa, L., Plaza, A.: Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. *J. Real Time Image Proc.* **10**(3), 469–483 (2015)
32. Singh, A.: Review article digital change detection techniques using remotely-sensed data. *Int. J. Remote Sens.* **10**(6), 989–1003 (1989)
33. Singh, S., Talwar, R.: Review on different change vector analysis algorithms based change detection techniques. In: 2013 IEEE Second International Conference on Image Information Processing (ICIIP), pp. 136–141. IEEE (2013)

34. Tarabalka, Y., Benediktsson, J.A., Chanussot, J., Tilton, J.C.: Multiple spectral–spatial classification approach for hyperspectral data. *IEEE Trans. Geosci. Remote Sens.* **48**(11), 4122–4132 (2010)
35. Van der Meer, F.D., van der Werff, H., van Ruitenbeek, F.J., Hecker, C.A., Bakker, W.H., Noomen, M.F., van der Meijde, M., Carranza, E.J.M., Smeth, J., Woldai, T.: Multi-and hyperspectral geologic remote sensing: a review. *Int. J. Appl. Earth Obs. Geoinf.* **14**(1), 112–128 (2012)
36. Volpi, M., Tuia, D., Bovolo, F., Kanevski, M., Bruzzone, L.: Supervised change detection in vhr images using contextual information and support vector machines. *Int. J. Appl. Earth Obs. Geoinf.* **20**, 77–85 (2013)
37. Yang, B., Yang, M., Plaza, A., Gao, L., Zhang, B.: Dual-mode FPGA implementation of target and anomaly detection algorithms for real-time hyperspectral imaging. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **8**(6), 2950–2961 (2015)
38. Zhang, W., Luo, G., Shen, L., Page, T., Li, P., Jiang, M., Maass, P., Cong, J.: FPGA acceleration by asynchronous parallelization for simultaneous image reconstruction and segmentation based on the Mumford–Shah regularization. In: *SPIE Optical Engineering + Applications*, p. 96000H. International Society for Optics and Photonics (2015)
39. Zhu, H., Cao, Y., Zhou, Z., Gong, M.: Parallel multi-temporal remote sensing image change detection on GPU. In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, pp. 1898–1904. IEEE (2012)