Journal of Real-Time Image Processing
https://doi.org/10.1007/s11554-018-0765-0

SPECIAL ISSUE PAPER

A suite of parallel algorithms for efficient band selection from hyperspectral images

Alessandro Fontanella[1] · Elisa Marenzi[1] · Emanuele Torti[1] · Giovanni Danese[1] · Antonio Plaza[2] · Francesco Leporati[1]

Received: 17 November 2017 / Accepted: 9 March 2018

Abstract
The analysis of hyperspectral images is usually very heavy from the computational point-of-view, due to their high dimensionality. In order to avoid this problem, band selection (BS) has been widely used to reduce the dimensionality before the analysis. The aim is to extract a subset of the original bands of the hyperspectral image, preserving most of the information contained in the original data. The BS technique can be performed by prioritizing the bands on the basis of a score, assigned by specific criteria; in this case, BS turns out in the so-called band prioritization (BP). This paper focuses on BP algorithms based on the following parameters: signal-to-noise ratio, kurtosis, entropy, information divergence, variance and linearly constrained minimum variance. In particular, an optimized C serial version has been developed for each algorithm from which two parallel versions have been derived using OpenMP and NVIDIA's compute unified device architecture. The former is designed for a multi-core CPU, while the latter is designed for a many-core graphics processing unit. For each version of these algorithms, several tests have been performed on a large database containing both synthetic and real hyperspectral images. In this way, scientists can integrate the proposed suite of efficient BP algorithms into existing frameworks, choosing the most suitable technique for their specific applications.

Keywords Hyperspectral imaging · Band selection (BS) · Band prioritization (BP) · Real-time processing · Central processing unit (CPU) · Graphics processing unit (GPU)

1 Introduction

Hyperspectral sensors acquire scenes at different wavelengths of the electromagnetic spectrum, and they produce, as output, the so-called *image cube*. Contiguous wavelengths are grouped into *bands*, and each pixel of the cube can be represented by three components (x, y, z), where x and y denote a spatial position on the scene, while z identifies a particular band. Thus, a hyperspectral cube is obtained that represents the same image acquired at *multiple* different wavelengths. Typically, its memory size can range from hundreds of megabytes up to many gigabytes. For this reason and due to the complexity of the algorithms used, the analysis of the image cube is characterized by high computational requirements. In order to tackle this problem, different approaches have been designed with the aim of reducing information redundancy, that is due to the high correlation between adjacent bands of the hyperspectral image. A common adopted strategy is *dimensionality reduction* (DR), that produces a new compact dataset by applying a transformation based on a suitable criterion. For instance, the principal component analysis (PCA) performs DR in order to maximize the variance of the transformed data. DR algorithms have to deal with two issues. The first one is related to the choice of the dimension number needed to avoid significant loss of information. The second issue concerns the transformation

✉ Francesco Leporati
francesco.leporati@unipv.it

Alessandro Fontanella
alessandro.fontanella01@universitadipavia.it

[1] Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Pavia, Italy

[2] Department of Technology of Computers and Communications, Escuela Politecnica de Caceres, University of Extremadura, Cáceres, Spain

Published online: 21 March 2018

of the input data, which can compromise or corrupt the original information [1].

Another approach to reduce information redundancy is the so-called *band selection* (BS), introduced in the early 1990s [2–4] and it has been improved in recent years [5–7]. The BS technique can be divided into two categories: supervised and unsupervised. In supervised approaches, the knowledge about classes or objects is inside the hyperspectral image, while in unsupervised approaches no previous information is required. BS techniques extract a bands subset of the input hyperspectral image that can represent to the best the original data. Since no transformation is applied on the input dataset, it is possible to preserve the original information. This property represents the main advantage compared to the DR strategy. However, in this case it is also necessary to deal with two issues: the choice of the number of bands to preserve and the BS criterion. In particular, in this work we decided to adopt a particular BS methodology called *band prioritization* (BP) since it is the most recent and accurate one. This approach assigns a score to each band using a suitable approach. Then, the BS is performed by extracting a certain number of bands on the basis of each score.

The key point in the BP approach is the choice of the criterion used to assign a score to each band. Different approaches have been analyzed. Some of them, like signal-to-noise ratio (SNR), kurtosis, entropy, ID and variance [8], are based on statistical techniques. Another criterion relies on the concept of linearly constrained minimum variance (LCMV) [1]. All of them can be adopted in BP algorithms, which therefore could feature a high computational complexity. The operations involved in those algorithms are however intrinsically parallel. For this reason, it is possible to develop efficient solutions based on specific many-core devices such as graphics processing units (GPUs) which provide high computational capability at low cost. Recently, GPUs are being used not only in computer graphics, but also for general purpose applications [9]. For instance, in the remote sensing field, GPUs have been successfully employed in real-time hyperspectral unmixing [10–14].

This work focuses on the development of a suite of parallel BP algorithms based on different methodologies. In particular, the aim of this paper is to provide scientists with a set of computationally efficient algorithms that can be integrated in existing frameworks. In this way, it is possible to choose the best algorithm depending on the purposes and constraints of each application, such as the ones requiring real-time processing, i.e., fire detection or pollution monitoring.

The development phase started from a MATLAB implementation based on the algorithms proposed in [1, 8]. A serial C code version has been derived, and then, the OpenMP API has been adopted for a multi-core CPU implementation. One last parallelization has been carried out with an NVIDIA GPU based on the Kepler architecture and using the NVIDIA compute unified device architecture (CUDA) standard. In particular, an accurate comparison among the C serial, OpenMP and CUDA solutions has been conducted to identify the most suitable implementation of each algorithm.

The paper is organized as follows: Sect. 2 explains all the BP algorithms considered in this work. Sections 3 and 4 contain the description of the strategy followed to parallelize the algorithms through GPU technology and OpenMP API, respectively. It is important to highlight that the focus of the proposed work concerns the GPU implementation. Therefore, the OpenMP versions of the algorithms have been concisely described. Section 5 shows the results of tests performed on a wide hyperspectral image database, while Sect. 6 concludes the paper with some remarks and hints to possible future research lines.

2 Band prioritization algorithms

In this section, the BP algorithms will be analyzed considering an input hyperspectral data cube with L bands, n_r rows, n_c columns and N pixels.

2.1 SNR-based algorithm

In order to understand the behavior of the SNR-based algorithm, it is necessary to introduce the two sets $\{\lambda_{\mathrm{adj}_l}\}_{l=1}^{L}$ and $\{\upsilon_{k_l}\}_{l=1}^{L}$ which contain the eigenvalues of the noise-adjusted sample covariance matrix and their associated orthonormal eigenvectors, respectively. Thus, a priority score can be calculated for the l-th hyperspectral image band as follows:

$$\rho_l = \sum_{k=1}^{L} \xi_{k_l}^2 \tag{1}$$

where ξ_{k_l} is a loading factor given by:

$$\xi_{k_l} = \sqrt{\lambda_{\mathrm{adj}_k}}\,\upsilon_{k_l} \tag{2}$$

2.2 Entropy-based algorithm

As far as the entropy-based algorithm is considered, it is necessary to evaluate the image histogram (p) for each hyperspectral band. The total number of bins (NBINS) depends on the amount of grey-scale levels adopted. After that, the l-th band priority score is calculated as follows:

$$\rho_l = \sum_{k=0}^{\text{NBINS}} p_{l_k} \cdot \log_2(p_{l_k}) \tag{3}$$

where p_{l_k} represents the value contained in the kth histogram bin evaluated for the image of the lth band.

2.3 ID-based algorithm

Instead of using the entropy (3), a different criterion based on the information divergence (ID) can be adopted. The ID-based algorithm evaluates the band priority score for the lth band with the following equation:

$$\rho_l = \sum_{k=0}^{\text{NBINS}} p_{l_k} \cdot \log_2\left(\frac{p_{l_k}}{g_{l_k}}\right) + \sum_{k=0}^{\text{NBINS}} g_{l_k} \cdot \log_2\left(\frac{g_{l_k}}{p_{l_k}}\right) \tag{4}$$

where p_{l_k} is the image histogram, composed by NBINS bins, for the lth band which must be normalized and g_{l_k} is the lth band Gaussian distribution with mean and variance relative to the image of the lth band.

2.4 Variance-based algorithm

The last algorithm considered, based on statistics techniques, evaluates the variance σ^2 for each band of the hyperspectral cube and is used as a measure of the band priority as follows:

$$\rho_l = \sigma_l^2 \tag{5}$$

2.5 LCMV-based algorithm

Concerning the algorithms based on the concept of LCMV, four different versions have been investigated: LCMV-band correlation minimization (LCMV-BCM), LCMV-band dependence minimization (LCMV-BDM), LCMV-band correlation constraint (LCMV-BCC) and LCMV-band dependence constraint (LCMV-BDC). All these algorithms are based on the same concept, but they assign the band priority with different methods.

As said before, for each band the acquired image has n_r rows and n_c columns.

First, the lth band image of the hyperspectral cube can be expressed with a matrix notation:

$$B_l = \left[c_{l,1}, c_{l,2}, \ldots, c_{l,n_\text{c}}\right] \tag{6}$$

where $c_{l,j}$ is a n_r-dimensional vector:

$$c_{l,j} = \left(c_{l,1j}, c_{l,2j}, \ldots, c_{l,n_\text{r}j}\right)^\text{T} \quad \text{for } j = 1, \ldots, n_\text{c} \tag{7}$$

The goal of the LCMV-based algorithms is to carry out a constrained finite impulse response (FIR) linear filter. Let w_l be a column vector used to specify a FIR filter for the lth band image (B_l), and y_l be the filter output evaluated as follows:

$$y_l = w_l^\text{T} B_l \tag{8}$$

The averaged least squares filter output is given by:

$$\frac{1}{L}\sum_{l=1}^{L} y_l^2 = w_l^\text{T}\left(\frac{1}{L}\sum_{l=1}^{L} B_l B_l^\text{T}\right) w_l \tag{9}$$

Let $\Sigma = \frac{1}{L}\sum_{l=1}^{L} B_l B_l^\text{T}$ be the sample band correlation matrix, the coefficient values of the vector w_l are the solutions to the following minimization problem:

$$\min_{w_l}\left\{w_l^\text{T}\Sigma w_l\right\} \text{ subject to } B_l^\text{T} w_l = 1_{n_\text{c}} \tag{10}$$

where 1_{n_c} is a n_c-dimensional column vector of ones.

The solution to (10) is given by:

$$w_l^\text{LCMV} = \Sigma^{-1} B_l \left(B_l^\text{T}\Sigma^{-1}B_l\right)^{-1} 1_{n_\text{c}} \tag{11}$$

The BCM and BCC priority scores for the lth band are assigned as follows:

$$\rho_l = \left(w_l^\text{LCMV}\right)^\text{T}\Sigma w_l^\text{LCMV} \quad \text{for LCMV-BCM} \tag{12}$$

and

$$\rho_l = \sum_{k=1,k\neq l}^{L} l_{n_\text{c}}^\text{T}\left(B_k^\text{T} w_l^\text{LCMV}\right) \text{ for } \quad \text{LCMV-BCC} \tag{13}$$

However, it is possible to exclude the lth band image (B_l) from the matrix Σ obtaining the so-called *sample band image dependence matrix*:

$$\tilde{\Sigma} = \frac{1}{L-1}\sum_{j=1,j\neq l}^{L} B_j B_j^\text{T} \tag{14}$$

Replacing Σ in (10) with $\tilde{\Sigma}$, a new minimization problem is derived. Its solution is similar to (10) and it is given by:

$$\tilde{w}_l^\text{LCMV} = \tilde{\Sigma}^{-1} B_l \left(B_l^\text{T}\tilde{\Sigma}^{-1}B_l\right)^{-1} 1_{n_\text{c}} \tag{15}$$

The BDM and BDC priority scores for the lth band are assigned as follows:

$$\rho_l = \left(\tilde{w}_l^\text{LCMV}\right)^\text{T}\tilde{\Sigma}^{-1}\tilde{w}_l^\text{LCMV} \text{ for } \quad \text{LCMV-BDM} \tag{16}$$

and

$$\rho_l = \sum_{k=1,k\neq l}^{L} 1_{n_\text{c}}^T\left(B_k^\text{T}\tilde{w}_l^\text{LCMV}\right) \quad \text{for LCMV-BDC} \tag{17}$$

Most of these algorithms feature high computational complexity that implies very long execution times. For this reason, we decided to implement the BS algorithms exploiting the GPU technology since it can guarantee a

significant speedup, to satisfy real-time requirements in many existing frameworks.

3 Algorithm implementations on GPU

In this section, the GPU implementations of the different BS algorithms together with our design choices (which are a key factor to obtain performance improvements) are presented. The employed NVIDIA's GPU processor is based on the "Kepler" architecture [15], made up of 15 streaming multiprocessors (SMXs) that internally contain 192 cores. Each core has a fully pipelined integer and floating point arithmetic logic unit, compliant with the IEEE 754-2008 single and double precision standards. Threads, which have to be executed by the GPU, are divided into blocks that are assigned to SMXs. Then, each block is divided into groups of 32 threads, called *warps*, and executed by assigning each thread to a SMX core. Each SMX contains also 64 KB of *shared memory*, which can be used to exchange data between threads of the same block, and 65,356 registers in which all the private variables of each thread are stored. The described architecture is shown in Fig. 1. Finally, the GPU has an off-chip RAM memory, called *global memory*, which can be used to exchange data with the CPU through the PCI-express bus. This memory features a higher latency than the shared one. The code, to be executed on the GPU, has been developed using CUDA, a parallel computing platform and programming model introduced by NVIDIA. The GPU can be seen as a coprocessor that, after being invoked by the CPU, can execute a task through thousands of threads running in parallel. The execution of the program starts from the CPU, identified as the *host*, until a parallel code section is reached. After that, the GPU, identified as the *device*, starts processing data transmitted by the host to the device memory. Finally, when the device has completed its task, it transfers back the result to the host.

The first step of the proposed work requires to develop a standard serial C version of each algorithm starting from a MATLAB one. This allows to perform an extensive code profiling in order to identify the most time-consuming parts of each algorithm. Table 1 shows the results of the profiling obtained with a synthetic image of about 170 MB. This table highlights that for the ID, entropy and variance algorithms no routines are heavier than the others. Thus, the notation "Equidistribution" is used to indicate this characteristic.

This table clearly points out which computations are suitable for GPU and OpenMP implementations and which ones should be performed in a serial way. The C implementation has been tested on different synthetic hyperspectral images, whose size ranges from 1.7 MB up to

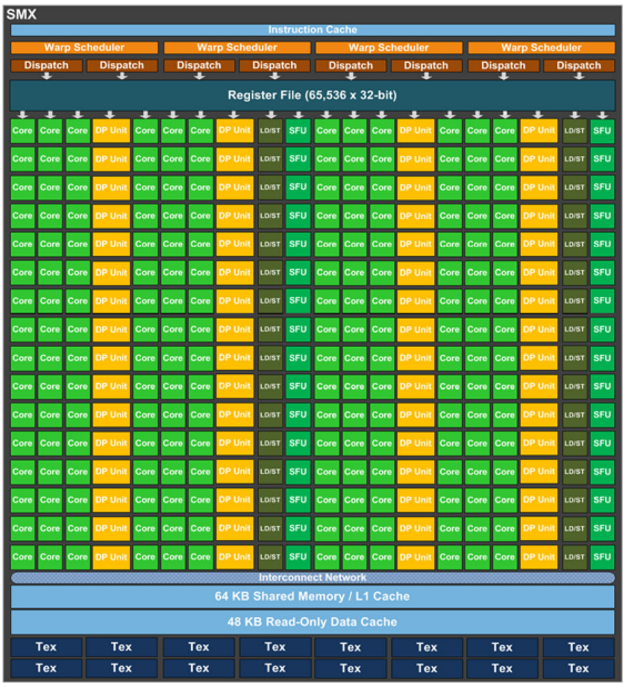


Fig. 1 NVIDIA Kepler SMX architecture

427 MB, which have been generated using the MATLAB script developed by Nascimento et al. [16]. Tests have been used also to decide the numerical precision (single or double precision floating point) that guarantees the same results between the MATLAB and the C versions of each algorithm. The experiments indicate that, for some algorithms, only the double precision arithmetic assures the needed accuracy of results. Since the considered algorithms differ from one another, multiple strategies have to be adopted in order to reach the best performance improvements.

3.1 SNR-based algorithm

Concerning the SNR-BS algorithm, the most time-consuming part is the computation of the correlation matrix. Thus, this task has been assigned to the GPU that, after receiving the hyperspectral image from the host, performs the matrix multiplication and transfers back the result to the CPU. In particular, the cuBLAS library, which contains different highly optimized linear algebra routines, has been used to perform the matrix multiplication through the *cublasDgemm* function that evaluates the following expression:

$$C = \alpha\, \mathrm{op}(A)\mathrm{op}(B) + \beta\, \mathrm{op}(C) \tag{18}$$

where A and B are the input matrices, C is an input/output matrix, and α and β are scalar values. In the cuBLAS library, the matrices are read and stored by default in

Table 1 Profiling results

SNR		ID		Entropy		Variance	
Correlation matrix	99%	Equidistribution		Equidistribution		Equidistribution	
Other	1%						
LCMV							
BCC		BCM		BDC		BDM	
Equation (13)	85%	Equation (12)	89%	Equation (17)	90%	Equation (16)	95%
Equation (9)	13%	Equation (9)	9%	Equation (14)	9%	Equation (14)	4%
$B_l^{\mathrm{T}}\Sigma^{-1}B_l$	1%	$B_l^{\mathrm{T}}\Sigma^{-1}B_l$	1%	Other	1%	Other	1%
other	1%	other	1%				

column-major format while the CUDA C language, chosen for the SNR-BS implementation, adopts the row-major format which is the standard array storing method in the C language. To tackle this problem, matrices can be read as if they were transposed by properly setting the op() parameter in (18). However, the result of an operation is stored in column-major format and this cannot be modified. In this case, this is not a problem since the correlation matrix is symmetric and the C matrix can be transferred back to the host and read as if it was stored in row-major format.

Concerning the choice of the two scalar values, β was set equal to 0 since the C matrix must be only an output matrix while α was set equal to the inverse of the number of pixels of the hyperspectral image.

The correlation matrix is transferred back to the host memory since its dimensions are significantly lower than the hyperspectral image ones. Thus, the subsequent operations can be carried out in a sequential way. The next steps of the algorithm concern the estimation of the noise-adjusted sample covariance matrix and the eigenvalues and eigenvectors computation. This last operation has been done by exploiting two highly optimized routines (*tred2* and *tqli*), described in [17]. The general flow of the algorithm is reported in Fig. 2.

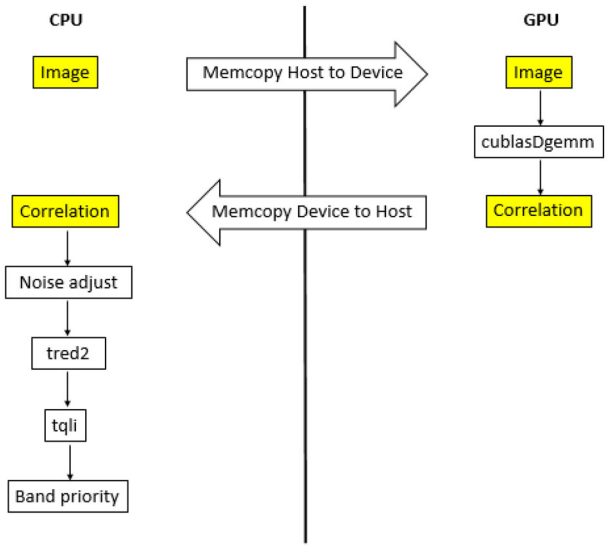


Fig. 2 SNR algorithm flow. Data and operations are in yellow and white boxes, respectively

3.2 Entropy-based algorithm

As shown in Table 1, in the entropy-based algorithm, no portion of the code is more computationally expensive than others. For this reason, we decided to implement the entire algorithm on the GPU in order to obtain a significant speedup. By examining Eq. (3), the band priority assignment requires a summation and the evaluation of the image histogram for each hyperspectral band. In particular, the key point is represented by the development of the summations required by the algorithm. To overcome this, it is possible to apply the reduction technique that exploits GPU parallelism to find the summation of values present in an array stored in the device global memory. First of all, a specific kernel has to be launched. All the threads of each block store a sub-part of the original array inside the shared memory which, as previously said, has a lower latency compared to the global memory. This process requires that each thread loads from the global memory a single element of the original array. Then, only half of the total available threads will be active and each of them will evaluate cyclically the sum of two elements of the array stored in the shared memory until a single value is obtained. Finally, only one thread for each block will write the summation of the sub-array elements in a new array in the global memory. Thereafter, another kernel has to be launched and, using the same strategy as before, it will evaluate the summation of the elements of the new array obtaining the desired result. The strategy adopted by the reduction techniques is shown in Fig. 3.

In order to build the histogram, the array containing the pixel values of a particular band has to be transferred to the

GPU global memory. Then, each thread has to read a single array element and, depending on its value, it has to update the count of the corresponding histogram bin. However, it could happen that another thread, at the same time, has to update the same bin value. This situation could generate a wrong result that may also be different between consecutive executions. To avoid this problem, it is necessary to guarantee that a thread can access a memory address without any interference from other threads, which have to wait until the operation is completed. The CUDA platform provides a series of functions, called *atomic*, which manage this kind of operations both in global and shared memory. In particular, we have used the *atomicAdd* function, which allows to evaluate the image histogram in the correct way.

There is another feature about the GPU implementation that should be mentioned. It has to be taken into account that the atomic operation reduces the overall performance of the algorithm. Indeed, threads, which have to store a value in the same memory location, cannot be executed in parallel. However, this is the only way to obtain a correct image histogram.

The performance can be improved by making use of the shared memory. In a first attempt, the image histogram could be stored in an array in the global memory. All threads of the kernel can see this variable, but the continuous access to the global memory causes a high latency. Thus, the idea is to split the construction of the total histogram in two steps. In the first one, each thread-block reads a sub-part of the input image from the global memory and creates, through atomic operations, a temporary histogram in the shared memory. Then, all the temporary arrays will be stored in the global memory in order to be used in the second step. Moreover, in this case different threads, through atomic operations, cannot update the same bin value at the same time, but now it is possible to exploit the low latency of the shared memory. In the second step, another kernel will be launched to evaluate the final histogram. In particular, each thread will sum all the values present in the temporary arrays.

By analyzing Eq. (3), it is possible to notice that the priority of each band is assigned in an independent way considering every time a different band of the hyperspectral image. This characteristic of the algorithm can be exploited with the GPU stream: a sequence of operations that are executed sequentially on the device. On the other hand, operations in different streams may run concurrently, meaning that the execution of several kernels and data transfers to/from the device memory can be overlapped. In this way, while one stream is executing all the operations required to find the priority of a particular band, a different stream is loading a new band of the hyperspectral image inside the device memory. Of course, this process can be done only if a suitable GPU engine is available. The advantage of overlapping memory transfers and kernel execution can be visualized using the NVIDIA Visual Profiler tool. An example of the output produced by this tool is shown in Fig. 4.

The data flow related to the *i*th stream is shown in Fig. 5.

3.3 ID-based algorithm

In this case, as for the previous technique, the band priority is assigned through simple operations that are not demanding from a computational point of view. For this reason, the entire algorithm has been parallelized on GPU by using all the techniques adopted for the previous algorithms. In particular, the mean and the variance, required

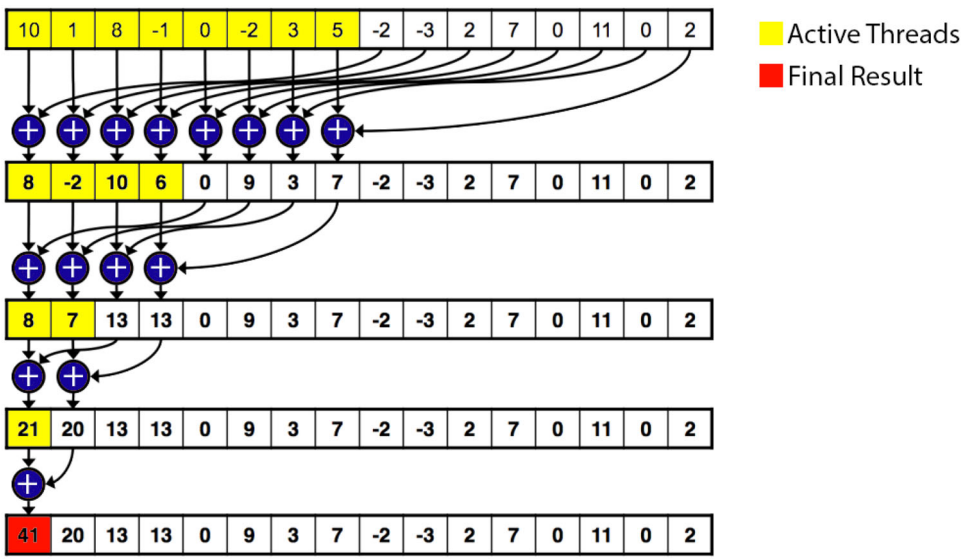


Fig. 3 Schematization of reduction technique

for the evaluation of the Gaussian function, and the final summation can be obtained through the reduction technique. The construction of the image histogram for each hyperspectral band can be developed in the same way as it was done in the entropy-based algorithm. Furthermore, here the computation of band priority is carried out in a scope that is related to a single band. Thus, it is possible to overlap the computation and the memory transfers through CUDA streams. The instructions performed by each stream are schematized in Fig. 6.

3.4 Variance-based algorithm

As it happened in the ID-based algorithm, the variance of each band has been evaluated again through the reduction technique. Even in this case, the use of streams is crucial in order to improve performance. In the same way as other algorithms, a single stream performs the operations related to the assigned band. Figure 7 shows the flow for each stream.

3.5 LCMV-based algorithm

Finally, concerning the four types of LCMV-based algorithm, only some operations have been performed on the GPU device: the evaluation of the sample band correlation and dependence matrices and of the band priority. The former has been carried out, for all the algorithms, by using the *cublasDgemm* and setting β equal to 1 in (18), while the latter has been developed in different ways depending on the particular algorithm.

In the BCC and BDC algorithms (Figs. 8 and 9), the band priority has been evaluated through the *cublasDgemv* function that performs a matrix–vector multiplication:

$$y = \alpha \, \mathrm{op}(A)x + \beta y \tag{19}$$

where A and x are input matrix and vector, respectively, y is an input/output vector, and α and β are scalar values and op() has the same meaning as Eq. (18). In order to obtain the desired result, α and β have been set equal to 1 and 0, respectively.

On the other hand, the *cublasDgemm* and the *cublasDdot* functions have been used to calculate the band priority

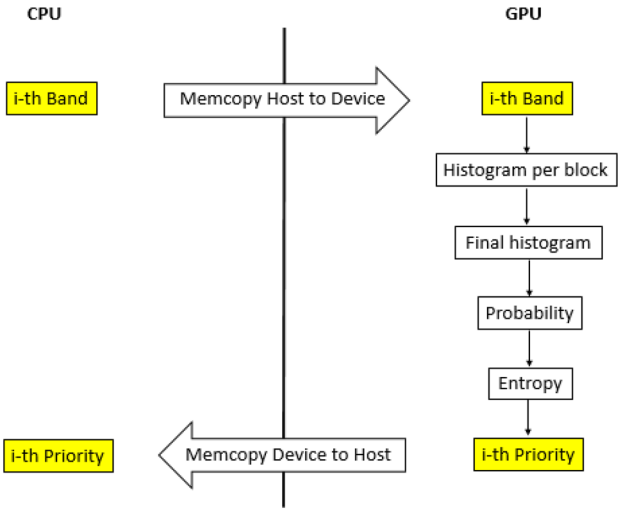


Fig. 5 Entropy algorithm flow for a single stream. Data and operations are in yellow and white boxes, respectively

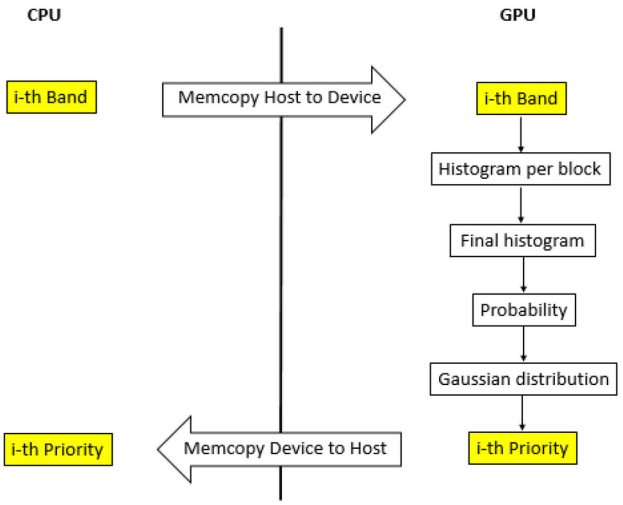


Fig. 6 ID algorithm flow for a single stream. Data and operations are in yellow and white boxes, respectively

for the BDM and BCM algorithms (Figs. 10 and 11). In particular, the *cublasDdot* function requires two vectors of the same length (N) as input (x and y) and performs the dot product operation as follows:

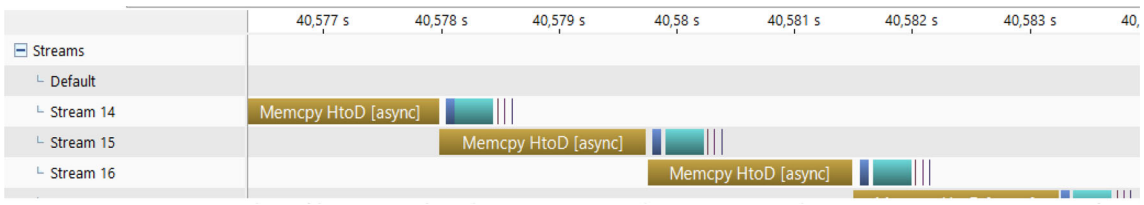


Fig. 4 NVIDIA Visual Profiler output that shows stream overlapping. Brown boxes represent memory transfers from host to device, while the other boxes concern different kernel executions. Data transfers from device to host are omitted in order to preserve readability of this figure

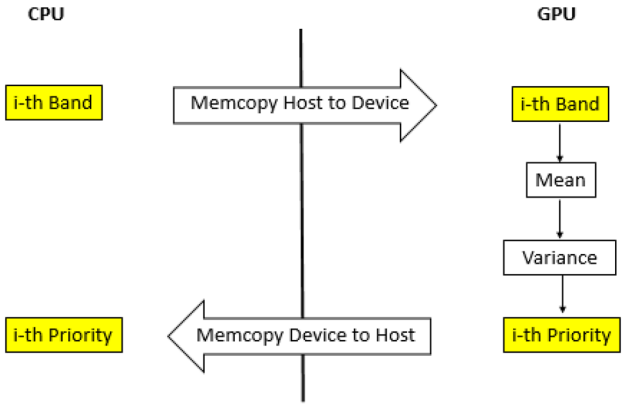


Fig. 7 Variance algorithm flow for a single stream. Data and operations are in yellow and white boxes, respectively

$$\text{res} = \sum_{i=0}^{N} x[i]y[i] \tag{20}$$

The rest of the algorithm has been developed in standard C serial code. In particular, since the band correlation and dependence matrices can be nearly singular due to the working precision, their inverses have to be calculated in a proper way. As far as matrices with big dimensions are considered, it is better to evaluate the pseudo-inverse matrix by applying the singular value decomposition technique, since it can provide a stable result. The main drawback of this solution concerns the execution time, which is longer compared to the one required for evaluating the inverse matrix through the lower–upper factorization technique. After different empirical experiments, a threshold based on the matrix dimensionality has been identified and it allows to choose a proper method to find the inverse matrix. In particular, several tests have been performed with synthetic hyperspectral images of growing sizes and it has been noticed that the lower–upper factorization technique provides stable results for hyperspectral images smaller than 100 MB, while for bigger ones it is necessary to use the pseudo-inversion.

4 Algorithm implementations through OpenMP API

The OpenMP API exploits multi-core parallelism through a set of compiler directives. In this way, the work can be divided into independent elaborations that are assigned to different threads. Starting from the serial C versions, the most time-consuming parts have been parallelized through OpenMP. In particular, the *#pragma omp parallel for* directive allows an automatic parallelization of *for loops*. It is also possible to instruct the compiler on which variables are shared among the threads and which ones are private to the single thread. Typically, the hyperspectral image is declared *shared* while all the *for loop* variables are *private*. Moreover, it is also possible to decide how the work must be subdivided among the threads by using the *schedule* clause inside the OpenMP directives. Since the computational cost of different iterations of the same *for loop* is nearly constant, the static scheduling has been adopted, which equally subdivides all iterations among the threads.

Concerning the SNR-based algorithm, the *#pragma omp parallel for* directive parallelizes the evaluation of the correlation matrix. In this case, the correlation and the hyperspectral image matrices have been declared shared while the accumulator and *for loops* variables are private.

A different approach has been used for the ID, entropy and variance algorithms. The bands are divided in groups, each assigned to a thread that computes the scores. In this case, the hyperspectral image and the score array are shared among the threads while all the support variables are private.

Concerning the LCMV techniques, it is possible to identify two structural similarities: the first between BCC and BDM and the second between BCM and BDM. The former consists in a covariance computation followed by a *for loop* which iterates over the bands to compute the final score. Those two phases have been parallelized, as before, using the *#pragma omp parallel for* directive. Even in this case, the covariance matrix, the hyperspectral image and the band scores have been declared shared while the other support variables are private. On the other hand, the second group consists on a series of matrix–matrix and matrix–vector operations which have been implemented through for loops. Therefore, they can be parallelized using the same OpenMP directive as before.

5 Experimental results

As mentioned before, a C language version has been developed for each algorithm and it has been profiled through synthetic images. After that, the CUDA and OpenMP versions have been carried out. However, before evaluating the execution times, the correctness of each algorithm has been tested by comparing the bands extracted by the MATLAB code and the other implementations. Both synthetic and real hyperspectral images have been used for the comparison. In particular, the real images used in the testing phase are the following: two ROSIS hyperspectral images, five AVIRIS scenes, one EO-1 scene and a Hydice hyperspectral image collected over a forest (hereinafter identified as Hydice). The two ROSIS hyperspectral images are acquired over the city center and the University of Pavia, Italy (later identified with PaviaC and PaviaU). The five AVIRIS hyperspectral images are: Kennedy Space

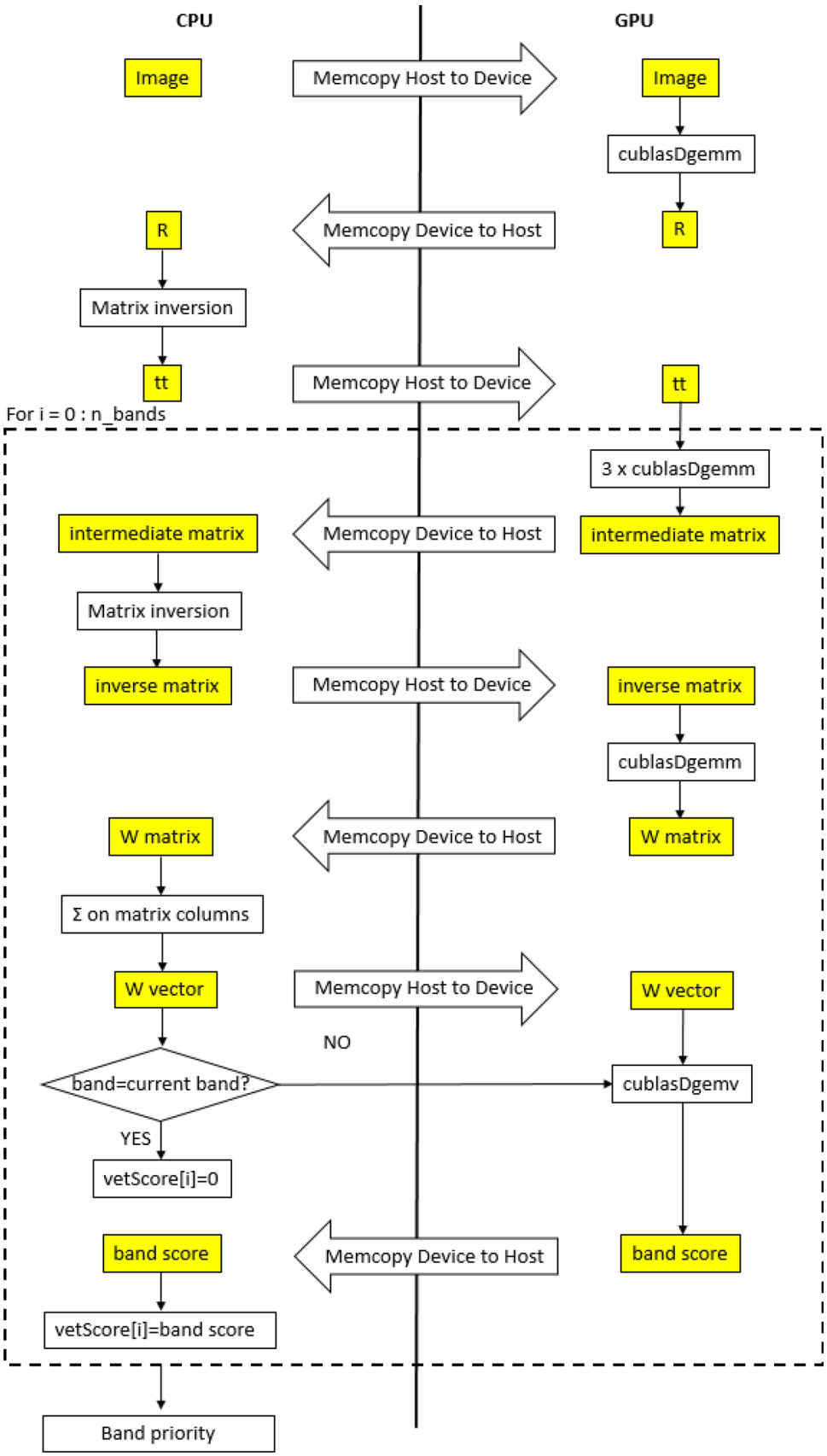


Fig. 8 LCMV-BCC algorithm for the entire matrix. Operations shown in the dotted box are performed for every band in the hyperspectral image. Data and operations are in yellow and white boxes, respectively

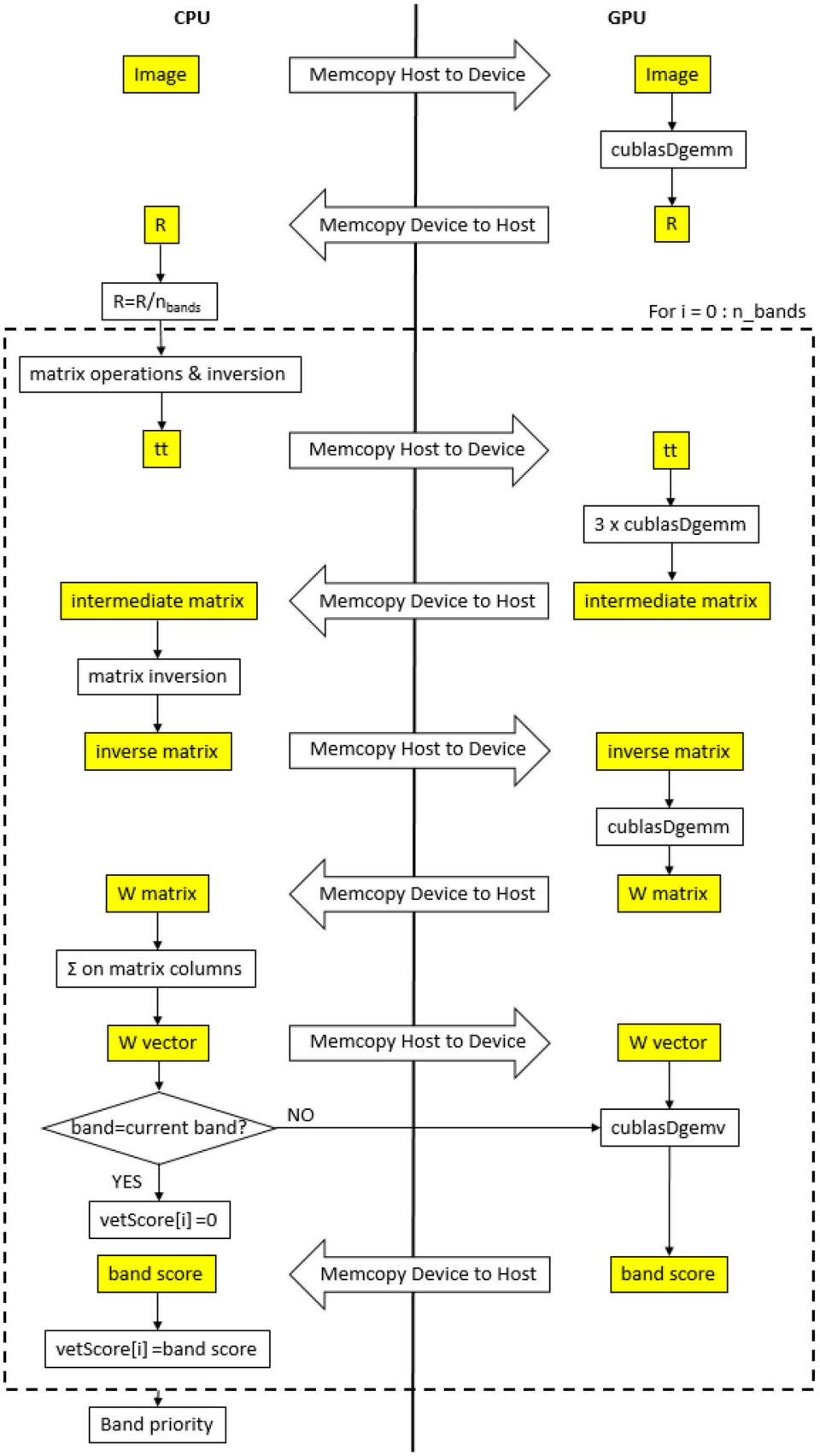


Fig. 9 LCMV-BDC algorithm for the entire matrix. Operations shown in the dotted box are performed for every band in the hyperspectral image. Data and operations are in yellow and white boxes, respectively

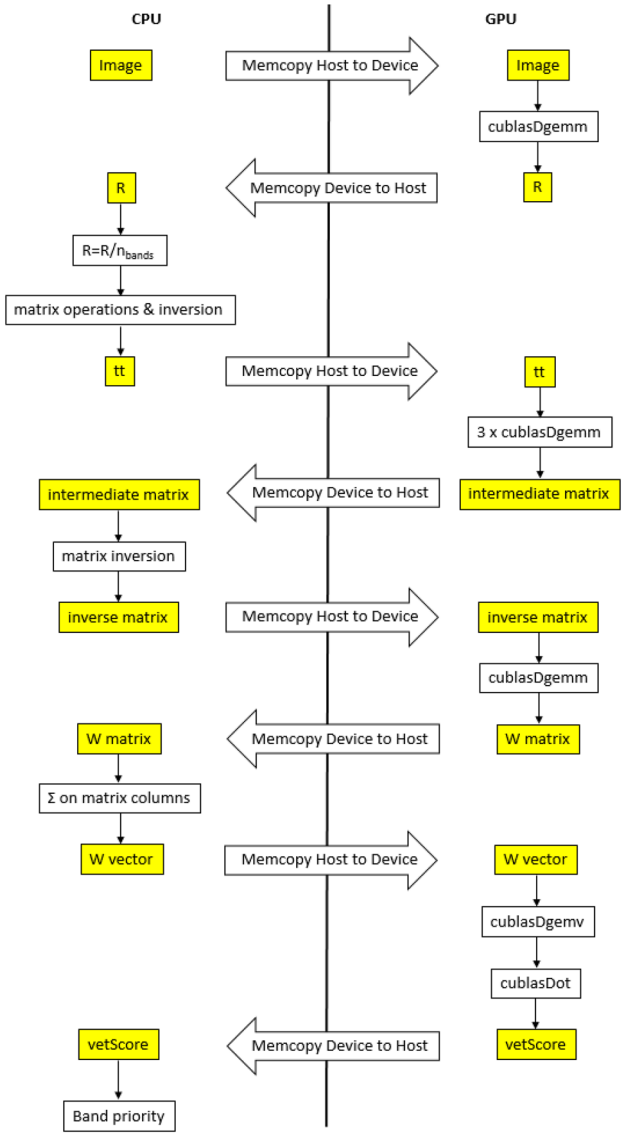


Fig. 10 LCMV-BDM algorithm for the entire matrix. Operations shown in the dotted box are performed for every band in the hyperspectral image. Data and operations are in yellow and white boxes, respectively

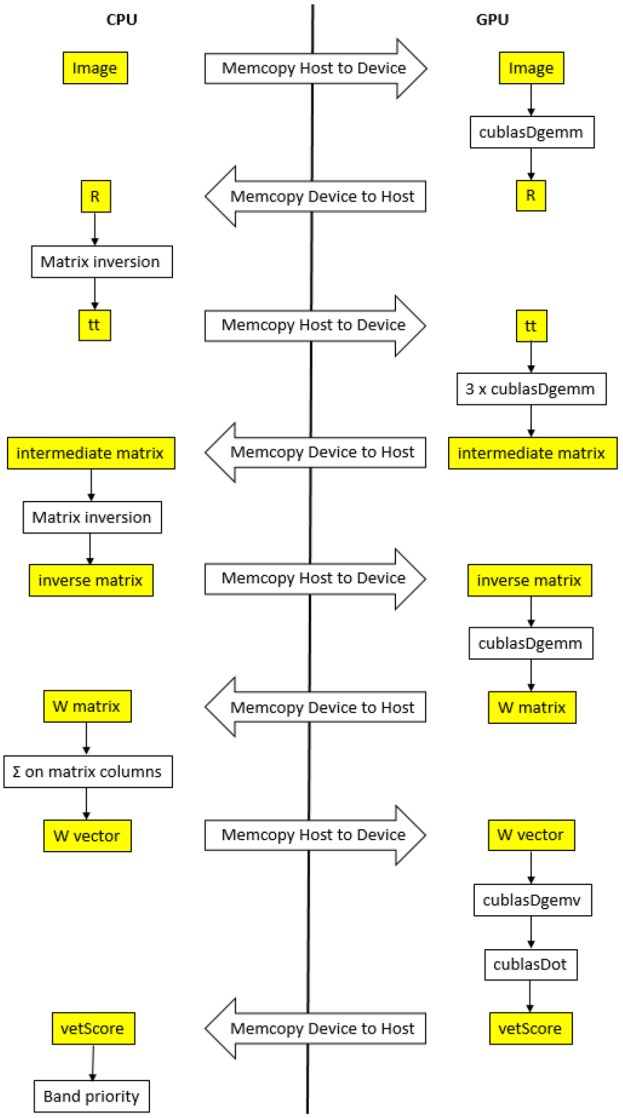


Fig. 11 LCMV-BCM algorithm for the entire matrix. Operations shown in the dotted box are performed for every band in the hyperspectral image. Data and operations are in yellow and white boxes, respectively

Center, World Trade Center, Cuprite mining district, Salinas Valley and Indian Pines (hereinafter identified as KSC, WTC, Cuprite, Salinas and Indian Pines, respectively). The EO-1 hyperspectral image has been acquired over Okavango delta, Botswana (hereinafter identified as Botswana). Other two real hyperspectral images were obtained by joining multiple Indian Pines hyperspectral images to increase dimensionality (hereinafter identified as Indian Pines_EW and Indian Pines_NS).

In Table 2, all the characteristic of the images used for the testing phase are reported. In particular, the total number of pixels and bands is shown for each image, both synthetic and real. Through this dataset, it has been verified that the bands selected by each algorithm were the same in all the different versions.

All the tests have been performed on a PC equipped with an Intel i7 3740 processor (4 physical cores), 8 GB of RAM and a NVIDIA GPU Tesla K40 which has 2880 CUDA cores running at 875 MHz and 12 GB of DDR5 RAM.

Each CUDA version of the different algorithms is based on CUDA environment 8.0, and it has been developed through Microsoft Visual Studio 2015 (64 bit Microsoft Windows 10).

For each algorithm, the execution times are reported in Fig. 12 using a logarithmic scale obtained by the results shown in Tables 3, 4 and 5. In this case, the results have

been reported only for real hyperspectral images. Indeed, they are more interesting than synthetic ones since they describe the algorithm behavior for real applications. Moreover, synthetic images have the same number of bands and an increasing number of pixels. On the other hand, the real dataset is made up of hyperspectral images with varying number of bands and pixels, allowing better algorithms analysis.

Before analyzing the obtained results, it is important to highlight a relevant characteristic of the BP algorithms. Usually, those kinds of techniques constitute a part of a more complex analysis chain executed on the hyperspectral image and optimally satisfying real-time constraints. Some examples are pollution monitoring, fire and biohazard detection [18–20]. In particular, the real-time constraints depend on the hyperspectral sensor that has been used to acquire the scene. In this work, a duration of 5 s has been considered as the threshold which determines whether a BP algorithm is executed in real time or not. That value has been chosen by considering the scanning characteristics of the NASA AVIRIS sensor, which acquires a hyperspectral image of 614 lines with 512 samples and 224 spectral bands every 5 s [21]. Thus, adopting the double precision representation, the sensor generates an image cube whose dimension is approximately 500 MB.

By analyzing the result of the SNR-based algorithm, reported in Fig. 12(a), it is possible to notice that the CUDA version works better than the other ones for each tested hyperspectral image. In particular, considering hyperspectral image size close to 500 MB, the CUDA version is able to satisfy the real-time constraint, whereas this does not happen in the serial and OpenMP versions. Moreover, there is no linear relation between hyperspectral images sizes and execution times. In fact, the time spent on the evaluation of the correlation matrix depends on the input hyperspectral image geometry (number of bands, rows and columns), which changes as shown in Table 1. Furthermore, the number of bands influences the complexity of the remaining operations of the algorithm. In other words, hyperspectral images with a lower number of bands have a lower computational weight on the operations that follow the correlation matrix computation.

Concerning the remaining algorithms based on statistical techniques, none of them is heavy from the computational point of view. Indeed, as Fig. 12(b–d) show, all the versions of the entropy-based, ID-based and variance-based algorithms have an execution time below 1 s for all the hyperspectral images (with the exceptions of ID-based algorithm serial implementation for the two biggest hyperspectral images). In these cases, the CUDA version is not always able to provide the best performance and this is due to different reasons. In the entropy-based and ID-based algorithms, the evaluation of the image histogram has a high impact on the execution performance. This part of the

Table 2 Dataset characteristics

Name	Pixel number	Rows	Cols	Bands	Size (MB)
Synthetic images					
p15N2k	2000	50	40	224	1.70
p15N10k	10,000	100	100	224	8.54
p15N20k	20,000	100	200	224	17.09
p15N50k	50,000	500	100	224	42.72
p20N75k	75,000	750	100	224	64.09
p20N100k	100,000	200	500	224	85.45
p20N200k	200,000	500	400	224	170.90
p25N500k	500,000	500	1000	224	427.25
Real images					
Hydice	4096	64	64	169	2.64
Cuprite	47,750	250	191	188	34.24
WTC	314,368	512	614	224	268.63
KSC	314,368	512	614	176	211.06
Indian Pines	21,025	145	145	220	17.64
Salinas	111,104	512	217	224	94.94
PaviaC	1024,000	2000	512	102	398.44
PaviaU	207,400	610	340	103	81.49
Botswana	377,856	1476	256	145	209.00
Indian Pines_EW	1134,672	1848	614	220	952.25
Indian Pines_NS	1644,292	2678	614	220	1379.94

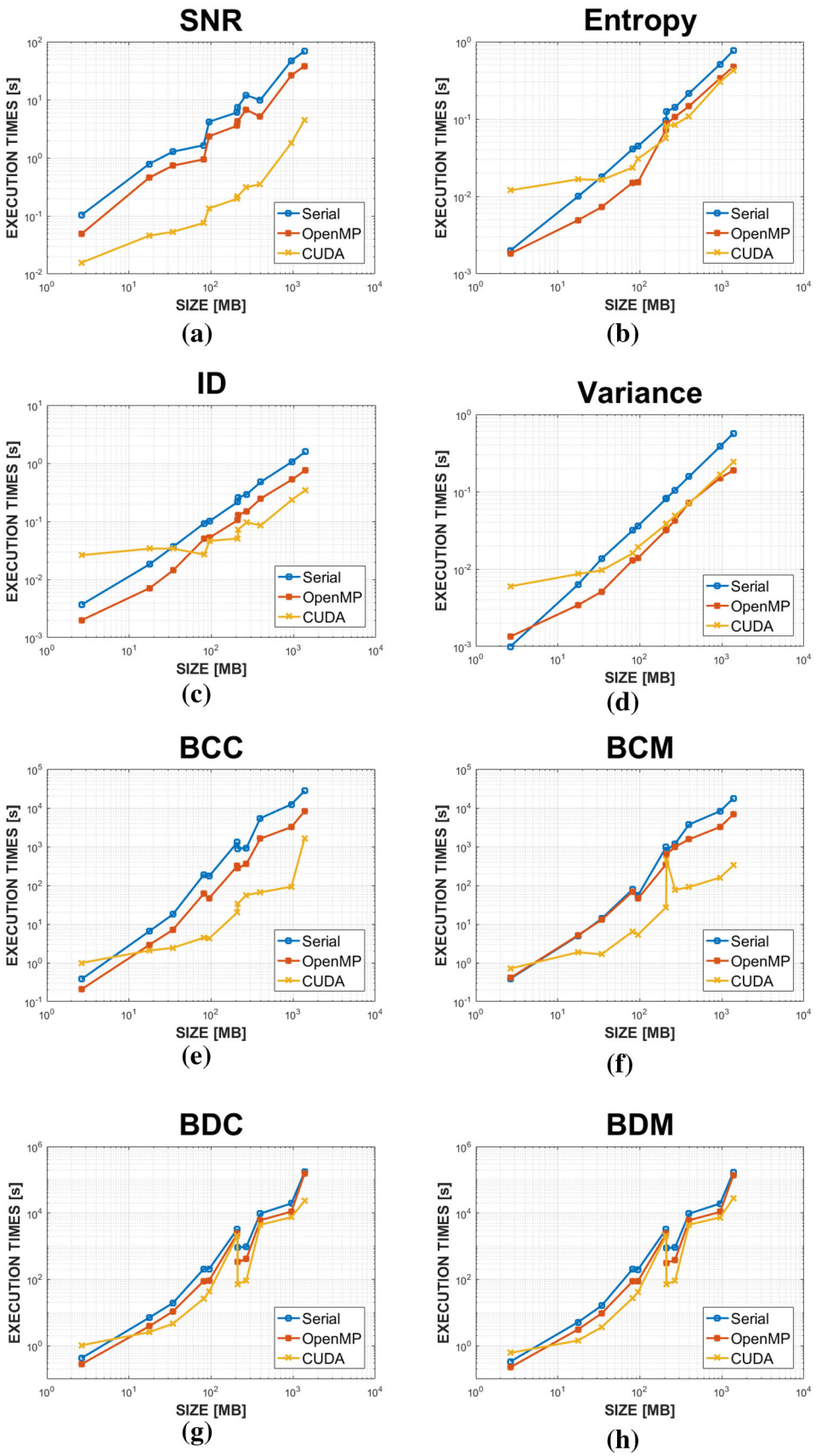


Fig. 12 Execution times for: **a** SNR, **b** entropy, **c** ID, **d** variance, **e** BCC, **f** BCM, **g** BDC, **h** BDM

Table 3 Processing times of the serial algorithms expressed in seconds

Images	SNR	Entropy	ID	Variance	BCC	BCM	BDC	BDM
Hydice	0.101	0.002	0.010	0.001	0.379	0.391	0.425	0.334
Indian Pines	0.835	0.010	0.016	0.007	6.533	4.987	7.043	5.145
Cuprite	1.360	0.018	0.037	0.014	17.849	13.825	19.414	16.199
PaviaU	1.784	0.041	0.094	0.031	188.524	79.016	206.588	204.265
Salinas	4.504	0.044	0.098	0.037	172.278	55.124	207.769	196.806
Botswana	7.164	0.094	0.208	0.081	1293.205	970.177	3269.517	3242.152
KSC	8.718	0.127	0.250	0.082	895.205	802.097	930.891	878.256
WTC	14.077	0.139	0.281	0.105	913.907	1188.852	949.980	912.068
PaviaC	11.006	0.219	0.468	0.157	5401.517	3735.483	9564.221	9530.064
Indian Pines_EW	49.657	0.517	1.040	0.382	12,371.923	8263.084	19,455.534	19,274.487
Indian Pines_NS	73.644	0.777	1.534	0.561	27,525.149	17,679.541	171,693.610	170,740.420

Table 4 Processing times of the OpenMP algorithms expressed in seconds

Images	SNR	Entropy	ID	Variance	BCC	BCM	BDC	BDM
Hydice	0.054	0.002	0.001	0.001	0.207	0.409	0.280	0.225
Indian Pines	0.501	0.005	0.007	0.004	2.900	5.172	3.869	3.117
Cuprite	0.753	0.007	0.012	0.006	7.134	12.934	10.801	9.465
PaviaU	0.991	0.017	0.030	0.010	60.940	68.963	89.710	87.400
Salinas	2.457	0.015	0.029	0.014	46.929	46.027	93.595	89.379
Botswana	3.774	0.076	0.098	0.029	318.019	350.751	2404.952	2398.143
KSC	4.435	0.080	0.108	0.033	276.154	641.355	331.507	313.995
WTC	7.205	0.101	0.127	0.038	353.557	985.211	409.967	385.263
PaviaC	5.214	0.155	0.212	0.057	1613.255	1547.618	6061.804	6035.858
Indian Pines_EW	26.398	0.336	0.474	0.146	3195.122	3214.139	10,991.141	10,760.593
Indian Pines_NS	38.693	0.483	0.700	0.186	8292.284	6834.271	155,359.390	137,841.230

Table 5 Processing times of the CUDA algorithms expressed in seconds

Images	SNR	Entropy	ID	Variance	BCC	BCM	BDC	BDM
Hydice	0.017	0.011	0.015	0.007	0.999	0.713	1.036	0.612
Indian Pines	0.047	0.017	0.016	0.010	2.080	1.883	2.581	1.439
Cuprite	0.054	0.015	0.026	0.010	2.442	1.656	4.614	3.643
PaviaU	0.079	0.023	0.026	0.016	4.470	6.485	26.439	27.114
Salinas	0.138	0.031	0.036	0.019	4.280	5.364	43.094	41.150
Botswana	0.213	0.056	0.062	0.039	20.017	27.174	2074.890	2072.621
KSC	0.224	0.081	0.114	0.039	33.824	466.136	72.557	71.083
WTC	0.310	0.083	0.115	0.050	56.001	76.439	92.519	90.697
PaviaC	0.366	0.106	0.161	0.071	65.862	90.449	4424.833	4432.392
Indian Pines_EW	1.859	0.306	0.385	0.169	92.916	159.538	7450.677	7365.001
Indian Pines_NS	4.550	0.435	0.578	0.244	1652.312	329.341	23,472.677	27,880.005

algorithm is strictly dependent on the input hyperspectral image since it determines the number of concurrent accesses in the memory.

This characteristic and the method used to find the histogram penalizes the execution of the overall algorithms.

On the other hand, the poor performance of the CUDA version of the variance-based algorithm is due to the low complexity of the algorithm. Indeed, memory transfers from CPU to GPU have a higher impact on the performance than the operations required for the evaluation of the variance even if the stream technique has been used. This phenomenon happens also in the entropy-based algorithm, but has a lower impact on the execution times since the involved computations are more complex compared to the ones involved in the variance algorithm.

By analyzing Fig. 12e–h, different comments arise. First of all, none of the LCMV-based algorithms is real-time compliant. However, an efficient implementation has been provided for each algorithm and, except for the smallest image, the CUDA implementations perform better than the other ones. In particular, the CUDA version of the BCC-based and BCM-based algorithms is able to obtain better performance than the BDC-based and BDM-based techniques. This is due to the fact that the former algorithms require, for each band, to evaluate every time the sample band image dependence matrix which increases the computational complexity of the algorithms. The last comment is related to the algorithms dependence on the hyperspectral image geometry, which prevents a direct proportionality between the execution time and the image size. This property can be seen in the BDC-based, BCM-based and BDM-based algorithms and is highlighted by the peak of the execution times in the hyperspectral image whose size is nearly 200 MB (Botswana), while this is not observed in the BCC-based algorithm. A final comment concerns the different algorithm used for the sample band correlation matrix and the sample band image dependence matrix inversions. Different precisions are needed in order to compute matrices which are nearly singular. Thus, the strategy adopted changes with the input image and this leads to a nonlinear relation, as shown by the charts.

6 Conclusions

In this paper, several parallel implementations of different band prioritization (BP) algorithms have been presented. All of them have been tested through a wide hyperspectral image database containing both synthetic and real scenes. The results allow to subdivide the proposed techniques in different categories, obtaining a complete suite of BP parallelized algorithms among which the scientist can choose on the basis of the specific application implemented. In particular, there are some algorithms such as the ones based on ID, entropy and variance, which can satisfy real-time constraints even with a serial implementation. On the other hand, concerning the SNR-based technique, only the GPU version is able to satisfy the real-time constraints for each considered image. Finally, it has been shown that all the LCMV-based algorithms are heavy from a computational viewpoint. Indeed, none of them real-time compliant, however, efficient GPU parallel implementations have been proposed, that significantly reduce execution times.

Table 6 Algorithm categories

Category	Algorithms
Real-time serial	ID, entropy, variance
Real-time parallel	SNR
Non-real-time	LCMV group

To the best of authors' knowledge, this paper presents the first GPU implementation of the considered BP algorithms in the literature. Other works regarding band selection can be found in [22–26], but they concern different types of procedures. For this reason, it is difficult and unfair to make a direct comparison. However, the SNR-based, variance-based, ID-based and entropy-based techniques are able to perform the band selection procedure in reduced times and on bigger hyperspectral images compared to [22] and [24]. This is not the case for the LCMV-based algorithms since they are computationally heavier than the techniques proposed in [22, 24].

This lead to subdivide the presented algorithms into three distinct categories, as shown in Table 6.

A final comment regards the experimental results reported in [22–26]. In all these works, the tests have been conducted only on few images while, in this work, a large hyperspectral database has been used to derive a precise characterization of each algorithm in terms of computational complexity.

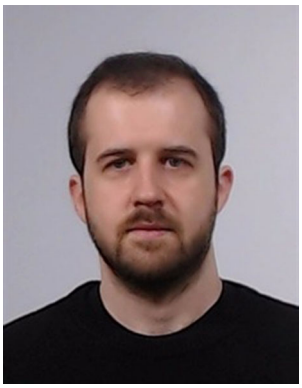
In the future, all the proposed algorithms will be inserted into a hyperspectral unmixing chain as well as a pre-processing phase for classification algorithms. Moreover, an FPGA-based implementation will be developed for the LCMV-based techniques, which do not meet real-time constraints even using GPUs.

Acknowledgements The authors gratefully thank NVIDIA Corporation for the donation of the GPU Tesla K40 used for this research.

References

1. Chang, C.-I., Wang, Su: Constrained band selection for hyperspectral imagery. IEEE Trans. Geosci. Remote Sens. **44**(6), 1575–1585 (2006)
2. Mausel, P.W., Kramber, W.J., Lee, J.K.: Optimum band selection for supervised classification of multispectral data. Photogramm. Eng. Remote Sens. **56**(1), 55–60 (1990)

3. Stearns, S.D., Wilson, B.E., Peterson, J.R.: Dimensionality reduction by optimal band selection for pixel classification of hyperspectral imagery. In: Applications of Digital Image Processing XVI, SPIE, vol. 2028, pp. 118–127 (1993)
4. Chang, C.-I., Du, Q., Sun, T.S., Althouse, M.L.G.: A joint band prioritization and band decorrelation approach to band selection for hyperspectral image classification. IEEE Trans. Geosci. Remote Sens. **37**(6), 2631–2641 (1999)
5. Gong, M., Zhang, M., Yuan, Y.: Unsupervised band selection based on evolutionary multiobjective optimization for hyperspectral images. IEEE Trans. Geosci. Remote Sens. **54**(1), 544–557 (2016)
6. Sun, K., Geng, X., Ji, L., Lu, Y.: A new band selection method for hyperspectral image based on data quality. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **7**(6), 2697–2703 (2014)
7. Jia, S., Tang, G., Zhu, J., Li, Q.: A novel ranking-based clustering approach for hyperspectral band selection. IEEE Trans. Geosci. Remote Sens. **54**(1), 88–102 (2016)
8. Wang, S., Chang, C.-I.: Band prioritization for hyperspectral imagery. In: Proceedings of SPIE 6302, Imaging Spectrometry XI, 63020I, https://doi.org/10.1117/12.681658 (2006)
9. Petaccia, G., Leporati, F., Torti, E.: OpenMP and CUDA simulations of Sella Zerbino Dam break on unstructured grids. Comput. Geosci. **20**(5), 1123–1132 (2016)
10. Torti, E., Acquistapace, M., Danese, G., Leporati, F., Plaza, A.: Real-time identification of hyperspectral subspaces. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **7**(6), 2680–2687 (2014)
11. Barberis, A., Danese, G., Leporati, F., Plaza, A., Torti, E.: Real-time implementation of the vertex component analysis algorithm on GPUs. IEEE Geosci. Remote Sens. Lett. **10**(2), 251–255 (2013)
12. Torti, E., Danese, G., Leporati, F., Plaza, A.: A hybrid CPU–GPU real-time hyperspectral unmixing chain. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **9**(2), 945–951 (2016)
13. Bernabé, S., Botella, G., Martín, G., Prieto-Matias, M., Plaza, A.: Parallel implementation of a full hyperspectral unmixing chain using OpenCL. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **10**(6), 2452–2461 (2017)
14. Wu, Z., Shi, L., Li, J., Wang, Q., Sun, L., Wei, Z., Plaza, J., Plaza, A.: GPU parallel implementation of spatially adaptive hyperspectral image classification. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **PP**(99), 1–13 (2017)
15. NVIDIA Corp.: NVIDIA Kepler GK110 architecture whitepaper. https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf. Accessed Feb 2017
16. Nascimento, J.M.P., Bioucas-Dias, J.M.: Hyperspectral subspace identification. IEEE Trans. Geosci. Remote Sens. **46**(8), 1445–2435 (2008)
17. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C: The Art of Scientific Computing, 2nd edn. Cambridge University Press, Cambridge (2007)
18. Torti, E., Fontanella, A., Plaza, A.: Parallel real-time virtual dimensionality estimation for hyperspectral images. J. Real-Time Image Proc. (2017). https://doi.org/10.1007/s11554-017-0703-6
19. Sánchez, S., Plaza, A.: Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs. J. Real-Time Image Proc. **9**(3), 397–405 (2014)
20. Rossi, A., Acito, N., Diani, M., Corsini, G.: RX architectures for real-time anomaly detection in hyperspectral images. J. Real-Time Image Proc. **9**(3), 503–517 (2014)
21. Green, R.O., Eastwood, M.L., Sarture, C.M., Chrien, T.G., Aronsson, M., Chippendale, B.J., Faust, J.A., Pavri, B.E., Chovit, C.J., Solis, M., Olah, M.R., Williams, O.: Imaging spectroscopy and the airborne visible/infrared imaging spectrometer. Remote Sens. Environ. **65**(3), 227–248 (1998)
22. Yang, H., Du, Q.: Fast band selection for hyperspectral imagery. In: 2011 IEEE 17th international conference on parallel and distributed systems, Tainan, pp. 1048–1051 (2011)
23. Zheng, J., Zhao, L., Li, X., Zhou, X., Li, J.: GPU-based acceleration of the hyperspectral band selection by SNR estimation using wavelet transform. In: Proceedings of SPIE 9263, multispectral, hyperspectral, and ultraspectral remote sensing technology, techniques and applications V (2014)
24. Yang, H., Du, Q., Chen, G.: Unsupervised hyperspectral band selection using graphics processing units. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **4**(3), 660–668 (2011)
25. Wei, W., Du, Q., Younan, N.H.: Fast supervised hyperspectral band selection using graphics processing unit. J. Appl. Remote Sens. **6**(1), 061504 (2012). https://doi.org/10.1117/1.jrs.6.061504
26. Chang, Y.L., Fang, J.P., Benediktsson, J.A., Chang, L., Ren, H., Chen, K.S.: Band selection for hyperspectral images based on parallel particle swarm optimization schemes. In: 2009 IEEE international geoscience and remote sensing symposium, Cape Town, pp. V-84–V-87 (2009)



Alessandro Fontanella was born in Pavia, Italy, in 1991. He received the Bachelor's degree in Computer Science Engineering and Master's degree in Computer Science Engineering (cum laude) from the University of Pavia, Pavia, Italy, in 2013 and 2015, respectively, where he is currently pursuing the Ph.D. degree in Computer Science Engineering. His research is focused on high-performance architectures for real-time image processing.

Elisa Marenzi received the B.S. degree and the M.S. degree both in Biomedical Engineering at the University of Pavia, in 2007 and 2010, respectively. She obtained the Ph.D. in Bioengineering and Bioinformatics from the University of Pavia in 2014. She is currently working as a post-doc research fellow in the Custom Computing and Programmable Systems Laboratory at the University of Pavia. Her research interests concern GPU parallel computing and the design and development of electronic and embedded systems, in particular for biomedical applications. In February 2012, she obtained the second place in the Best Student Paper contest organized by the IEEE Sensors Applications Symposium (SAS) 2012. In June 2012, she won two awards: third place in an academic competition promoted by the local industrial association, and she won a national contest for young researchers an start-uppers regarding the theme of social innovation and technology, the Lifebility Award 2012, promoted by Milan's Lion Club. In 2014, she obtained an award for her Ph.D. thesis from the Rotary and the National Association for Automatic Computing.

Emanuele Torti (M'13) was born in Voghera, Italy, in 1987. He received the Ph.D. degree in Electronics and Computer Science Engineering from University of Pavia, Pavia, Italy, in 2014. He received the Bachelor's Degree in Electronic Engineering and Master's Degree in Computer Science Engineering (cum laude) from University of Pavia in 2009 and 2011, respectively. He is a post-doc researcher at the Engineering Faculty of the University of Pavia. His research is focused on high performance architectures for real-time image processing and signal elaboration.

Giovanni Danese (M'00) received the Ph.D. degree in Electronics and Computer Engineering from the University of Pavia, Pavia, Italy, in 1987. He is a Full Professor with the Computer Programming and Computer Architecture in the Engineering Faculty at the University of Pavia. His current research interests include parallel computing, computerized instrumentation special-purpose computers, and signal and image processing.

Antonio Plaza (SM'07) was born in Caceres, Spain, in March 1975. He received the Computer Engineer degree in 1997, the M.Sc. degree in 1999, and the Ph.D. degree in 2002, all in Computer Engineering. Dr. Plaza is currently an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp). Prof. Plaza is an Associate Editor for IEEE Access and the IEEE Geoscience and Remote Sensing Magazine, and was a member of the Editorial Board of the IEEE Geoscience and Remote Sensing Newsletter (2011–2012) and a member of the steering committee of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (2012). Currently, he is serving as the Editor-in-Chief of the IEEE Transactions on Geoscience and Remote Sensing journal (since January 2013).

Francesco Leporati (M'14) received the Ph.D. degree in Electronics and Computer Engineering from the University of Pavia, Pavia, Italy, in 1993. He is an Associate Professor with the Industrial Informatics and Embedded Systems and Digital Systems Design in the Engineering Faculty at the University of Pavia. His current research interests include automotive applications, FPGA and application-specific processors, embedded real-time systems, and computational physics. Prof. Leporati is a member of the Euromicro Society and Associate Editor of Microprocessors and Microsystems.