Efficient Implementation of Hyperspectral Unmixing Methods

Seminário Apresentado no Âmbito das

Provas de Agregação em Informática

José M. P. Nascimento

May 15, 2023

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Hyperspectral remote sensing exploits the fact that all substances scatter electromagnetic energy, at specific wavelengths, depending on their molecular composition. Hyperspectral sensors have been developed to sample the scattered portion of the electromagnetic spectrum collecting hundreds even thousands of spectral bands at different wavelengths of the same area on the Earth surface [1]. For instance, the NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) [2] is able to record the visible and near-infrared spectrum of the reflected light of an area of several square kilometers (depending on the duration of the flight) using 224 spectral bands, which generates large data volumes comprising several GBs per flight. This high spectral resolution can be used for object detection and for discrimination between different objects based on their spectral characteristics [3].

Analyzing the spectral information of hyperspectral images has allowed the development of many applications in the fields of agriculture [4], surveillance [5], medical imaging [6, 7], food safety [8], forensic applications [9, 10], target detection for security/militar purposes, hazard prevention, and monitoring oil spills among others [1, 11]. In most of those applications, one of the requirements of paramount importance is the ability to give real-time or near real-time response [12].

Although, these sensors, that have been incorporated in satellite missions, allow to capture large three dimensional data cubes [2], the available downlink bandwidth to ground stations is limited, which brings prohibitive delays that endanger the real-time or near real-time requirements of such applications.

In recent years, graphics processing units (GPUs) have evolved into

highly parallel and programmable systems [13]. Specifically, several hyperspectral imaging algorithms have shown to be able to benefit from this hardware taking advantage of the extremely high floating-point processing performance, compact size, huge memory bandwidth, and relatively low cost of these units, which make them appealing for onboard data processing [14]. These processing systems can overcome the delays between hyperspectral image acquisition and its interpretation.

On the other hand, the power consumption requirements of this hardware makes them ineffective for onboard applications. Fortunately, over the last years, the advances in semiconductor industry and the huge interest on developing mobile devices have allowed companies such as Nvidia to develop low power GPUs like the Jetson AGX or Jetson Orin, which are a low power consumption GPUs, that nevertheless, can achieve high throughput in image processing applications at the same time. Additionally, these onboard systems could also be flexible in order to be adaptable to the needs of different missions.

Field-programmable gate arrays (FPGA) also represent a good choice to achieve the above mentioned requirements [15,16]. In recent years, scientific community have proposed several FPGA implementations of hyperspectral processing techniques, namely for unmixing [17–19], endmember extraction [18, 20, 21], abundance estimation [22], target detection [23], dimensionality reduction [24], image compression [25], among others.

Although hyperspectral sensors have high spectral resolution, due to technological reasons, hyperspectral images are limited by their relatively low spatial resolution [26]. In the case of AVIRIS sensor, a pixel cover an area of approximately 20 meters diameter on the ground. This means that several spectrally pure signatures (*endmembers*) are combined into the same mixed pixel. As a result, spectral unmixing is a very important task for hyperspectral data exploitation since the spectral signatures collected in natural environments are invariably a mixture of the pure signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument.

This document presents the implementation of two unmixing methods on GPU and FPGA platforms. The remain of the document is organized as follows, chapter 2 presents the hyperspectral imaging concept and the models to represent the datasets, chapter 3 present two unmixing methods, chapter 4 presents the methods implementation on GPU hardware and their results and chapter 5 presents the design of one method for an FPGA implementation and its results. Chapter 6 draws some lines of future research lines.

.

Chapter 2

Background

Hyperspectral imaging is an emerging and fast growing area in remote sensing [27]. The main characteristic of hyperspectral images is the high resolution they present in the spectral domain, since they are collected by instruments able to measure hundreds of narrow spectral bands corresponding to continuous wavelength channels [28]. The very high spectral resolution of hyperspectral data offers very significant potential in the identification of materials and their properties which has opened ground-breaking perspectives in several applications. Figure 2.1 illustrates the concept of hyperspectral imagery in Earth surface remote sensing application. Hyperspectral image consists of a 3D (three dimensional) datacube containing both spectral (one dimension) and spatial (two dimensions) information.

Although, there have been significant improvements in hyperspectral sensors, the spectral signatures collected in natural environments are invariably a mixture of the signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument [28]. These pixels, called mixed pixels, occur mainly due to the low spatial resolution of such images, but they can also result when distinct materials are combined into a homogeneous or intimate mixture [30].

Spectral unmixing consist on the identification of the spectrally pure signatures of the materials present in the scene (called endmembers in hyperspectral imaging terminology) and the estimation of the fractional abundances for each endmember, or in other words, the contribution of each endmember on each mixed pixel. In order to perform the spectral unmixing process, one of the most simple and widely used approaches for characterize mixed pixels in hyperspectral imagery is the Linear Mixture Model (LMM) [28].

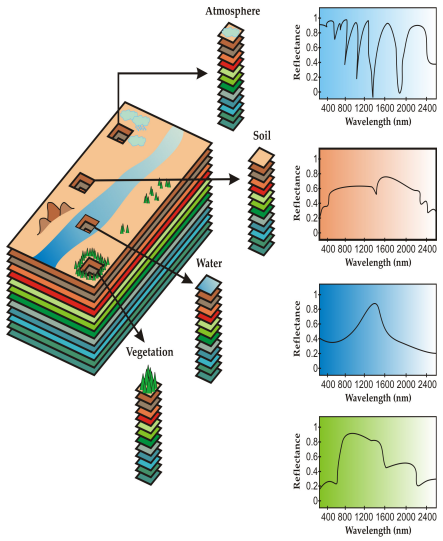


Figure 2.1: Illustration of the hyperspectral imaging concept (courtesy of the authors of work [29]).

The LMM considers that a mixed pixel is a linear combination of endmember signatures weighted by the correspondent abundance fractions, *i.e.*, at each pixel abundances represent the percentage of each endmember that is present in the pixel. Let us assume $\mathbf{Y} \equiv [\mathbf{y}_1, \ldots, \mathbf{y}_n] \in \mathbb{R}^{l \times n}$ denotes a matrix holding the n observed spectral vectors with l different spectral bands. This matrix can be modelled as:

$$\mathbf{Y} = \mathbf{MS} + \mathbf{N} \tag{2.1}$$

where $\mathbf{M} \equiv [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_p] \in \mathbb{R}^{l \times p}$ is a matrix containing p endmember signatures, $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_n] \in \mathbb{R}^{p \times n}$ is a matrix containing the abundance fractions for each of the p endmembers in $\mathbf{M}$, and $\mathbf{N}$ is a noise term. Notice, however, that this matrix organizations does not take into account the spatial information contained in the dataset. Due to the nature of the acquisition process, at a given pixel, abundance fractions sum to one and are nonnegative, the so-called abundance sum constraint (ASC) and abundance nonnegativity constraint (ANC). This abundance fraction are in the following $p - 1$ probability simplex:

$$\{\mathbf{S} \in \mathbb{R}^{p \times n} : \mathbf{S} \succeq 0\,,\, \mathbf{1}_p^T \mathbf{S} = \mathbf{1}_n^T\}, \tag{2.2}$$

where $\mathbf{S} \succeq 0$ means $\mathbf{s}_{ij} \geq 0$, for $i = 1, \ldots, p$, and $j = 1, \ldots, n$, $\mathbf{1}_p$ and $\mathbf{1}_n$ denote a $p \times 1$ and $n \times 1$ column vectors filled of 1's, and $(\cdot)^T$ denotes the transpose operator. Thus, under the linear mixing model the observed spectral vectors in a given scene are in a simplex whose vertices correspond to the endmembers. Assuming that each endmember has at least one pure pixel on the dataset, *i.e.*, one pixel containing only one endmember, the endmember extraction aims to identify those pure pixels.

The LMM assumes negligible interaction between distinct endmembers, which is strictly valid when the endmembers are arranged side-by-side, as in a chequerboard. Fig. 2.2 schematizes the LMM scenario. It has been widely exploited and it has been demonstrated in numerous applications that it is a useful technique in hyperspectral remote sensing [31–34]. There are, however, many situations, involving multiple light scattering effects, in which the linear mixing model is not a good approximation. In these cases, the nonlinear models may provide better assessment of endmember signatures and abundances, improving the unmixing accuracy [30, 35].

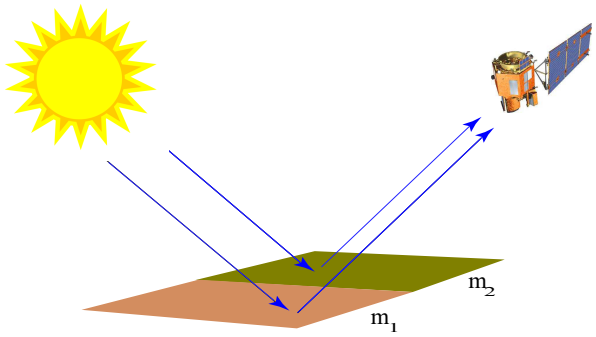
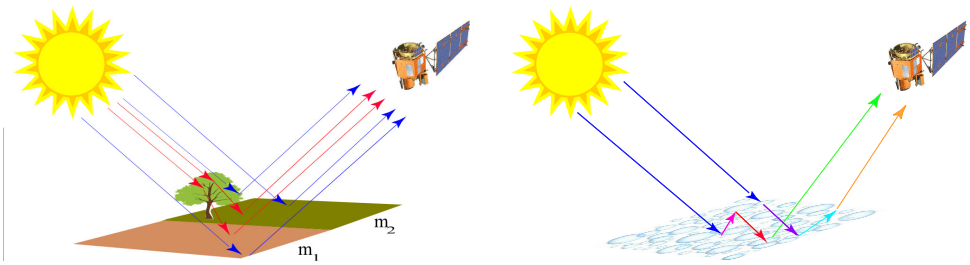


Figure 2.2: Illustration of the linear mixture model.

A general nonlinear mixture model can be expressed as follows:

$$\mathbf{y}_i = f(\mathbf{M}, \mathbf{s}_i) + \mathbf{n}_i, \tag{2.3}$$

where f is a nonlinear function and $\mathbf{n}_i \in \mathbb{R}^l$ is a noise perturbation associated to the i-th pixel vector. In the recent past, several methods have been proposed in the literature to estimate the function f. Depending on how they estimate the function f is it possible to do a distinction between physics-based methods, and data-driven techniques that do not require any underlying physical assumptions on f. This two major categories may be divided into sub-categories depending on the different techniques used to



(a) Multi-layered mixtures (b) Intimate mixtures

Figure 2.3: Illustration of different nonlinear scenarios (a) Multi-layered mixtures (b) Intimate mixtures.

estimate f such as bilinear models [35–37], models for intimate mineral mixtures [38, 39], radiosity-based approaches [40], ray tracing [41], neural networks [42], kernel methods [43], support vectors machine (SVM) techniques [44], manifold learning methods [45], piecewise linear techniques [46], multilinear model (MLM) [47–49] among others. A full review of many different nonlinear techniques may be found in [35].

However typically there are two distinct situations where the multiple scattering effects can be observed [50]:

- Multiple scattering effects on complex vegetated surfaces [35,37]. This model consist on a multilayered scene model, where there are multiple interactions among the scatterers at each layer, to study the nonlinear effects among different materials. Fig. 2.3(a) illustrates this model, where it is assumed that incident solar radiation is scattered by the scene through multiple bounces involving several endmembers.

- Materials are intimately mixed [42]. Fig. 2.3(b), schematizes this scenario, where the mixing scale is microscopic. Although, this scenario is more complex than the linear model, in [51] it is proposed the use of a nonlinear mixture model based on Hapkes bidirectional spectroscopy theory [39] to improve the unmixing results in intimate mixtures of soil.

For both scenarios unmixing methods were developed and can be found on works [36, 50, 52].

2.1 Hyperspectral Linear Unmixing

Hyperspectral unmixing amounts to identify the set of endmembers present in the dataset and their abundance fractions at each pixel. Methods for hyperspectral linear unmixing can be classified as geometrical, statistical, and deep learning (DL)-based methods [53].

Geometrical unmixing algorithms work under the assumption that the endmembers of an HSI are the vertices of a simplex with the minimum volume enclosing the dataset or of a simplex with the maximum volume contained in the convex hull of the dataset. Pure pixel-based and minimum volume (MV)-based methods belong to this category. The pure pixel-based algorithms assume that there is one pure pixel at least per endmember, *i.e.*, there is at least one spectral vector on each vertex of the data simplex [1]. Some popular and efficient algorithms taking this assumption are *vertex component analysis* (VCA) [31], the *automated morphological endmember extraction* (AMEE) [32], the *pixel purity index* (PPI) [54], the N-FINDR [55], the *successive volume maximization* (SVMAX) [56],the *automatic target generation process* (ATGP) [57], and the *Negative Abundance-Oriented* (NABO) [58].

If the pure pixel assumption is not fulfilled, which is a more realistic scenario, the unmixing process is a rather challenging task, since some (or even all) endmembers are not in the dataset. Some methods that have been developed to cope these scenarios are, the *simplex identification via split augmented Lagrangian* (SISAL) [59], the *robust minimum volume enclosing simplex* (RMVES) [56], the *minimum volume transform-nonnegative matrix factorization* (MVC-NMF) [34], the *sparsity-promoting iterative constrained endmembers* (ICE) [60], and the *robust and recursive non-negative matrix factorization* (RRNMF) [61].

Statistical methods focus on using parameter estimation techniques to determine endmember and abundance parameters, very often the problem is formulated under the Bayesian framework [33, 62–64]. Although these methods are robust, they are computational very heavy and time expensive.

Recently, some deep learning-based unmixing methods have emerged. For instance, the model-inspired neural networks [65], that is based on the LMM and the iterative shrinkage thresholding algorithm, the LMM-based end-to-end deep neural network with sparsity-constrained nonnegative matrix factorization [66], and the unsupervised nonlinear spectral unmixing method based on a CNN autoencoder network is presented in [67]. A review of deep learning techniques applied to hyperspectral unmixing can be found in [68]. Nevertheless, there are still several drawbacks [69]. For instance,

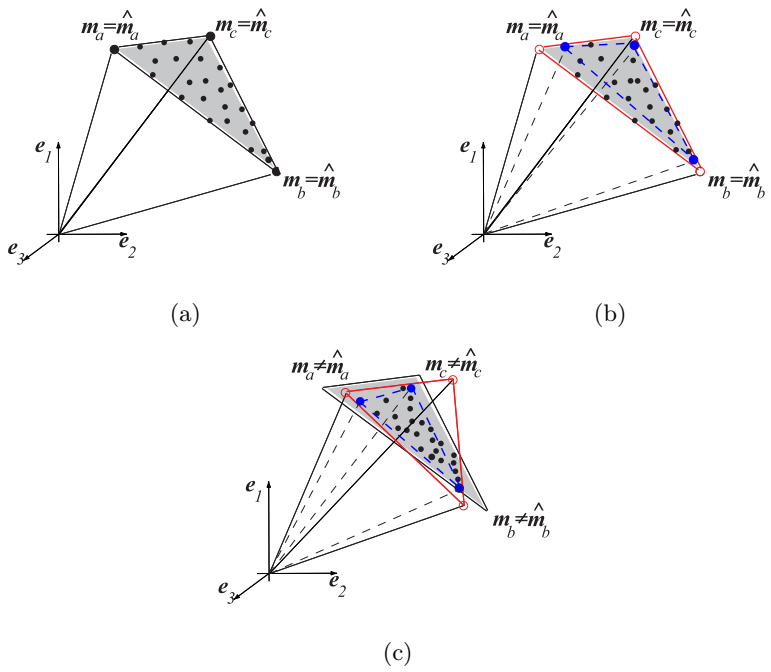


Figure 2.4: Illustration of tree linear mixture scenarios: (a): with pure pixels (solid line - estimated simplex by all methods); (b): without pure pixels and with pixels in the facets (solid red line - estimated simplex based on minimum volume; dashed blue line - estimated simplex by pure-pixel based methods); (c): highly mixed pixels (solid red line - estimated simplex based on minimum volume; dashed blue line - estimated simplex by pure-pixel based methods).

current approaches often require a lot of training samples and network parameters to achieve satisfactory unmixing performance.

Fig. 2.4 illustrates three datasets raising different degrees of difficulties in what unmixing is concerned: the dataset shown in Fig.2.4(a) contains pure pixels, *i.e.*, the spectra corresponding to the simplex vertices are in the dataset. The dataset shown in Fig.2.4(b) does not contain pure pixels, at least for some endmembers. This is a much more challenging, usually attacked with the minimum volume based methods, note that pure-pixels based methods are outperformed under these circumstances; Fig.2.4(c), contains a highly mixed dataset where only statistical and deep learning-based methods can give accurate unmixing results.

Chapter 3

Unmixing Methods

Herein two well-known methods are presented, *Vertex Component Analysis* (VCA) [31] and *simplex identification via split augmented Lagrangian* (SISAL) [59]. These methods belong to the geometrical unmixing methods class, where VCA assumes that there are pure pixels in the dataset and SISAL works without this assumption.

3.1 VCA Unmixing Method

VCA is a unmixing method proposed by Nascimento and Bioucas [31] and it is one the most popular method within the literature. VCA is an unsupervised method to extract endmembers from hyperspectral datasets and is based on the geometry of convex sets. It exploits two facts: 1) the endmembers are the vertices of a simplex and 2) the affine transformation of a simplex is also a simplex. VCA is a fully automatic algorithm and it works with or without Dimensionality Reduction (DR) pre-processing step. The algorithm iteratively projects data onto a direction orthogonal to the subspace spanned by the endmembers already determined. The new endmember signature corresponds to the pixel with largest projection. The algorithm repeats the procedure until the whole set of p endmembers is found. Figure 3.1 illustrates the VCA method working on a simplex defined by a mixture of three endmembers. In the first iteration, data is projected onto the first direction $\mathbf{f}_1$. The extreme of the projection corresponds to endmember $\mathbf{m}_a$. In the next iteration, endmember $\mathbf{m}_b$ is found by projecting data onto direction $\mathbf{f}_2$, which is orthogonal to $\mathbf{m}_a$. Finally, a new direction $\mathbf{f}_3$, orthogonal to the subspace spanned by $\mathbf{m}_a$ and $\mathbf{m}_b$ is generated and the endmember $\mathbf{m}_c$ is found by seeking the extreme of the projection of the dataset onto $\mathbf{f}_3$.

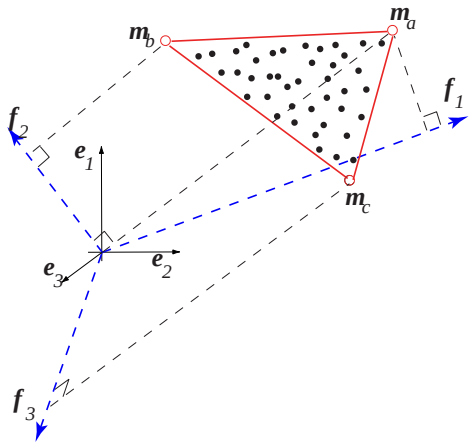


Figure 3.1: Hyperspectral mixture of three endmembers illustrating the VCA method.

Algorithm 1, shows the main steps of the VCA, without the DR step. Although the number of endmembers is much lower than the number of bands ($p \ll l$) and, thus, it is advantageous, in terms of signal-to-noise ratio (SNR) to represent the spectral vectors in a signal subspace basis [70], this step can take a larger time consumption, which could not be compatible with real-time requirements. More details on the full VCA algorithm can be found in [31]. In Algorithm 1, notation $(\cdot)^{\#}$ stands for the pseudoinverse matrix, symbol $\mathbf{V}$ denotes the estimated mixing matrix and $[\mathbf{V}]_{:,j}$ stands for jth column of $\mathbf{V}$.

Algorithm 1 : VCA

1: $\mathbf{V} := [\mathbf{e}_u \,|\, \mathbf{0} \,|\, \ldots \,|\, \mathbf{0}]$; {$\mathbf{e}_u := [0, \ldots, 0, 1]^T$ and $\mathbf{V}$ is a $l \times p$ auxiliary matrix}
2: **for** $i := 1$ to p **do**
3: $\quad \mathbf{w} := \text{randn}\,(0, \mathbf{I})$; { $\mathbf{w}$ is a zero-mean random Gaussian vector}
4: $\quad \mathbf{f} := (\mathbf{I} - \mathbf{V}\mathbf{V}^{\#})\mathbf{w}$; {$\mathbf{f}$ is a vector orthogonal to the subspace spanned by $\mathbf{V}_{:,1:i}$}
5: $\quad \mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
6: $\quad k := \arg\max_{j=1,\ldots,n} |[\mathbf{v}]_{:,j}|$;
7: $\quad [\mathbf{V}]_{:,i} := [\mathbf{Y}]_{:,k}$;
8: **end for**

3.1.1 Dimensionality Reduction Methods

Under the linear observation model, spectral vectors are in a subspace of dimension p. Usually the number of endmembers is much lower than the

number of bands ($p \ll l$) and, has mentioned before, it is worthy to work on the signal subspace. This leads to significant savings in computational complexity and to *signal-to-noise ratio* (SNR) improvements.

Nascimento and Bioucas method, called *hyperspectral signal identification by minimum error* (HySime) [70], estimates the number of endmembers present in the dataset and selects the subset of eigenvalues that best represents the signal subspace in the least squared error sense. The results of using a HySime before dataset unmixing is further illustrated in this section.

Estimating the number of endmembers p is regarded as the first step of the overall endmember estimation task. The number of endmembers is often unavailable in realist scenarios. Particularly, information theory-based algorithms, eigenvalue thresholding algorithms, and geometry characterization algorithms are three main families of techniques for estimating the number of endmembers. A brief summary of each family can be found in [1,71] Some examples of methods that have been proposed to estimate the number of endmembers, are, NWHFC [72], HySime [70], and *Second moment linear dimensionality* (SML) [73]. The *robust signal subspace estimation* (RSSE) [74] have been proposed in order to estimate the signal subspace in the presence of rare signal pixels, thus it can be used as a preprocessing step for small target detection applications. *Sparsity promoting ICE* (SPICE) [60] is an extension of ICE algorithm that incorporates sparsity-promoting priors to find the correct number of endmembers. The framework presented in [75] also estimates the number of endmembers when it unmix the data.

Dimensionality reduction aims to find the subspace that best represent the signal in a lower dimension. This can be done after knowing the number of endmembers present in the scene or done at the same time. Several approaches have been published, for instance band selection or band extraction, as the name suggests, exploits the high correlation existing between adjacent bands to select a few spectral components among those with higher SNR. Projection techniques seek for the best subspace to project data by minimizing an objective function. Among these methods are, *principal component analysis* (PCA) [76] which is the best data representation in the least squares sense; *singular value decomposition* (SVD) [77] provides the projection that best represents data in the maximum power sense; *maximum noise fraction* (MNF) [78] seek for the projection that optimizes the ratio of noise power to signal power; HySime [70], selects the subset of eigenvalues that best represents the signal subspace in the least squared error sense; and HFC [72] which is a Neyman-Pearson detection theory-based thresholding method to determine the *virtual dimensionality* (VD).

For illustration purposes, a simulated scene was generated according to

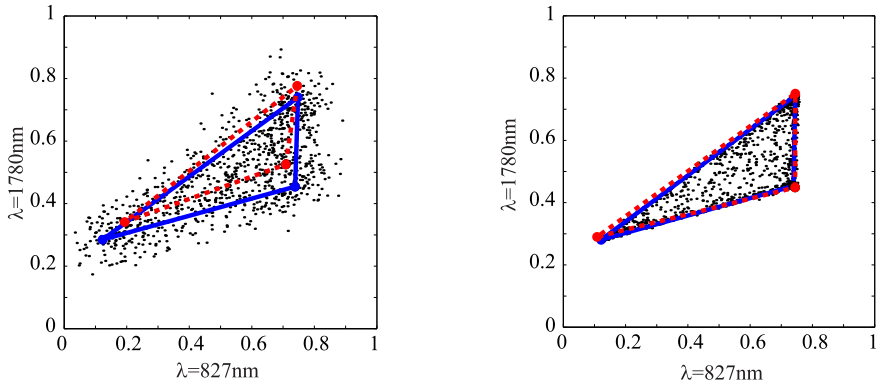


Figure 3.2: Scatter-plot (bands $\lambda = 827nm$ and $\lambda = 1780nm$) of the three endmembers mixture. (a) Unprojected data. (b) Projected data using SVD. Solid and dashed lines represent, respectively, simplexes computed from original and estimated endmembers (using VCA).

the linear mixture model. Three spectral signatures were selected from the U.S. geological survey (USGS) digital spectral library [79], the abundance fractions follow a Dirichlet distribution in order to respect the nonegativity and full additivity constraints [33], and the additive noise is zero-mean white Gaussian, where the SNR is set to 20 dB. Fig. 3.2(a) presents a scatter-plot of the simulated spectral mixtures without projection. It is also plotted two triangles whose vertices represent the true endmembers (solid line) and the estimated endmembers (dashed line) by the VCA algorithm, respectively. Fig. 3.2(b) presents a scatter-plot (same bands) of projected data onto the estimated affine set inferred by SVD. Noise is clearly reduced, leading to a visible improvement on the VCA results.

3.2 SISAL Unmixing Method

SISAL is a unmixing method proposed by Bioucas [59] and it belongs to the minimum volume class unmixing methods. SISAL is able to unmix hyperspectral datasets in which the pure pixel assumption is violated. Moreover, on SISAL method, the positivity hard constraints are replaced by hinge type soft constraints, whose strength is controlled by a regularization parameter. This replacement has three advantages: 1) robustness to outliers and noise;

2) robustness to poor initialization; 3) opens the door to dealing with large problems. The non-convex optimization problem is solved as a sequence of non-smooth convex sub-problems using variable splitting to obtain a constraint formulation, and then applying an augmented Lagrangian technique.

SISAL method assumes that the number of endmembers and signal subspace is known before hand [70] and that the observed vectors $\mathbf{y}_i$, for $i = 1, \ldots, n$, are projected to a p-dimensional basis signal subspace.

Let $\mathbf{E}_p$ be a matrix, with orthonormal columns, spanning the signal subspace. Thus

$$\begin{aligned} \mathbf{X} &\equiv \mathbf{E}_p^T \mathbf{Y} \\ \mathbf{X} &= \mathbf{E}_p^T \mathbf{M}\mathbf{S} + \mathbf{E}_p^T \mathbf{N} \\ &= \mathbf{A}\mathbf{S} + \mathbf{N}^* \end{aligned} \tag{3.1}$$

where $\mathbf{X} \equiv [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{p \times n}$ denote a matrix holding the projected spectral vectors, $\mathbf{A} = \mathbf{E}_p^T \mathbf{M}$ is a $p \times p$ square mixing matrix, and $\mathbf{N}^*$ accounts for the projected noise. The input data of this method are the regularization parameter in the problem λ, the augmented Lagrange regularization parameter τ, the quadratic regularization parameter μ, the reduced matrix $\mathbf{X}$ which size is $p \times n$, and the initial simplex, a $p \times p$ matrix, performed by VCA using the reduced matrix $\mathbf{X}$.

Linear unmixing amounts to infer matrices $\mathbf{A}$ and $\mathbf{S}$. This can be achieved by fitting a minimum volume simplex to the dataset [80]. Since the volume defined by the columns of $\mathbf{A}$ is proportional to $|det\mathbf{A}|$, then finding a minimum volume matrix $\mathbf{A}$ subject to the ANC and ASC, leads to the non-convex optimization problem

$$\begin{aligned} \widehat{\mathbf{A}} &= \arg\min_A |\det \mathbf{A}| \\ \text{s.t. } &: \mathbf{Q}\mathbf{X} \succeq \mathbf{0}\,,\ \mathbf{1}_p^T \mathbf{Q}\mathbf{X} = \mathbf{1}_n^T, \end{aligned} \tag{3.2}$$

where $\mathbf{Q} \equiv \mathbf{A}^{-1}$. Since $|\det \mathbf{Q}| = 1/|\det \mathbf{A}|$, problem (3.2) can be replaced by

$$\begin{aligned} \widehat{\mathbf{Q}} &= \arg\min_Q \{-\log |\det \mathbf{Q}|\} \\ \text{s.t. } &: \mathbf{Q}\mathbf{X} \succeq \mathbf{0}\,,\ \mathbf{1}_p^T \mathbf{Q}\mathbf{X} = \mathbf{1}_n^T. \end{aligned} \tag{3.3}$$

Considering that matrix $\mathbf{Q}$ is symmetric and positive-definite, problem (3.3) is convex. However, in most of practical scenarios matrix $\mathbf{Q}$ is not symmetric nor positive-definite, thus the minimization of expression (3.3) is not convex.

Aiming to give a sub-optimal solution of (3.3), a sequence of augmented Lagrangian optimizations is applied (see [59] for details).

Instead of solving problem (3.3), a modified version is proposed:

$$\begin{aligned} \widehat{\mathbf{Q}} &= \arg\min_{Q} \{-\log|\det \mathbf{Q}| + \lambda\|\mathbf{QX}\|_h\} \\ \text{s.t. } &: \mathbf{1}_p^T\mathbf{Q} = \mathbf{a}^T, \end{aligned} \tag{3.4}$$

where $\|\mathbf{QX}\|_h \equiv \sum_{ij} h(\mathbf{QX})$, $h(x) \equiv \max(-x, 0)$ is the so-called hinge function, λ is the regularization parameter, and $\mathbf{a}^T = \mathbf{1}_n^T\mathbf{X}^T\left(\mathbf{XX}^T\right)^{-1}$. Notice that $\|\mathbf{QX}\|_h$ penalizes the negative components of $\mathbf{QX}$ proportionally to their magnitude, thus playing the rule of a soft constraint or a regularizer, yielding solutions that are robust to outliers, noise, and poor initialization.

Chapter 4

GPU Methods Design and Implementation

In recent years, high-performance computing systems have become more widespread in remote sensing applications, namely the emergence of programmable graphics processing units (GPUs). Driven by the increasing demands of the videogame industry, GPUs have evolved from expensive application specific units into highly parallel and programmable systems. GPUs can be abstracted as an array of highly threaded streaming multiprocessors (SMs), where each multiprocessor is characterized by a single instruction multiple data (SIMD) architecture, i.e., in each clock cycle each processor executes the same instruction while operating on multiple data streams. Each SM has a number of streaming processors that share a control logic and instruction cache and have access to a local shared memory and to local cache memories in the multiprocessor, while the multiprocessors have access to the global GPU (device) memory. Fig. 4.1 presents a typical architecture and the data flow communication between CPU and GPU.

The algorithms are constructed by chaining the so-called *kernels* which operate on entire streams and which are executed by a multiprocessor, taking one or more streams as inputs and producing one or more streams as outputs. Thereby, data-level parallelism is exposed to hardware, and kernels can be concurrently applied without any sort of synchronization. The kernels can perform a kind of batch processing arranged in the form of a grid of blocks where each block is composed by a group of threads that share data efficiently through the shared local memory and synchronize their execution for coordinating accesses to memory. As a result, there are different levels of memory in the GPU for the thread, block, and grid concepts. There is

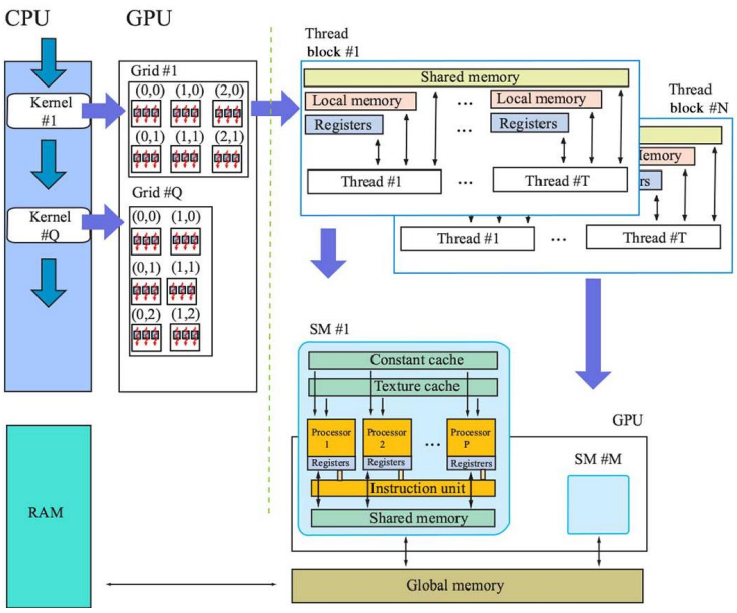


Figure 4.1: Typical *NVidia* GPU architecture, computation, and data transfer flow from/to CPU.

also a maximum number of threads that a block can contain (depending on the GPU model), however, the number of threads that can be concurrently executed is much larger due to the fact that several blocks executed by the same kernel can be managed concurrently. With the above ideas in mind, the next sections present the unmixing methods implementations and their fast optimizations [81].

4.1 VCA implementation on GPU

As mentioned in the previous section, VCA iteratively project all pixels to a direction orthogonal to the subspace spanned by the endmembers found so far. At each iteration the pixels projection are independent from each other, thus, it is highly desirable that this procedure be parallel implemented in order to increase in processing speed.

Algorithm 2, shows the main steps of the VCA, where symbol $\widehat{\mathbf{M}}$ denotes the estimated mixing matrix and $[\widehat{\mathbf{M}}]_{:,j}$ stands for the jth column of $\widehat{\mathbf{M}}$. Red and blue lines denote operations to be computed in CPU and GPU,

respectively. Briefly, the dataset is read to the memory, where is shared by the GPU cores, the direction determination, which is composed by small matrix operations, is made on the CPU, whereas the pixel projections are made in the GPU in parallel fashion.

Algorithm 2 : VCA Parallel Implementation

1: $\widehat{\mathbf{M}} := \mathbf{0}_p$;
2: **for** $i := 1$ to p **do**
3: $\quad \mathbf{V} := \text{orth}\ (\widehat{\mathbf{M}})$ {Orthogonal vectors that spans $\widehat{\mathbf{M}}$ range}
4: $\quad \mathbf{P} := (\mathbf{I} - \mathbf{V}\mathbf{V}^T)$
5: $\quad \mathbf{f} :=$ generate a vector from span($\mathbf{P}$)
6: $\quad \mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
7: $\quad k := \arg\max_{j=1,\ldots,n} |\mathbf{v}|$;
8: $\quad [\widehat{\mathbf{M}}]_{:,i} := [\mathbf{Y}]_{:,k}$;
9: **end for**

(Red and blue lines denote operations to be computed in CPU and GPU, respectively.)

4.1.1 Abundance Estimation Method

Has the VCA method extract the endmember signatures, the endmember's abundance estimation problem can be posed in the framework of convex optimization [82]. Under this context, the alternating direction method of multipliers (ADMM) [83] is a powerful algorithm that can be parallelized. SUNSAL method proposed by Bioucas and Figueiredo [82] is an ADMM based approach to estimate the abundance fractions under ASC and ANC. Assuming that matrix $\mathbf{M}$ is already obtained by the VCA algorithm, the abundance fraction estimation can be defined as

$$\begin{aligned} &\min_{\mathbf{s}} \frac{1}{2}\|\mathbf{Y} - \mathbf{M}\mathbf{S}\|_F^2 \\ &\text{subject to : } \mathbf{S} \succeq 0\,,\, \mathbf{1}_p^T\mathbf{S} = \mathbf{1}_n^T, \end{aligned} \tag{4.1}$$

where notation $\|(\cdot)\|_F$ stands for Frobenius norm. The above formulation is a particular case of the constrained $\ell_2 - \ell_1$ problems solved by SUNSAL, corresponding to the absence of the ℓ_1 term. The pseudo-code is presented in Algorithm 3.

Algorithm 3 : SUNSAL Parallel Implementation

1: choose $\mu > 0$, $\mathbf{U}_0$, and $\mathbf{D}_0$.
2: $\mathbf{A} := \mathbf{M}^T\mathbf{Y}$
3: $\mathbf{B} := \left(\mathbf{M}^T\mathbf{M} + \mu\mathbf{I}\right)^{-1}$
4: $\mathbf{c} := \mathbf{B}\mathbf{1}_p \left(\mathbf{1}_p^T\mathbf{B}\mathbf{1}_p\right)^{-1}$
5: $\mathbf{G} := \mathbf{B} - \mathbf{c}\mathbf{1}_p^T\mathbf{B}$
6: $k := 0$
7: **repeat**
8: $\quad\mathbf{R} := \mathbf{A} + \mu\left(\mathbf{U}_k + \mathbf{D}_k\right)$
9: $\quad\mathbf{S}_{k+1} := \mathbf{G}\mathbf{R} + \mathbf{c}\mathbf{1}_N^T$
10: $\quad\mathbf{V}_k := \mathbf{S}_{k+1} - \mathbf{D}_k$
11: $\quad\mathbf{U}_{k+1} := \max\{0, \mathbf{V}_k\}$
12: $\quad\mathbf{D}_{k+1} := \mathbf{D}_k - \left(\mathbf{S}_{k+1} - \mathbf{U}_{k+1}\right)$
13: $\quad k := k + 1$
14: **until** stopping criterion is satisfied
15: $\widehat{\mathbf{S}} = \mathbf{S}_k$

(Red and blue lines denote operations to be computed in CPU and GPU, respectively.)

4.2 VCA - Experimental Results on GPU

Nacimento *et al*, on work [84] have developed methods VCA and SUNSAL for GPU. These methods were analyzed in order to determine the most consuming parts that can be parallelized. Table 4.1 presents the processing time of each step of both algorithms for a CPU (Intel core i7-2600) as a function of the number of endmembers ($p = \{10, 20\}$) and the number of pixels ($n = \{10^5, 5 \times 10^5\}$). It should be noted that these processing times only depend on the number of pixels (n) and the number of endmembers (p), hence these can be generalized for any hyperspectral dataset. It is clear that the projection of all pixels onto direction $\mathbf{f}$ inside VCA loop Algorithm 2 (line 6), calculation of $\mathbf{A}$ in Algorithm 3 (line 2), and the operations inside Algorithm 3 loop (lines 8 - 12), are the most consuming parts of the method and they grow with p and n.

Assuming that VCA is applied after the projection of the dataset onto the signal subspace [70], the computational complexity of VCA is $2p^2n$ floating point operations, where the projection of each pixel of $\mathbf{Y}$ onto direction $\mathbf{f}_k$, can be done independently from the remaining pixels, thus, it can be parallelized. After the dataset is transferred to the global memory of GPU, using $4pn$ bytes, on each iteration the generation of the direction $\mathbf{f}_k$ is performed on the CPU and transferred to the constant memory of the device

Table 4.1: VCA and SUNSAL processing time (10^{-3} seconds) for a Intel core i7-2600 CPU as a function of the number of endmembers (p) and of the number of pixels (n).

		$n = 1 \times 10^5$		$n = 5 \times 10^5$	
	operation	$p = 10$	$p = 20$	$p = 10$	$p = 20$
VCA	**f** (lines 3-5)	0.09	0.57	0.09	0.57
	v (line 6)	535.16	445.68	2 677.62	2 193.38
	k (lines 7)	2.36	4.72	12.40	24.82
SUNSAL	**A** (line 2)	2 394.23	4 868.62	10 509.91	21 160.15
	B (line 3)	0.01	0.08	0.01	0.08
	c (line 4)	≈ 0.00	≈ 0.00	≈ 0.00	≈ 0.00
	G (line 5)	≈ 0.00	0.02	≈ 0.00	0.02
	R (line 8)	476.00	1 601.21	2 903.37	9 040.34
	S (line 9)	2 400.48	15 560.41	14 486.93	88 144.44
	V (lines 10)	169.72	572.17	1 034.20	3 210.11
	U (lines 11)	794.46	2 689.25	4 741.11	15 580.76
	D (lines 12)	337.27	1 127.14	2 044.35	6 414.02

($4p$ bytes). Then, a first kernel initially puts into execution many threads as the number of pixels vectors of the image (n) divided into blocks of 32 threads, so that each thread is responsible for computing the dot product of $\mathbf{f}_k^T$ by the pixel vector $\mathbf{Y}_i$, *i.e.*, one element of vector $\mathbf{v}$. Then, the second kernel determines the index of the maximum absolute value of vector $\mathbf{v}$, indicating the position of the endmember signature on the dataset. This task is performed using a binary reduction operation. These kernels correspond to instruction on line 7 of Algorithm 2 (blue lines). Fig. 4.2 presents a illustrative example of a single thread functioning. It is worth noting that to fully optimize the parallel algorithm, the size of $\mathbf{v}$ must be power of two (with zero padding, if necessary).

The proposed implementation of SUNSAL follows the same rule, *i.e.*, the most consuming time operations are developed in a parallel fashion to be processed in the GPU. The operations outside the loop, namely, the inversion of $p \times p$ matrix in line 3 of Algorithm 3 are implemented in the CPU since it has a low computational cost (red lines in Algorithm 3). Inside the SUNSAL loop the first kernel compute matrix $\mathbf{R}$ (see line 8 of Algorithm 3). This kernel launches as many threads as elements that are present in $\mathbf{R}$, where each thread computes an element of the $\mathbf{A} + \mu\,(\mathbf{U}_k + \mathbf{D}_k)$. The result

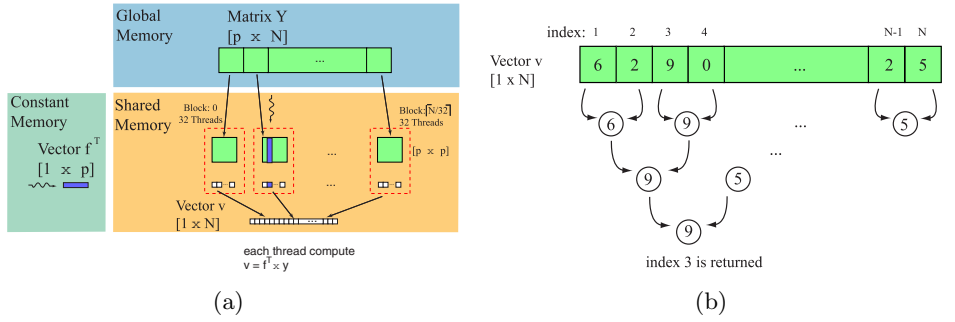


Figure 4.2: Illustration of parallel VCA in the GPU:(a) thread to compute $\mathbf{v} = \mathbf{f}^T\mathbf{Y}$; (b) thread to find the index of maximum absolute value of $\mathbf{v}$.

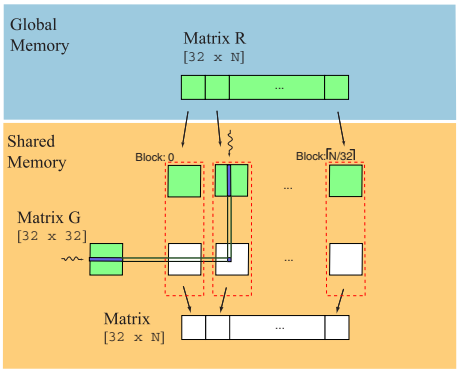


Figure 4.3: Illustration of parallel SUNSAL in the GPU: Thread to compute one element of $\mathbf{RG}$.

is stored in the global memory. The second kernel computes the abundance estimates, $\mathbf{S}$, on each iteration (line 9 of Algorithm 3), by first computing the product of matrices $\mathbf{G}$ and $\mathbf{R}$ followed by the addition of matrix $\mathbf{c1}_n^T$. In order to minimize the number of global memory accesses, matrix $\mathbf{R}$ is partitioned into sub-blocks of 32×32 elements, which is the size of the block, and transferred to the shared memory. Each block uses a total of 8 Kbytes of the shared memory. Fig 4.3 illustrates this procedure, where one can see that each thread is responsible to compute each element of $\mathbf{S}$. The kernels for update $\mathbf{V}$ and $\mathbf{D}$ follow the same strategy used on the first kernel, whereas $\mathbf{U}$ is updated by a kernel which analyze if each element of $\mathbf{V}$ is negative. It should be noted that matrices are stored in global memory which occupies around $28pn$ bytes.

4.2.1 Accuracy Evaluation

In this section, the proposed implementation accuracy is evaluated using a real hyperspectral data collected by the AVIRIS sensor. A subset of the Cuprite dataset [1] containing 350×350 pixels with 187 spectral bands (noisy and water absorption bands were removed) is considered. This site is well understood mineralogically, it has several exposed minerals of interest, including Alunite GDS83 Na63, Buddingtonite GDS85 D-206, Calcite WS272, Kaolinite CM9, and Muscovite GDS108, and it has been extensively used for remote sensing experiments over the past years and its geology was previously mapped in detail [85].

Table 4.2 shows the well-known evaluation metric, spectral angle distance (SAD), for five minerals of interest where the signatures estimated by the different methods are compared with the nearest laboratory spectra.

Table 4.2: Spectral angle scores (in degrees) between the USGS mineral spectra and their corresponding endmember pixel produced by our implementation for the AVIRIS Cuprite scene.

Alunite	**Buddingtonite**	**Calcite**	**Kaolinite**	**Muscovite**
5.15	5.71	7.36	2.24	4.90

4.2.2 Performance Evaluation

In this section, we apply the sequential and the parallel versions of the unmixing method based on VCA and SUNSAL for both simulated data and for the real dataset of Cuprite collected by the AVIRIS sensor.

In order to evaluate the performance of the proposed method in terms of processing time, the sequential version was implemented in C programming language running on a computer platform equipped with a quad core Intel i7-2600 CPU, 3.4GHz clock speed, 16 Gbyte memory, 1Mbyte and 8Mbyte of L2 and L3 cache memory. The parallel version was implemented in OpenCL programming language for three different GPU cards from $NVidia^{TM}$. Table 4.3 presents a summary of the characteristics of the three cards. The following subsections will present the performance results in terms of acceleration factors or speedups.

[1] Available online at http://aviris.jpl.nasa. gov/html/aviris.freedata.html

Table 4.3: characteristics of *NVidia* GPU cards used in the VCA tests.

	GTX 590	GTX 680	C2050
Cores	1024	1536	448
Clock (MHz)	607	1006	574
Memory (GB)	3.0	2.0	3.0
Bandwidth (GB/s)	163.9	192.2	144

Evaluation with Simulated Data

Several synthetic scenes are created with different number of pixels and different number of endmembers. Each pixel is generated according to expression (2.1), where spectral signatures are selected from the USGS digital spectral library, containing 224 spectral bands covering wavelengths from 0.38 to 2.5 μm with a spectral resolution of 10 nm. The abundance fractions are generated according to a Dirichlet distribution which enforces positivity and full additivity constraints (see [33] for details).

Table 4.4 illustrates data transfer from/to device time, processing time, and speedup factors for a dataset composed of 10^5 pixels with thirty endmembers ($p = 30$). The processing time on GPU is lower than the CPU time, due to the parallel processing implementation.

Fig. 4.4 shows the speedup of the parallel version (with regards to the sequential version) for three different GPU cards as a function of the number of endmembers and for $n = 5 \times 10^5$. For illustration purposes the results for the method running on all CPU cores is also presented. Herein, the method implemented in OpenCL is compiled for the quad core Intel processor. The speedup of GTX680 is higher than 100, for $p = 30$, which is quite remarkable taking into account that the sequential version has been carefully optimized. As expected the speedup grows with the number of endmembers. Note that GTX680 GPU card, which has more cores, has the best performance, and the 4 core CPU has a speedup smaller than 8, achieving always the worst result.

Evaluation with Real Data

The proposed method is applied to real hyperspectral data collected by the AVIRIS sensor. Table 4.5 shows the processing time (in seconds) for the CPU sequential versions of VCA and SUNSAL and also for the paralell versions using three different GPUs. One can note that the best speedup is higher than 100 times, achieved on the GTX680 card. These results are in

Table 4.4: Processing time and data transfer (seconds) for a Intel core i7-2600 CPU and for a GTX590 GPU card ($p = 30$ and $n = 10^5$).

	CPU	GTX 590
RAM $\rightarrow$ Global Mem.	-	0.014
VCA: **f** (lines 3 - 5)	0.002	
VCA loop (line 6 - 7)	1.479	0.008
RAM $\leftarrow$ Global Mem.	-	0.001
SUNSAL: compute **A** (line 2)	7.230	0.423
SUNSAL (lines 3 - 5)	0.036	
RAM $\rightarrow$ Global Mem.	-	0.221
SUNSAL loop (lines 8 - 12)	57.39	0.624
RAM $\leftarrow$ Global Mem.	-	0.028
Total Time	65.389	1.607
Speedup (CPU time / GTX590 time)	-	48.12

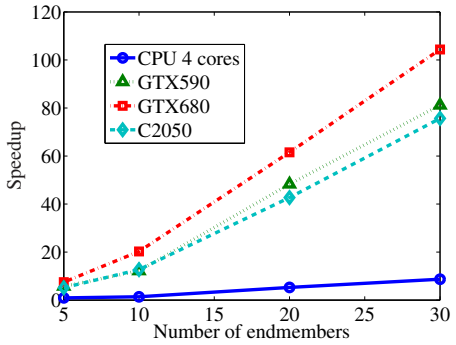


Figure 4.4: Speedup of parallel version as a function of the number of endmembers (p) for $n = 5 \times 10^5$.

Table 4.5: Processing time (seconds) for the Cuprite dataset.

	Total time	VCA	SUNSAL	Speedup
CPU	98.270	0.400	97.870	-
CPU (4 cores)	19.537	0.296	19.241	5.029
GTX590	1.169	0.021	1.148	84.06
GTX680	0.975	0.021	0.954	100.79
Tesla C2050	1.367	0.022	1.345	71.88

agreement with the ones for the simulated scenarios.

Finally, it is worth to mention that as the AVIRIS sensor is able to collect 512 hyperspectral pixels in 8.3 ms [86], thus, a 350×350 subimage takes nearly 2 seconds. Consequently, the proposed parallel method using GPUs is suitable for real-time hyperspectral unmixing systems.

4.3 SISAL Implementation on GPU

In this section, the GPU implementation of the SISAL algorithm, developed in Nascimento *el al* work [87] is described. This work is carried out using the compute unified device architecture (CUDA) developed by $NVidia^{TM}$.

SISAL implementation has been first optimized in the C version by reducing the number of operations and the size of the matrices. Since the original algorithm proposed in [59] is based on diagonal matrices in the range of $p^2 \times p^2$, herein the same calculations by using just the profitable data, in this way the algorithm works with $p \times p$ matrices instead of $p^2 \times p^2$. Algorithm 4 shows the parallel implementation of SISAL, the method was analyzed in order to determine the most consuming parts that can be parallelized. The part of the code that can be parallelized is limited by the inverse and determinant matrix calculations that can be performed faster in CPU, due to the small size of the $p \times p$ matrices. In addition, lines 16 and 25 were tested in both architectures and they show better results in CPU. Algorithm 4 shows in blue color the operations performed in the GPU. These operations have been implemented in order to optimize the memory access. Notice that $\mathbf{q}$ matrix is the only dataset updated in CPU memory because its dependency on the GPU code. The kernels provide memory coalescence by setting contiguous memory access to contiguous threads, in addition to use shared memory to store and sum the thread results in parallel. First, the $p \times p$ matrix $\widehat{\mathbf{Y}}$ is performed in GPU using a *CublasDgemm* multiplication to solve $\mathbf{Y}\mathbf{Y}^{\mathbf{T}}$. In line 7, a kernel called *computeA* calculates the vector $\mathbf{a}$ with p

elements. This launches p blocks with the maximum number of thread possible in the block. Each thread calculates a given number of matrix products (depending of number of threads) and sum the thread results in parallel, then the result of each block is multiplied by a position of $\mathbf{H}$ (p elements). Later, a kernel called *computeHinge* is in charge of executing line 14, which launches as many blocks as number of bands present in the reduced image, p, and uses the maximum number of thread possible in the block to perform the multiplication and evaluate the maximum values. On the other hand, the operations inside the *quadratic approximation loop* are performed in the GPU, a kernel called *computeSum* executes the sum of $\mathbf{b1} = (\mathbf{z} + \mathbf{d})\mathbf{Y}^T$ launching as many threads as elements of $(\mathbf{z} + \mathbf{d})$, then the result is multiplied by $\mathbf{Y}^T$ using the *CublasDgemm* multiplication. Later, a new kernel called *computeQ* is in charge of calculate $\mathbf{q} = \mathbf{G}(\mathbf{b1}\tau + \mathbf{b\_aux}) + \mathbf{a}$ using as many threads as elements in the resulting matrix $p \times p$. First, each thread sums each element of $(\mathbf{b1}\tau + \mathbf{b\_aux})$ and stores the results in shared memory, then multiplies the resulting matrix stored in shared memory by $\mathbf{G}$ and sums $\mathbf{a}$ to the results. The last two operations, lines 20 and 21 are performed by a *CublasDgemm* multiplication and a new kernel called *computeSoft*, this kernel launches as many threads as number of elements in data set $p \times n$, and updates the values of $\mathbf{z}$ and $\mathbf{d}$. In our implementation, parallel streams and dynamic parallelism do not improve the timing results because of the data dependence among the kernels, and dynamic parallelism also induces to repeat memory access and thread divergence.

4.4 SISAL - Experimental Results on GPU

The experiments have been carried out using both synthetic and real datasets. Three synthetic hyperspectral images were generated from spectral signatures, with 224 bands, randomly selected from the United States Geological Survey (USGS), and using the procedure described in work [88] to simulate natural spatial patterns and applying uniform signal-to-noise ratio with $SNR = 40$ dB: *SyntheticSmall* with 100×100 pixels, *SyntheticMedium* with 400×250 pixels and *SyntheticLarge* with 512×600 pixels. Each image has three different versions generated with different number of endmembers $p = 10$, $p = 20$ and $p = 30$, without including pure pixel in the scenes. In this way, the experiments evaluate the performance behaviour increasing both the number of pixels and the number of endmembers.

On the other hand, in order to illustrate the efficiency of our method over real data sets, the well-known AVIRIS Cuprite scene has been analyzed. In

Algorithm 4 : SISAL Parallel Implementation

1: Set $\lambda > 0$, $\mu > 0$, $\tau > 0$ and $\widehat{\mathbf{M}} = \text{VCA}(\mathbf{Y})$
2: $\mathbf{Q} = \widehat{\mathbf{M}}^{-1}; \mathbf{q} = vec(\mathbf{Q})$;
3: $\boxed{\widehat{\mathbf{Y}} = \mathbf{Y}\mathbf{Y}^{\mathbf{T}}}$
4: $\mathbf{IF} = (\mu\mathbf{I} + \tau\widehat{\mathbf{Y}})^{-1}$
5: $\mathbf{H} = (\mathbf{IF}\mathbf{1_p})\frac{1}{||\mathbf{IF}||_\mathbf{F}}$
6: $\widehat{\mathbf{IY}} = \widehat{\mathbf{Y}}^{-1}$
7: $\boxed{\mathbf{a} = \mathbf{H} \bigotimes ((\widehat{\mathbf{IY}}\mathbf{Y})\mathbf{1_n})}$ {Simbol $\bigotimes$ denotes the Hadamard product}
8: $\mathbf{G} = \mathbf{IF} - \mathbf{H}(\mathbf{1_p^T}\mathbf{IF})$
9: **for** $k = 1$ to Max_Iter **do**
10: $\quad \mathbf{g} = -vec(\mathbf{Q}^{-1})$ {$\mathbf{Q}$ is updated with $\mathbf{q}$}
11: $\quad \mathbf{b\_aux} = \mu\mathbf{q} - \mathbf{g}$
12: $\quad \mathbf{q}_k = \mathbf{q}$
13: $\quad$ **repeat**
14: $\quad\quad \boxed{\mathbf{hinge} = \lambda \max\{-\mathbf{qY}, 0\}}$
15: $\quad\quad \mathbf{val}_k = -\log(\text{abs}(\det(\mathbf{q}))) + \mathbf{hinge}$
16: $\quad\quad \mathbf{quad}_k = (\mathbf{q} - \mathbf{q}_k)^T\mathbf{g} + \frac{\mu}{2}(\mathbf{q} - \mathbf{q}_k)^T(\mathbf{q} - \mathbf{q}_k) + \mathbf{hinge}$
17: $\quad\quad$ **for** $t := 1$ to AL_Iter **do** {Quadratic approximation}
18: $\quad\quad\quad \boxed{\mathbf{b} = \mathbf{b\_aux} + \tau((\mathbf{z} + \mathbf{d})\mathbf{Y}^T)}$
19: $\quad\quad\quad \boxed{\mathbf{q} = \mathbf{Gb} + \mathbf{a}}$
20: $\quad\quad\quad \boxed{\mathbf{z} = soft(\mathbf{qY} - \mathbf{d}, \lambda/\tau)}$
21: $\quad\quad\quad \boxed{\mathbf{d} = \mathbf{d} - (\mathbf{qY} - \mathbf{z})}$
22: $\quad\quad$ **end for**
23: $\quad\quad \boxed{\mathbf{hinge} = \lambda \max\{-\mathbf{qY}, 0\}}$
24: $\quad\quad \mathbf{val}_{k+1} = -\log(\text{abs}(\det(\mathbf{q}))) + \mathbf{hinge}$
25: $\quad\quad \mathbf{quad}_{k+1} = (\mathbf{q} - \mathbf{q}_k)^T\mathbf{g} + \frac{\mu}{2}(\mathbf{q} - \mathbf{q}_k)^T(\mathbf{q} - \mathbf{q}_k) + \mathbf{hinge}$
26: $\quad\quad$ **if** $\mathbf{quad}_k \geqslant \mathbf{quad}_{k+1}$ **then**
27: $\quad\quad\quad$ **repeat**
28: $\quad\quad\quad\quad \mathbf{q} = (\mathbf{q} + \mathbf{q}_k)/2$
29: $\quad\quad\quad\quad \boxed{\mathbf{hinge} = \lambda \max\{-\mathbf{qY}, 0\}}$
30: $\quad\quad\quad\quad \mathbf{val}_{k+1} = -\log(\text{abs}(\det(\mathbf{q}))) + \mathbf{hinge}$
31: $\quad\quad\quad$ **until** $\mathbf{val}_k \geqslant \mathbf{val}_{k+1}$
32: $\quad\quad$ **end if**
33: $\quad$ **until** $\mathbf{val}_k \geqslant \mathbf{val}_{k+1}$ AND $\mathbf{quad}_k \geqslant \mathbf{quad}_{k+1}$
34: **end for**

(blue lines denote operations performed in the GPU, respectively.)

Table 4.6: Spectral angle scores (in degrees) between the USGS mineral spectra and their corresponding endmember pixel produced by our implementation for the AVIRIS Cuprite scene.

Alunite	**Buddingtonite**	**Calcite**	**Kaolinite**	**Muscovite**
4.89	7.13	10.59	7.16	5.64

the following, the accuracy and performance results of the proposed parallel method are analyzed.

4.4.1 Accuracy Evaluation

In order to analyze the accuracy of the parallel implementation, the well-known spectral angle distance (SAD) is adopted. All the synthetic images involved in the experiments were made up using the same matrix of $p = 30$ (generated randomly from the USGS spectral library) and applying $SNR = 40dB$. The parameters of the algorithm were set to $Max\_Iter = 60$, $AL\_Iter = 10$, $10^{-2} \leq \lambda \leq 10^{-4}$, $\tau = p \times 1000/n$, and $\mu = 10^{-6}$, which show very accurate results over all the images. The SAD was performed between the extracted endmembers and ground-truth spectral signatures used to generate the image. In all the cases, the results present very low average SAD scores under 9 degrees in the worst case, which indicates that the implementation provides accurate results, since the best case of SAD is 0 degrees and the worst case is 90 degrees.

On the other hand, Table 4.6 shows the SAD scores performed between the endmembers extracted from AVIRIS Cuprite dataset and the spectral signatures of the USGS spectral library. The SAD scores are in the range of 4 to 11, which demonstrates the accuracy of the proposed implementation over real datasets.

Before describing the results, it is important to emphasize that our GPU version provides exactly the same results as the serial version of the algorithms, implemented in C programming language using the gcc (gnu compiler default) with optimization flag -O3. In order to keep the method's accuracy, the algorithm is implemented using double precision.

4.4.2 Performance Evaluation

In order to evaluate the performance of the proposed method in terms of processing time, our implementations were executed on two GPU platforms. First, a quad-core Intel i7-4790 CPU, 3.6-GHz clock speed and 32-GB RAM

Table 4.7: characteristics of Nvidia GPU cards used in the SISAL tests.

	GTX 980	TITAN X
Cores	2048	3072
Clock (MHz)	1126	1000
Memory (GB)	4.0	12.0
Bandwidth (GB/s)	224.0	336.6

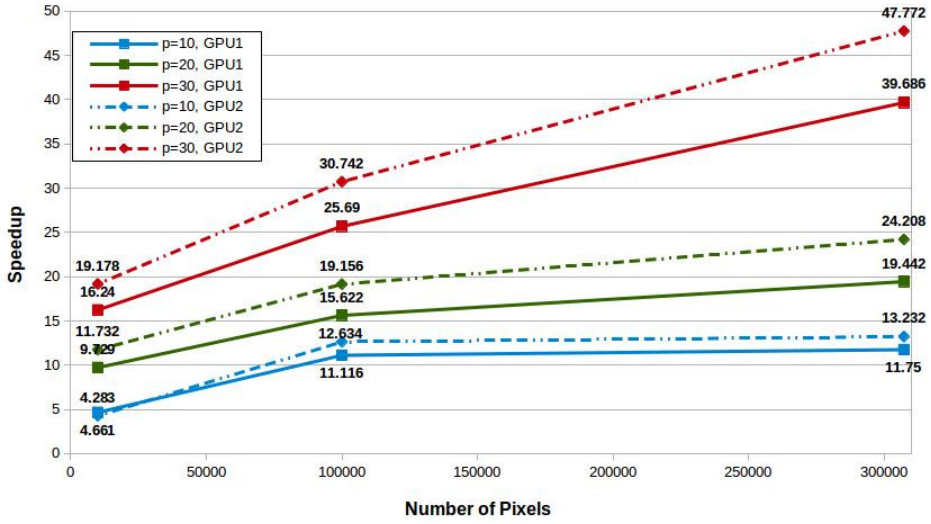


Figure 4.5: Speedup of the parallel version as a function of the number of pixels and number of endmembers (p) over the synthetic datasets.

memory connected to a GPU *NVidia*^{TM} GeForce GTX 980 (denoted by *GPU1*). Second, a quad-core Intel i7-4790K CPU, 4.0-GHz clock speed and 32-GB RAM memory connected to a GPU *NVidia*^{TM} GeForce TITAN X (denoted by *GPU2*). The sequential implementation, in C programming language, was executed in one core available of the multi-core Intel i7-4790K. For each experiment, ten runs were performed and the mean values were reported with minor standard deviations. Table 4.7 presents a summary of the characteristics of the cards.

Evaluation with Simulated Data

This section presents the results over three synthetic hyperspectral images. Fig. 4.5 displays the results based on the experiments conducted using

the three synthetic hyperspectral datasets (described above) with different number of endmembers.

These results indicate that the GPU implementation can significantly accelerate the execution of SISAL over big datasets, when compared with the sequential version which has been carefully optimized for one CPU core. As expected, the speedups factor is higher as the number of endmembers in the image increases.

All of the results include the memory transfers overhead among CPU and GPU; those transfers are synchronous since the kernel invocation is iterative. Times for VCA endmember initial estimation and dimensionality reduction step are not included in our experiments since these algorithms are aside from the original SISAL method.

Fig. 4.5 shows that the best results are achieved on *GPU2* with speedup higher than 47 times. The *GPU2* clock frequency is slower but it has more CUDA cores than *GPU1*.

In order to widely analyze the GPU implementation performance, Table 4.8 displays the resulting times of the different parts of the code, measured using the $NVidia^{TM}$ profile tool, after processing the synthetic datasets with 30 endmembers on the considered GPU device. This table specifies the memory transfers overhead among CPU and GPU, the non-parallelized execution time, the total times for each kernel (described in Section 4.3), the *Cublas* multiplications times and finally the speedups achieved over the synthetic datasets. Note that each kernel is launched several times during the different iterations that the algorithm takes to converge.

Table 4.8 shows that, in all the cases, the memory transfers among CPU and GPU take a tiny percentage of the execution time, thus the GPU spent most of the time in data processing which is an outstanding result. In addition, one can also note that the times spent executing the non-parallelized code are very small comparing to the parallel execution time. Therefore, the most time consuming parts of the algorithm have been parallelized. Furthermore, the scores for each particular kernel reveals that all of them are positively contributing to reach the encouraging speedups.

Table 4.9 presents the GPU occupancy achieved by the different kernels for the experiments conducted using the synthetic hyperspectral datasets with 30 endmembers. The scores reveal that the GPU implementation properly takes advantage of the GPU architecture in all the experiments. Note, however, that although the *ComputeQ* kernel works with small matrices and shows a low use of the GPU, the main reason to keep this kernel is to avoid memory transfers between CPU and GPU.

Table 4.8: Processing times (in seconds) and Speedups achieved for the GPU implementation of the algorithm on the CPU-GPU platform, tested with the four synthetic datasets ($p = 30$).

	Synthetic_1 100 × 100		**Synthetic_2** 400 × 250		**Synthetic_3** 512 × 600		**Synthetic_4** 600 × 1024	
	CPU	**GPU**	**CPU**	**GPU**	**CPU**	**GPU**	**CPU**	**GPU**
RAM → *GlobalMem.* (Image)	–	0.0029	–	0.0286	–	0.0899	–	0.1762
RAM → *GlobalMem.*	–	0.0001	–	0.0011	–	0.0038	–	0.0077
Global Mem. → *RAM*	–	0.0003	–	0.0003	–	0.0003	–	0.0003
$\mathbf{Y^T} \times \mathbf{Y}$ (Line 3)	0.0068	0.0003	0.0722	0.0020	0.2192	0.0055	0.5278	0.0109
A (Line 7)	0.0057	0.0002	0.0720	0.0019	0.4079	0.0058	0.8330	0.0151
Sum (Line 18)	0.2261	0.0266	3.5457	0.2642	10.8152	0.8114	21.5404	1.6215
$\mathbf{sum} \times \mathbf{Y^T}$ (Line 18)	3.3205	0.1878	37.3522	1.2214	111.7478	3.2046	224.8251	6.1614
Q (Line 19)	0.3058	0.0150	3.6222	0.0144	10.7925	0.0144	21.3635	0.0144
$\mathbf{q} \times \mathbf{Y}$ (Line 20)	3.3957	0.1195	34.4545	0.9056	217.2635	2.7357	414.3540	5.4544
Hinge (Lines 14,23,29)	0.6989	0.0217	7.1441	0.1965	42.9824	0.5905	86.8990	1.6623
Soft (Line 20,21)	0.7290	0.0643	9.7433	0.5787	29.6798	1.3329	59.4543	2.6829
Total Parallelized	-	0.4354	-	3.0305	-	8.5268	-	17.6229
Non-parallelized	0.0067		0.0118		0.0378		0.0534	
Total	8.6952	0.4448	96.0180	3.0673	423.9461	8.6465	856.8505	17.7774
Speedups	-	19.2873	-	31.3037	-	49.0309	-	48.2091

Table 4.9: Achieved occupancy (%) for the GPU implementation of the algorithm on the CPU-GPU platform, tested with the four synthetic datasets ($p = 30$).

	Synthetic_1 100 × 100	**Synthetic_2** 400 × 250	**Synthetic_3** 512 × 600	**Synthetic_4** 1024 × 600
A	96.2	96.2	95.9	96.5
Sum	84.2	84.1	84.4	84.3
Q	45.3	45.3	45.3	45.3
Hinge	96.1	96.6	96.1	96.7
Soft	86.7	86.2	85.8	79.3

Table 4.10: Processing times (in seconds) and speedups achieved for the GPU implementation of the algorithm on GPU1 and GPU2, tested for the real AVIRIS Cuprite data set ($p = 19$).

n	**CPU**	**GPU1**	**Speedup1**	**GPU2**	**Speedup2**
350×350	45.267	3.585	12.626	2.910	15.556

Evaluation with Real Data

The proposed method is applied to real hyperspectral data collected by the AVIRIS sensor. Table 4.10 shows the resulting times for the Cuprite dataset for 19 endmembers, where the results include the memory transfers overhead among CPU and GPU.

This results reveals that the proposed implementation can reach high speedups over real datasets, and are in agreement with the ones for the simulated scenarios. One can note that best results are achieved on *GPU2* with speedup higher than 15 times. The *GPU2* clock frequency is slower but it has more CUDA cores than *GPU1*. It is worth to mention that, considering the results achieved over Cuprite dataset the proposed parallel method using *GPU2* is suitable for real-time hyperspectral unmixing systems.

Chapter 5

VCA Architecture Design and Implementation on FPGA

Herein, the Nascimento and Vestias work on VCA Architecture Design for FPGA is presented [89]. The methodology followed to design the hardware architecture starts with an analysis of the algorithm to identify the most computationally demanding steps followed by a detailed description of how each step of the algorithm is calculated. Then, a hardware architecture that supports the execution of each step of the algorithm is proposed.

5.1 Analysis and Implementation of the VCA Method

Knowing that the dataset live in a lower signal subspace one can use a pre-processing (DR) step to reduce the dimension of the data space before entering the hardware system, and consequently to simplify the hardware. However, the time taken by the DR step is quite large and is not compatible with real-time requirements making these hardware solutions not applicable to real-time processing. Since VCA also works without this pre-processing step, the proposed hardware implementation does not consider a DR pre-processing step.

The most complex task of VCA is the calculation of vector $\mathbf{f}$ (see line 4 on Algorithm 1), which requires the computation of a pseudo-inverse matrix, typically using the SVD [90], which is numerically very stable and accurate [91], but it is computationally very complex and demanding. The projection of the image onto the orthogonal direction (see line 5 on Algorithm 1) is

also computationally very demanding, due to the large spatial dimensions of the image. To reduce the complexity and improve the performance of the hardware implementation, several optimizations have been considered in the implementation of the VCA algorithm (see Algorithm 5).

Algorithm 5 VCA hardware implementation

```
 1: INPUT Y
 2: w := rand(1, L);
 3: temp := e_u;
 4: f := w;
 5: j := 1;
 6: for i := 1 to 8 do
 7:    if i > 1 then
 8:       temp := Y(:, index(i − 1));
 9:       if i > 2 then
10:          j := j + 1;
11:          for k := 1 to j − 1 do
12:             r := Dot_product(Q(:, k), temp);
13:             temp := temp − r × Q(:, k);
14:          end for
15:       end if
16:       r := 1/√(Dot_product(temp, temp));
17:       Q(:, j) := r × temp;
18:       proj := Dot_product(w, Q(:, j));
19:       f := f − proj × Q(:, j);
20:    end if
21:    v_old = 0;
22:    for k := 1 to 614 do
23:       v := Dot_product(f, Y(:, k));
24:       if v > v_old then
25:          v_old := v;
26:          index(i) := k;
27:       end if
28:    end for
29: end for
30: OUPUT index
```

In Algorithm 5 a QR decomposition using the classic Gram-Schmidt process [90] is used to determine vector **f** on Algorithm 1 instead of the SVD method. The QR decomposition is computationally less complex and demanding than SVD, however it is numerically less stable and accurate [92]. To reduce the computation time of the QR decomposition an incremental procedure is used. The procedure consists on reusing the orthogonal vectors Q determined so far, thus only one orthogonal vector needs to be computed on each iteration (see lines 9–17 on algorithm 5). Moreover, instead of normalizing the vector by performing multiple divisions (lines 16 and 17 on algorithm 5), the reciprocal of the norm of the vector is calculated first and then multiplied by the vector. Since the multiplication operation is faster and uses less hardware resources compared to the division operation,

a parallel implementation of the operation becomes faster and smaller when implemented in hardware [93, 94].

The number of additions and products,N_a and N_p, respectively, will increase with the number of iteration, thus for a number of endmembers p, these number of operations are given by

$$N_a = 4l + 2l\,\frac{p^2 - 3p + 2}{2}, \tag{5.1}$$

$$N_a = 3l - 2 + (2l - 1)\,\frac{p^2 - 3p + 2}{2}, \tag{5.2}$$

where usually the number of endmember(p) is smaller than the number of bands (l).

The projection of the image onto a direction to obtain vector $\mathbf{v}$ on Algorithm 1 consists on a vector-matrix multiplication (see lines 22–28 on Algorithm 5). The operation was speeded up by parallelizing the dot-product operation (sub–routine *Dot_product* in line 23 on algorithm 5) in order to process several bands on each cycle reducing proportionally the number of cycles needed to project a single pixel. The determination of $\mathbf{k}$ on Algorithm 1, the maximum entry of vector $\mathbf{v}$, is done in parallel with the dot-product operations, that is, the result of each dot-product is immediately compared with the previous entry to find and keep the maximum value. This optimization avoids the storage of the result of the dot product and reduces the execution time of the algorithm.

5.2 FPGA Architecture

This section presents the proposed hardware implementation of Algorithm 5 described in the previous section. The architecture is designed to support real-time processing of hyperspectral images acquired from AVIRIS sensor.

The architecture consists of a general-purpose processor (*ARM*) used to send the image to the hardware and to get the index of the pixels that were identified as endmembers by the hardware, a random number generator (*Random*), a memory module (*Distributed memory*) to store inputs and outputs of the algorithm, as well as temporary vectors and variables, an integer to float converter (*Int2float*), a dot-product calculator (*Dot-product*), a multiply-subtract module (*Mult-Sub*), a reciprocal square-root calculator used to normalize the last coefficient r of the QR decomposition ($1/\sqrt{x}$), and a floating-point comparator (*Max*) used to determine the biggest argument of all projections (see Figure 5.1).

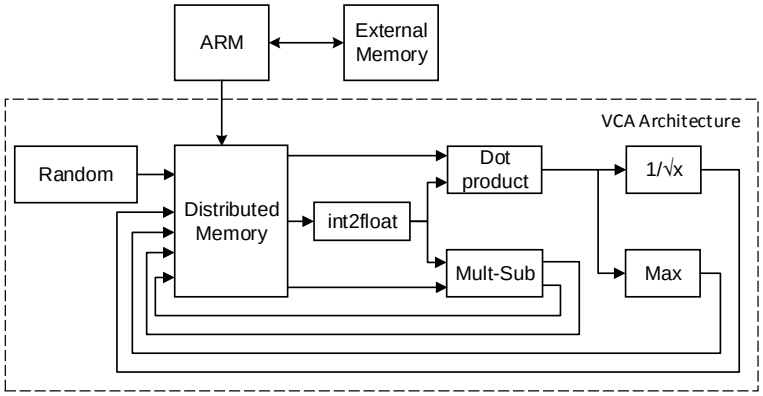


Figure 5.1: General overview of the VCA hardware architecture

Following the algorithm 5 described previously, the ARM starts transferring the hyperspectral image to the *Memory* module and then sends a control signal to the dedicated hardware. Initially, a pseudo-random vector ($\mathbf{w}$) is generated using the *Random-generator* module. At the same time a QR decomposition is computed using modules *Dot-product*, *Multiplier-Subtractor*, and *Reciprocal-square-root* that generate a new orthogonal vector. The result of this decomposition and its intermediate results are stored in *Memory*. Then, the orthogonal direction $\mathbf{f}$ is computed, using modules *Dot-product* and *Multiplier-Subtractor* and stored in *Memory*. Finally, the projection of image $\mathbf{Y}$ on vector $\mathbf{f}$ is computed using module *Dot-product*. As the results are produced they are sent to module *Comparator* that determines the index of the highest projection value. At the end of all processing a control signal is sent to the processor indicating the end of the algorithm and that the indexes of those pixels found as endmembers are available in memory.

In order to efficiently utilize the available resources, some modules are reused to implement different steps of the algorithm. For example, the *Dot-product* and the *Multiplier-Subtractor* modules are both used on the QR decomposition and the image projection. Also, all modules operates on single-precision floating-point data, following the standard IEEE 754-2008.51 [95]. Since the data image is stored as 15-bit integers, the *Int2float* module is used to convert from 15-bit integers to single-precision floating-point to be used on the *Dot-product* and in the *Multiplier-Subtrator* modules.

Hardware Modules of the VCA Architecture

The *Memory* module is composed by nine memory modules with different datawidths (see figure 5.2).

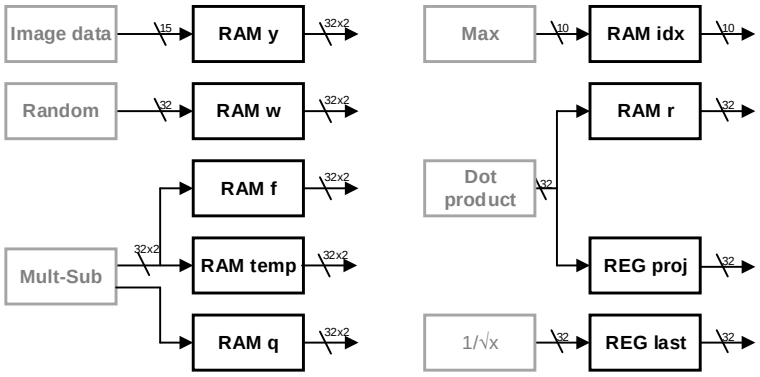


Figure 5.2: Structure of the memory module

RAM *y* is used to store the input image. RAM *w* is used to store the random vector consisting of the numbers generated by the random generator. Another RAM *f* is used to store the orthogonal direction produced by the multiplier-sutractor. RAM *temp* is used to store intermediate results from QR decomposition at the output of the multiplier-subtractor module. Memory RAM *q* stores orthogonal vectors from QR decomposition at the output of the multiplier-subtractor. Memory *idx* stores those pixels elected as endmembers found with module comparator. Memory *r* stores the coefficients from the QR decomposition produced by the dot product module. Register *last* stores the last coefficient from QR decomposition generated by the inverse square root module. Finally, register *proj* stores an intermediate result from the QR decomposition generated at module dot product.

The *Random-generator* module generates 32-bit integer numbers using the Mersenne twister algorithm [96]. The integers generated are then rearranged to form a single-precision floating point number, avoiding the use of an integer to floating point converter.

The *Multiplier-Subtractor* module is used to calculate a multiply-subtract operation over vectors. The module is designed with two multipliers and subtractors capable to process two bands at the same time (see figure 5.3).

The *Dot-product* block is used to multiply vectors and is composed by two parallel multiply-add blocks, *DP*, and an adder tree (see Figure 5.4). Each multiply-add block consists of a multiplier followed by an accumulator.

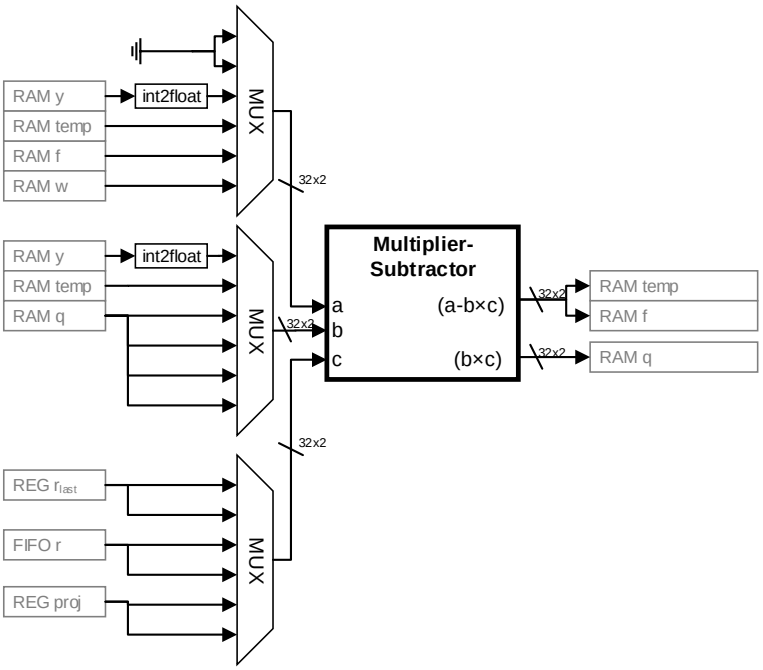


Figure 5.3: Structure of the multiplier-subtractor module

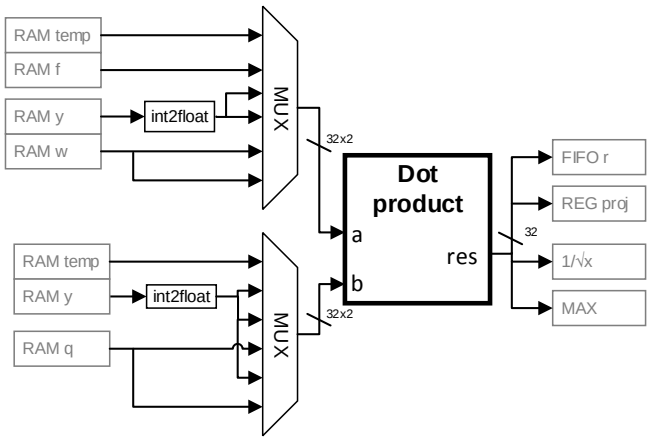


Figure 5.4: *Dot-product* module architecture.

The accumulator is designed with three cycles of latency to allow an higher operating frequency. Every block computes the dot product of *112* ($= \frac{224}{2}$) bands, which are then added with the tree adder.

This particular implementation uses two parallel multiplier blocks capable to process two spectral bands each clock cycle. Since the method is scalable, more parallel multiplier blocks can be considered, which would reduce the number of computation cycles and consequently the total processing time of the algorithm. The scalability is only limited by the available hardware resources. In the proposed architecture, the number of parallel multipliers was determined based on the timing requirements to guarantee real-time processing of hyperspectral data collected from AVIRIS, as will be shown in Section 5.3.

5.3 VCA - Experimental Results on FPGA

The proposed hardware architecture has been described in VHSIC (very high speed integrated circuit) hardware description language (VHDL) and implemented on a Xilinx Zynq Zedboard with a XC7Z020 SoC. The architecture is scalable in terms of parallel processing operators allowing to design architectures with different tradeoffs between area and performance. The hardware design has been done with Xilinx ISE, PlanAhead, Platform Studio and Software Development Kit tools. The FPGA board has 1 GB of DDR3 memory with 1.6 GB/s of memory bandwidth. This bandwidth is large enough to sustain the communication of the hyperspectral image to the FPGA. For example, real hyperspectral data collected by AVIRIS sensor requires only around 40 MB/s of memory bandwidth to guarantee real-time collection of data.

The proposed architecture may be easily implemented on other FPGAs with higher density or faster technologies, like Virtex-7. Since the proposed VCA implementation have reached real time results, there was no needed to use a better Xilinx FPGA. However, unmixing chain is a challenging task where DR step and inversion step (to estimate the abundances) are also included. For this purpose, in future work, an higher capability Xilinx FPGA may be used in order to implement the full unmixing chain and get real time or near real time response.

All basic modules, including multipliers, adders, subtractors and memories were generated using the Xilinx Core generator tool. The floating-point units were generated with the Xilinx LogiCORE IP CORE Floating point v6.1 [97], and memories were generated using the Xilinx LogiCORE IP Block Memory Generator v7.3 [98].

To test the architecture, the DDR memory available in the board was utilized to store the dataset. The ARM processor available in the FPGA

Table 5.1: SAD results between extracted endmembers and laboratory reflectances for the proposed Algorithm 2.

Substance	**SAD** without DR	**SAD** with DR
Alunite	3.95	3.72
Buddingtonite	3.41	3.12
Calcite	3.99	3.53
Kaolinite	4.74	4.50
Muscovite	3.78	3.44

was utilized to send and to receive data between the external memory and the architecture.

The following sections describe the endmembers extraction accuracy results performed by the developed hardware architecture and the implementation results in terms of used hardware resources and working frequency.

5.3.1 Endmember Extraction Accuracy Evaluation

The hardware implementation was evaluated with real hyperspectral data collected by AVIRIS [99] sensor over Cuprite mining district at Nevada. In order to process all pixels of the dataset, the architecture processes blocks of 614 pixels each time. The endmembers found on the last iteration are grouped with the next block of pixels.

To measure the accuracy of the implemented hardware the well known spectral angle distance (SAD) is adopted. Table 5.1 presents the SAD between the extracted endmembers and the closest laboratory reflectance for the substances that are most representative on the scene. For accuracy comparison purposes it is also shown the results for the same dataset with DR pre-processing step on the third column. As expected, the results with DR step are better than without this step.

5.3.2 Implementation Results

Due to the limited hardware resources the architecture is capable to process a dataset composed by 614 pixels with 224 spectral bands, which corresponds to one line of an image acquired by the AVIRIS sensor. For larger datasets, after the endmembers's signatures extraction of each line, a final selection is done to select the set of endmembers. Notice however, that if the dataset

has been already projected onto the signal subspace, by a DR method, the architecture herein presented still works in the same way, where unused bands are simply filled with zeros.

EEA	Size	Bands	p	Device	F.(MHz)	Time(s)	Slice	LUT	DSP
NFINDR [18]	350 × 350	224	16	Virtex-4 XC4VFX60	43	13,46	11700	20779	128
PPI [100]	350 × 350	189	22	Virtex-IIPROXC2VP30	187	31,30	10423	19317	0
PPI [101]	350 × 350	15	16	Virtex-4 XC4VFX60	102	1,35	21148	39432	0
NFINDR [101]	350 × 350	15	16	Virtex-4 XC4VFX60	42	13,46	11700	20779	128
MVCA [102]	250 × 191	14	14	Virtex-5 LX110T	150	0,063	NA	26164	54
MVCA [86]	250 × 191	14	14	Virtex5-XC5XS95T	210	0,042	NA	34000	0
MVCA [103]	250 × 191	14	14	Virtex5-XC5XS95T	244	0,042	NA	31000	0
VCA	614 × 512	224	8	ZYNQ-7020	100	2,86	3077	6753	35
VCA	350 × 350	224	8	ZYNQ-7020	100	1,13	3077	6753	35
VCA	614 × 512	224	16	ZYNQ-7020	100	2,90	4525	11159	63
VCA	350 × 350	224	16	ZYNQ-7020	100	1,16	4525	11159	63

Table 5.2: State-of-the-Art comparison with FPGAs.

Table 5.2 presents a comparison of the proposed algorithm and FPGA State-of-the-Art architectures. The proposed solution works at low frequency (100 MHz) and achieves the best performances compared to works that do not use dimensionality reduction. Works [86, 101, 102] perform dimensionality reduction whose execution time is not included in the total execution time. In these cases, we have tested our architecture considering also dimensionality reduction and achieved less than 30 ms, better than than the state-of-the-art at only 100 MHz of working frequency. Since AVIRIS sensor acquires 512 pixels of 224 spectral bands in 8.3 ms [104], the implemented architecture achieves real-time on-board hyperspectral data processing.

In terms of resources, it is difficult to compare with the state-of-art since different generations of FPGAs have different architectural structures. Virtex-4 and previous generations use 4-input LUTs, while recent FPGAs use 6-input LUTS. Also, some proposals use embedded multipliers and/or BRAM while others just use LUTs and most of the proposals work with integer and fixed-point arithmetic while our proposal considers single floating-point arithmetic. Comparing to previous Virtex-5 based architectures, which also use 6-input LUTs, our solution for 16 endmembers uses from 60% less number of LUTs and about the same number of DSP. Considering work [86] and [102], we use about 70% less number of LUTs but they do not use DSP. This is due to the fact that we are implementing single precision floating-point arithmetic, while they implemented integer arithmetic. A final aspect is the utilization of BRAMs. Our proposal utilizes the available BRAMs of FPGAs (72 BRAMs for 8 endmembers and 76 for 16 endmembers), which improves the performance of the solution and reduces the utilization of LUTs.

Considering the area of the target device, our architecture capable to

extract 16 endmembers from a 614×512 hyperspectral image at real-time uses only 34% of a low cost SoC FPGA. Since the architecture is scalable, more endmembers and faster architectures can be achieved with the same FPGA device. For example, with about 60% of the resources available we can improve performance almost two times.

EEA	**Size**	**Bands**	**p**	**Device**	**Freq. (MHz)**	**Time (s)**
OSP [105]	350×350	188	19	GeForce GTX 580	1544	0,56
VCA [106]	350×191	188	18	Nvidia GTX460	1350	0,81
PPI [107]	350×350	192	NA	Nvidia TeslaC1060	1300	3,37
VCA [84]	10^5	187	16	Nvidia GTX590	1250	0,72
NFINDR [108]	350×350	188	NA	Nvidia GTX275	1550	1,43
NFINDR [109]	250×191	189	16	Nvidia GTX460	1350	8.00
NFINDR [110]	350×350	192	19	IntelXeon E5520	2270	0,64
OSP [110]	350×350	192	19	IntelXeon E5520	2270	0,88
MVCA [111]	250×191	224	16	Nvidia GTX480	1400	0,56
VCA	614×512	224	8	ZYNQ-7020	100	2,86
VCA	350×350	224	8	ZYNQ-7020	100	1,13
VCA	614×512	224	16	ZYNQ-7020	100	2,90
VCA	350×350	224	16	ZYNQ-7020	100	1,16

Table 5.3: State-of-the-Art comparison with GPU and CPU.

Table 5.3 presents a comparison of the proposed algorithm and FPGA State-of-the-Art implementations for GPU and CPU. Compared to GPU and CPU platforms, FPGA proposals reach performances similar to the best GPU and CPU implementations but operating at 12 to 23 times lower clock frequencies. This is a very important factor given the low power requirements of embedded systems for hyperspectral unmixing. Also, since the architecture is scalable, we still have space for improvement as long as there are resources available in the FPGA.

Chapter 6

Conclusions

Onboard processing systems have recently emerged, in order to overcome the huge amount of data to transfer from the satellite to the ground station. Hyperspectral imagery is a remote sensing technology that can benefit of onboard processing. On the other hand, recently, deep learning-based methods applied to hyperspectral unmixing have emerged, nevertheless, there are still several drawbacks, namely, the requirement of a lot of training samples to achieve satisfactory accuracy performance.

As lines of research for the future are the optimization of the design of the methods herein presented in order to achieve higher working frequencies, opening new perspectives for low-cost onboard hyperspectral image processing; the implementation on reconfigurable hardware of a full hyperspectral unmixing chain and of statistical unmixing methods in order to get real-time or near real-time response for such demanding methods; the development of unmixing and compression methods based on deep-learning architectures with small data.

Bibliography

[1] P. Ghamisi, N. Yokoya, J. Li, W. Liao, S. Liu, J. Plaza, B. Rasti, and A. Plaza, "Advances in hyperspectral image and signal processing: A comprehensive overview of the state of the art," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 4, pp. 37–78, Dec 2017.

[2] S.-E. Qian, "Hyperspectral satellites, evolution, and development history," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 7032–7056, 2021.

[3] W.-K. Ma, J. M. Bioucas-Dias, T.-H. Chan, N. Gillis, P. Gader, A. J. Plaza, A. Ambikapathi, and C.-Y. Chi, "A signal processing perspective on hyperspectral unmixing: Insights from remote sensing," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 67–81, 2014.

[4] F. Vanegas, D. Bratanov, K. Powell, J. Weiss, and F. Gonzalez, "A novel methodology for improving plant pest surveillance in vineyards and crops using uav-based hyperspectral and spatial data," *Sensors*, vol. 18, no. 1, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/1/260

[5] M. Shimoni, R. Haelterman, and C. Perneel, "Hypersectral imaging for military and security applications: Combining myriad processing and sensing techniques," *IEEE Geoscience and Remote Sensing Magazine*, vol. 7, no. 2, pp. 101–117, 2019.

[6] H. Fabelo, S. Ortega, R. Guerra, G. Callico, A. Szolna, J. F. Pineiro, M. Tejedor, S. Lopez, and R. Sarmiento, "A novel use of hyperspectral images for human brain cancer detection using in-vivo samples," in *Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 4: Smart-BIODEV, (BIOSTEC 2016)*, INSTICC. SciTePress, 2016, pp. 311–320.

[7] G. Lu and B. Feia, "Medical hyperspectral imaging: a review," *J Biomed Opt*, vol. 24441941, 2014.

[8] X. Fu and J. Chen, "A review of hyperspectral imaging for chicken meat safety and quality evaluation: Application, hardware, and software," *Comprehensive Reviews in Food Science and Food Safety*, vol. 18, no. 2, pp. 535–547, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1541-4337.12428

[9] J. Yang, D. W. Messinger, and R. R. Dube, "Bloodstain detection and discrimination impacted by spectral shift when using an interference filter-based visible and near-infrared multispectral crime scene imaging system," *Optical Engineering*, vol. 57, no. 3, pp. 1 – 10, 2018. [Online]. Available: https://doi.org/10.1117/1.OE.57.3.033101

[10] B. Li, P. Beveridge, W. T. O'Hare, and M. Islam, "The application of visible wavelength reflectance hyperspectral imaging for the detection and identification of blood stains," *Science & Justice*, vol. 54, no. 6, pp. 432 – 438, 2014.

[11] S. Lopez, T. Vladimirova, C. Gonzalez, J. Resano, D. Mozos, and A. Plaza, "The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends," *Proceedings of the IEEE*, vol. 101, no. 3, pp. 698–722, March 2013.

[12] A. Plaza and C.-I. Chang, Eds., *High Performance Computing in Remote Sensing*. Chapman Hall CRC, 2007.

[13] A. Yusuf and S. Alawneh, "A survey of gpu implementations for hyperspectral image classification in remote sensing," *Canadian Journal of Remote Sensing*, 02 2019.

[14] E. Torti, G. Danese, F. Leporati, and A. Plaza, "A hybrid cpu-gpu real-time hyperspectral unmixing chain," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 945–951, 2016.

[15] S. Vellas, G. Lentaris, K. Maragos, D. Soudris, Z. Kandylakis, and K. Karantzalos, "Fpga acceleration of hyperspectral image processing for high-speed detection applications," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[16] A. Plaza, Q. Du, Y.-L. Chang, and R. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 4, no. 3, pp. 528 –544, 2011.

[17] S. Lopez, P. Horstrand, G. Callico, J. Lopez, and R. Sarmiento, "A low-computational-complexity algorithm for hyperspectral endmember extraction: Modified vertex component analysis," *IEEE Geosci. Remote Sensing Let.*, vol. 9, no. 3, pp. 502–506, 2012.

[18] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "Fpga implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 2, pp. 374–388, 2012.

[19] M. Hsueh and C.-I. Chang, "Field programmable gate arrays (fpga) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 408–423, 2008. [Online]. Available: http://hpc.sagepub.com/content/22/4/408.abstract

[20] C. i Chang, W. Xiong, and C.-C. Wu, "Field-programmable gate array design of implementing simplex growing algorithm for hyperspectral endmember extraction," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 51, no. 3, pp. 1693–1700, March 2013.

[21] S. Bernabe, S. Lopez, A. Plaza, R. Sarmiento, and P. Rodriguez, "Fpga design of an automatic target generation process for hyperspectral image analysis," in *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*, Dec 2011, pp. 1010–1015.

[22] C. Gonzalez, J. Resano, A. Plaza, and D. Mozos, "Fpga implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 5, no. 1, pp. 248–261, Feb 2012.

[23] R. Macias, S. Bernabe, D. Bascones, and C. Gonzalez, "Fpga implementation of a hardware optimized automatic target detection and classification algorithm for hyperspectral image analysis," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2022.

[24] C. Gonzalez, S. Lopez, D. Mozos, and R. Sarmiento, "Fpga implementation of the hysime algorithm for the determination of the number of endmembers in hyperspectral data," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2870–2883, 2015.

[25] A. Tsigkanos, N. Kranitis, D. Theodoropoulos, and A. Paschalis, "High-performance cots fpga soc for parallel hyperspectral image compression with ccsds-123.0-b-1," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2397–2409, 2020.

[26] X. Ceamanos and S. Valero, "4 - processing hyperspectral images," in *Optical Remote Sensing of Land Surface*, N. Baghdadi and M. Zribi, Eds. Elsevier, 2016, pp. 163 – 200.

[27] J. M. Amigo, Ed., *Hyperspectral Imaging*, 1st ed., ser. Data Handling in Science and Technology. Elsevier, 2019.

[28] R. A. Borsoi, T. Imbiriba, J. C. M. Bermudez, C. Richard, J. Chanussot, L. Drumetz, J.-Y. Tourneret, A. Zare, and C. Jutten, "Spectral variability in hyperspectral data unmixing: A comprehensive review," *IEEE Geoscience and Remote Sensing Magazine*, vol. 9, no. 4, pp. 223–270, 2021.

[29] A. Paz and A. Plaza, "Clusters vs. gpus for parallel automatic target detection in hyperspectral images," *EURASIP Journal of Advances in Signal Processing*, vol. 2010, p. 18, 2010.

[30] Y. Altmann, N. Dobigeon, and J.-Y. Tourneret, "Unsupervised postnonlinear unmixing of hyperspectral images using a hamiltonian monte carlo algorithm." *IEEE Trans. Image Processing*, vol. 23, no. 6, pp. 2663–2675, 2014.

[31] J. Nascimento and J. Bioucas-Dias, "Vertex Component Analysis: A Fast Algorithm to Unmix Hyperspectral Data," *IEEE Trans. Geosci. Remote Sensing*, vol. 43, no. 4, pp. 898–910, 2005.

[32] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "Spatial/Spectral Endmember Extraction by Multidimensional Morphological Operations," *IEEE Trans. Geosci. Remote Sensing*, vol. 40, no. 9, pp. 2025–2041, 2002.

[33] J. M. P. Nascimento and J. M. Bioucas-Dias, "Hyperspectral unmixing based on mixtures of dirichlet components," *IEEE Trans. Geosci. Remote Sensing*, vol. 50, no. 3, pp. 863–878, 2012.

[34] L. Miao and H. Qi, "Endmember Extraction From Highly Mixed Data Using Minimum Volume Constrained Nonnegative Matrix Factorization," *IEEE Trans. Geosci. Remote Sensing*, vol. 45, no. 3, pp. 765–777, 2007.

[35] R. Heylen, M. Parente, and P. D. Gader, "A review of nonlinear hyperspectral unmixing methods," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 6, pp. 1844–1868, 2014.

[36] J. M. P. Nascimento and J. M. Bioucas-Dias, "Nonlinear mixture model for hyperspectral unmixing," L. Bruzzone, C. Notarnicola, and F. Posa, Eds., vol. 7477. SPIE, September 2009, pp. 74 770I–74 770I–8. [Online]. Available: http://dx.doi.org/10.1117/12.830492

[37] G. Zhang, P. Scheunders, D. Cerra, and R. MÃ¼ller, "Shadow-aware nonlinear spectral unmixing for hyperspectral imagery," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 5514–5533, 2022.

[38] B. Hapke, "Bidirection Reflectance spectroscopy. I. Theory," *J. of Geophysical Research*, vol. 86, pp. 3039–3054, 1981.

[39] ——, *Theory of Reflectance and Emmittance Spectroscopy.* Cambridge, U. K.: Cambridge Univ. Press, 2005.

[40] B. Somers, K. Cool, S. Delalieux, J. Stuckens, D. V. der Zande, W. W. Verstraeten, and P. Coppin, "Nonlinear Hyerpspectral Mixture Analysis for Tree Cover estimates in Orchards," *Rem. Sens. of the Environ.*, vol. 113, pp. 1183–1193, 2009.

[41] L. Tits, B. Somers, J. Stuckens, and P. Coppin, "Validating nonlinear mixing models: Benchmark datasets from vegetated areas," in *2014 6th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS)*, 2014, pp. 1–4.

[42] B. Rasti, B. Koirala, and P. Scheunders, "Hapkecnn: Blind nonlinear unmixing for intimate mixtures using hapke model and convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–15, 2022.

[43] J. Broadwater, R. Chellappa, A. Banerjee, and P. Burlina, "Kernel fully constrained least squares abundance estimates," in *2007 IEEE International Geoscience and Remote Sensing Symposium*, 2007, pp. 4041–4044.

[44] M. Brown, H. Lewis, and S. Gunn, "Linear spectral mixture models and support vector machines for remote sensing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, no. 5, pp. 2346–2360, 2000.

[45] R. Heylen, D. Burazerovic, and P. Scheunders, "Non-linear spectral unmixing by geodesic simplex volume maximization," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 3, pp. 534–542, 2011.

[46] A. Zare, P. Gader, O. Bchir, and H. Frigui, "Piecewise convex multiple-model endmember detection and spectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 51, no. 5, pp. 2853–2862, 2013.

[47] M. Li, B. Yang, and B. Wang, "Spectral-spatial reweighted robust nonlinear unmixing for hyperspectral images based on an extended multilinear mixing model," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–17, 2022.

[48] B. Yang and B. Wang, "Band-wise nonlinear unmixing for hyperspectral imagery using an extended multilinear mixing model," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 11, pp. 6747–6762, 2018.

[49] R. Heylen and P. Scheunders, "A multilinear mixing model for nonlinear spectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 1, pp. 240–251, 2016.

[50] J. M. P. Nascimento and G. Martin, "Chapter 2.6 - nonlinear spectral unmixing," in *Hyperspectral Imaging*, ser. Data Handling in Science and Technology, J. M. Amigo, Ed. Elsevier, 2020, vol. 32, pp. 151–166.

[51] J. F. Mustard and C. Pieters, "Quantitative Abundance Estimates from Bidirecional Reflectance Measurements," in *Proc. of the 17th Lunar and Planetary Science. Conference*, 1987, pp. E617–E626.

[52] J. M. P. Nascimento and J. M. Bioucas-Dias, "Unmixing hyperspectral intimate mixtures," L. Bruzzone, Ed., vol. 7830. SPIE, September 2010, pp. 78 300C–78 300C–8. [Online]. Available: http://dx.doi.org/10.1117/12.865118

[53] X.-R. Feng, H.-C. Li, R. Wang, Q. Du, X. Jia, and A. Plaza, "Hyperspectral unmixing based on nonnegative matrix factorization: A comprehensive review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 4414–4436, 2022.

[54] J. Boardman, "Automating Spectral Unmixing of AVIRIS Data using Convex Geometry Concepts," in *Summaries of the Fourth Annual JPL Airborne Geoscience Workshop, JPL Pub. 93-26, AVIRIS Workshop.*, vol. 1, 1993, pp. 11–14.

[55] M. E. Winter, "N-FINDR: An Algorithm for Fast Autonomous Spectral End-member Determination in Hyperspectral Data," in *Proc. of the SPIE conference on Imaging Spectrometry V*, vol. 3753, 1999, pp. 266–275.

[56] T.-H. Chan, W.-K. Ma, A. Ambikapathi, and C.-Y. Chi, "A simplex volume maximization framework for hyperspectral endmember extraction," *IEEE Trans. Geosci. Remote Sensing*, vol. 49, no. 11, pp. 4177 –4193, 2011.

[57] H. Ren and C.-I. Chang, "Automatic spectral target recognition in hyperspectral imagery," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1232–1249, 2003.

[58] R. Marrero, S. Lopez, G. M. Callico, M. A. Veganzones, A. Plaza, J. Chanussot, and R. Sarmiento, "A novel negative abundance oriented hyperspectral unmixing algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 7, pp. 3772–3790, 2015.

[59] J. M. Bioucas-Dias, "A Variable Splitting Augmented Lagrangian Approach to Linear Spectral Unmixing," in *First IEEE GRSS Workshop on Hyperspectral Image and Signal Processing-WHISPERS'2009*, 2009.

[60] A. Zare and P. Gader, "Sparsity Promoting Iterated Constrained Endmember Detection in Hyperspectral Imagery," *IEEE Geosci. Remote Sensing Let.*, vol. 4, no. 3, pp. 446 – 450, 2007.

[61] N. Gillis and S. Vavasis, "Fast and robust recursive algorithms for separable nonnegative matrix factorization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2013.

[62] N. Dobigeon, S. Moussaoui, J.-Y. Tourneret, and C. Carteret, "Bayesian Separation of Spectral Sources under Non-negativity and Full Additivity Constraints," *Signal Processing*, vol. 89, no. 12, pp. 2657–2669, 2009.

[63] M. Arngren, M. N. Schmidt, and J. larsen, "Bayesian Nonnegative Matrix Factorization with Volume Prior for Unmixing of Hyperspectral Images," in *Machine Learning for Signal Processing, IEEE Workshop on (MLSP)*, Sep 2009.

[64] S. Moussaoui, C. Carteret, D. Brie, and A. Mohammad-Djafari, "Bayesian Analysis of Spectral Mixture Data using Markov Chain Monte Carlo Methods," *Chemometrics and Intelligent Laboratory Systems*, vol. 81, no. 2, pp. 137–148, 2006.

[65] Y. Qian, F. Xiong, Q. Qian, and J. Zhou, "Spectral mixture model inspired network architectures for hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 10, pp. 7418–7434, 2020.

[66] F. Xiong, J. Zhou, S. Tao, J. Lu, and Y. Qian, "Snmf-net: Learning a deep alternating neural network for hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–16, 2022.

[67] M. Zhao, M. Wang, J. Chen, and S. Rahardja, "Hyperspectral unmixing for additive nonlinear models with a 3-d-cnn autoencoder network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–15, 2022.

[68] J. S. Bhatt and M. V. Joshi, "Deep learning in hyperspectral unmixing: A review," in *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, 2020, pp. 2189–2192.

[69] C. Zhou and M. R. D. Rodrigues, "Admm-based hyperspectral unmixing networks for abundance and endmember estimation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–18, 2022.

[70] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral Subspace Identification," *IEEE Trans. Geosci. Remote Sensing*, vol. 46, no. 8, pp. 2435–2445, 2008.

[71] X. Tao, M. E. Paoletti, J. M. Haut, P. Ren, J. Plaza, and A. Plaza, "Endmember estimation with maximum distance analysis," *Remote Sensing*, vol. 13, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2072-4292/13/4/713

[72] C.-I. Chang and Q. Du, "Estimation of Number of Spectrally Distinct Signal Sources in Hyperspectral Imagery," *IEEE Trans. Geosci. Remote Sensing*, vol. 42, no. 3, pp. 608–619, 2004.

[73] P. Bajorski, "Second Moment Linear Dimensionality as an Alternative to Virtual Dimensionality," *IEEE Trans. Geosci. Remote Sensing*, vol. 49, no. 2, pp. 672–678, 2011.

[74] N. A. M. Diani and G. Corsini, "Hyperspectral Signal Subspace Identification in the Presence of Rare Signal Components," *IEEE Trans. Geosci. Remote Sensing*, vol. 48, no. 4, pp. 1940–1954, 2010.

[75] J. Broadwater, R. Meth, and R. Chellappa, "Dimensionality Estimation in Hyper-spectral Imagery Using Minimum Description Length," in *Proceedings of the Army Science Conference*, Orlando, FL, November 2004.

[76] I. T. Jolliffe, *Principal Component Analysis*. New York: Spriger Verlag, 1986.

[77] L. L. Scharf, *Statistical Signal Processing, Detection Estimation and Time Series Analysis*. Addison-Wesley Pub. Comp., 1991.

[78] A. Green, M. Berman, P. Switzer, and M. D. Craig, "A Transformation for Ordering Multispectral Data in Terms of Image Quality with Implications for Noise Removal," *IEEE Trans. Geosci. Remote Sensing*, vol. 26, no. 1, pp. 65–74, 1988.

[79] R. N. Clark, G. A. Swayze, A. Gallagher, T. V. King, and W. M. Calvin, "The U.S. Geological Survey Digital Spectral Library: Version 1: 0.2 to 3.0 μm," U.S. Geological Survey, Open File Report 93-592, 1993.

[80] M. D. Craig, "Minimum-volume Transforms for Remotely Sensed Data," *IEEE Trans. Geosci. Remote Sensing*, vol. 32, pp. 99–109, 1994.

[81] T. Han and T. Abdelrahman, "hicuda: High-level gpgpu programming," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 1, pp. 78–90, 2011.

[82] J. Bioucas-Dias and M. Figueiredo, "Alternating direction algorithms for constrained sparse regression: Application to hyperspectral unmixing," in *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), 2010 2nd Workshop on*, 2010, pp. 1 –4.

[83] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers." *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[84] J. Nascimento, J. Bioucas-Dias, J. Rodriguez, V. Silva, and A. Plaza, "Parallel hyperspectral unmixing on gpus," *Geoscience and Remote Sensing Letters, IEEE*, vol. 11, no. 3, pp. 666–670, 2014.

[85] G. Swayze, R. Clark, S. Sutley, and A. Gallagher, "Ground-Truthing AVIRIS Mineral Mapping at Cuprite, Nevada," in *Summaries of the Third Annual JPL Airborne Geosciences Workshop*, 1992, pp. 47–49.

[86] S. Lopez, P. Horstrand, G. Callico, J. Lopez, and R. Sarmiento, "A novel architecture for hyperspectral endmember extraction by means of the modified vertex component analysis (mvca) algorithm," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 6, pp. 1837–1848, 2012.

[87] J. Sevilla, G. MartÃn, and J. M. P. Nascimento, "Parallel hyperspectral unmixing method via split augmented lagrangian on gpu," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 5, pp. 626–630, May 2016.

[88] G. S. Miller, "The definition and rendering of terrain maps," in *ACM SIGGRAPH Computer Graphics*, vol. 20, no. 4. ACM, 1986, pp. 39–48.

[89] M. V. José M. P. Nascimento, "System-on-chip field-programmable gate array design for onboard real-time hyperspectral unmixing," *Journal of Applied Remote Sensing*, vol. 10, no. 1, pp. 1 – 17 – 17, 2016. [Online]. Available: https://doi.org/10.1117/1.JRS.10.015004

[90] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed., ser. Mathematical Sciences. John Hopkins University Press, 1996.

[91] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[92] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002.

[93] M. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand, "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 628–637, Jul. 2000.

[94] J. Blinn, "Floating-point tricks," *Computer Graphics and Applications, IEEE*, vol. 17, no. 4, pp. 80–84, Jul 1997.

[95] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, Aug 2008.

[96] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998. [Online]. Available: http://doi.acm.org/10.1145/272991.272995

[97] Xilinx, "Logicore ip floating-point operator v6.1," Jul. 2012. [Online]. Available: http://www.xilinx.com/support/documentation

[98] ——, "Logicore ip block memory generator v7.3," Dec. 2012. [Online]. Available: http://www.xilinx.com/support/documentation

[99] G. Vane, R. Green, T. Chrien, H. Enmark, E. Hansen, and W. Porter, "The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Rem. Sens. of the Environ.*, vol. 44, pp. 127–143, 1993.

[100] C. Gonzalez, S. Sanchez, A. Paz, J. Resano, D. Mozos, and A. Plaza, "Use of fpga or gpu-based architectures for remotely sensed hyperspectral image processing," *Integration*, vol. 46, no. 2, pp. 89–103, 2013.

[101] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of endmember extraction algorithms from hyperspectral imagery: pixel purity index versus N-FINDR," in *High-Performance Computing in Remote Sensing*, B. Huang and A. J. Plaza, Eds., vol. 8183, International Society for Optics and Photonics. SPIE, 2011, p. 81830F. [Online]. Available: https://doi.org/10.1117/12.897384

[102] T. G. Cervero, J. Caba, S. Lopez, J. D. Dondo, R. Sarmiento, F. Rincon, and J. Lopez, "A scalable and dynamically reconfigurable fpga-based embedded system for real-time hyperspectral unmixing," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2894–2911, 2015.

[103] T. Cervero, S. Lopez, G. Callico, J. Lopez, and R. Sarmiento, "Scalable architectures for real-time hyperspectral unmixing," *Microelectronics Journal*, vol. 45, no. 10, pp. 1292–1303, 2014, dCIS 12 Special Issue.

[104] R. Green, M. Eastwood, C. Sarture *et al.*, "Imaging Spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Rem. Sens. of the Environ.*, vol. 65, no. 3, pp. 227–248, 1998.

[105] S. Bernabe, S. Sánchez, A. Plaza, S. Lopez, J. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 6, no. 3, pp. 1386–1398, 2013.

[106] A. Barberis, G. Danese, F. Leporati, A. Plaza, and E. Torti, "Real-time implementation of the vertex component analysis algorithm on gpus," *IEEE Geoscience and Remote Sensing Letters*, vol. 10, no. 2, pp. 251–255, 2013.

[107] X. Wu, B. Huang, A. Plaza, Y. Li, and C. Wu, "Real-time implementation of the pixel purity index algorithm for endmember identification on gpus," *IEEE Geoscience and Remote Sensing Letters*, vol. 11, no. 5, pp. 955–959, 2014.

[108] S. Sanchez and A. Plaza, "A comparative analysis of GPU implementations of spectral unmixing algorithms," in *High-Performance*

Computing in Remote Sensing, B. Huang and A. J. Plaza, Eds., vol. 8183, International Society for Optics and Photonics. SPIE, 2011, p. 81830E. [Online]. Available: https://doi.org/10.1117/12.897329

[109] M. ElMaghrbay, R. Ammar, and S. Rajasekaran, "Fast gpu algorithms for endmember extraction from hyperspectral images," in *2012 IEEE Symposium on Computers and Communications (ISCC)*, 2012, pp. 000 631–000 636.

[110] A. Remon, S. Sanchez, A. Paz, E. S. Quintana-Orti, and A. Plaza, "Real-time endmember extraction on multicore processors," *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 5, pp. 924–928, 2011.

[111] G. M. Callico, S. Lopez, B. Aguilar, J. F. Lopez, and R. Sarmiento, "Parallel implementation of the modified vertex component analysis algorithm for hyperspectral unmixing using opencl," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 8, pp. 3650–3659, 2014.