

Article

A Runtime-Scalable and Hardware-Accelerated Approach to On-Board Linear Unmixing of Hyperspectral Images

Alberto Ortiz ^{1,*} , Alfonso Rodríguez ¹ , Raúl Guerra ² , Sebastián López ² ,
Andrés Otero ¹ , Roberto Sarmiento ²  and Eduardo de la Torre ¹ 

¹ Centro de Electrónica Industrial, Universidad Politécnica de Madrid, José Gutiérrez Abascal 2, 28006 Madrid, Spain; alfonso.rodriguez@upm.es (A.R.); joseandres.otero@upm.es (A.O.); eduardo.delatorre@upm.es (E.d.l.T.)

² Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria (ULPGC), 35001 Las Palmas de Gran Canaria, Spain; rguerra@iuma.ulpgc.es (R.G.); seblopez@iuma.ulpgc.es (S.L.); roberto@iuma.ulpgc.es (R.S.)

* Correspondence: alberto.ortiz@upm.es; Tel.: +34-910-676-952 or +34-910-676-953

Received: 12 October 2018; Accepted: 10 November 2018; Published: 12 November 2018



Abstract: Space missions are facing disruptive innovation since the appearance of small, lightweight, and low-cost satellites (e.g., CubeSats). The use of commercial devices and their limitations in cost usually entail a decrease in available on-board computing power. To face this change, the on-board processing paradigm is advancing towards the clustering of satellites, and moving to distributed and collaborative schemes in order to maintain acceptable performance levels in complex applications such as hyperspectral image processing. In this scenario, hybrid hardware/software and reconfigurable computing have appeared as key enabling technologies, even though they increase complexity in both design and run time. In this paper, the ARTICo³ framework, which abstracts and eases the design and run-time management of hardware-accelerated systems, has been used to deploy a networked implementation of the Fast UNmixing (FUN) algorithm, which performs linear unmixing of hyperspectral images in a small cluster of reconfigurable computing devices that emulates a distributed on-board processing scenario. Algorithmic modifications have been proposed to enable data-level parallelism and foster scalability in two ways: on the one hand, in the number of accelerators per reconfigurable device; on the other hand, in the number of network nodes. Experimental results motivate the use of ARTICo³-enabled systems for on-board processing in applications traditionally addressed by high-performance on-Earth computation. Results also show that the proposed implementation may be better, for certain configurations, than an equivalent software-based solution in both performance and energy efficiency, achieving great scalability that is only limited by communication bandwidth.

Keywords: hyperspectral imaging; linear unmixing; FPGAs; on-board processing; ARTICo³

1. Introduction

Hyperspectral imaging technology has been used in a wide range of applications in the field of Earth observation, such as vegetation control, precision agriculture, or urban surveillance [1]. The continuous evolution of this technology is building a promising future and bringing the dawn of new potential applications (e.g., future healthcare systems [2]). However, hyperspectral image processing poses several challenges in terms of computing requirements and algorithm development, especially when taking into account the need for increased spatial and temporal resolution in

hyperspectral sensors. In fact, the traditional approach relies on the use of High-Performance Computing (HPC) infrastructures to satisfy the required performance levels [3].

Hence, on-Earth processing has been the mainstream solution for remote-sensing applications that use hyperspectral images, relying on supercomputing systems typically based on GPUs [4], CPUs [5], heterogeneous CPU/GPU architectures [6], or even FPGAs [7]. In this scenario, on-board data compression techniques are used to minimize the overhead of data transmissions between sensors and processing facilities. These hyperspectral image compressors are usually implemented in large and computationally powerful FPGAs due to the combination of flexibility and reliability that these devices offer [8,9].

Nevertheless, the on-board sensing and processing paradigm has changed in the past few years [10]. The appearance of SmallSats, and more specifically CubeSats, has fostered the emergence of missions that target low-cost, -size, and -weight satellites and components. As a result, the overall price of spatial missions has decreased significantly, enabling the possibility of launching dozens of satellites in a single deployment. This approach was first envisioned for educational purposes and low-cost technology demonstrations, due to the satellites being limited in size ($10 \times 10 \times 10 \text{ cm}^3$) and power consumption (a few watts). This, in turn, allowed the deployment of low-cost Commercial Off-The-Shelf (COTS) devices to evaluate their feasibility within space applications [11].

Although limited at the beginning, new research lines have boosted the computing capabilities of CubeSat systems. An example can be found in satellite clustering [12], which is an attempt to provide comparable or even greater performance than before [13]. In fact, this approach offers new opportunities beyond the use of CubeSats as educational tools for low-cost science applications, creating an ecosystem for forthcoming paradigm shifts in space applications. In this scenario, on-board processing on the edge may see significant improvements, avoiding time-consuming data transmissions from satellite to on-Earth processing facilities and thus increasing temporal resolution. However, computing performance is not the only constraint, since severe restrictions in the available power budget make energy efficiency another key factor to take into account when evaluating platforms and applications for this kind of space missions [14].

Taking into account these restrictions, many devices have been evaluated [15], envisioning hybrid and reconfigurable computing systems as the target fabrics. Moreover, the combination of software- and hardware-based processing in a single board or even in a single device is hinted as the optimal choice. In this regard, Systems on Programmable Chip (SoPCs) can be used, since they provide multicore processing capabilities tightly coupled with dedicated hardware accelerators in an FPGA fabric. Furthermore, Dynamic and Partial Reconfiguration (DPR) capabilities can be exploited to provide not only functional adaptation, but also fault tolerance in the reconfigurable fabric [16,17]. However, the use of DPR at run time to search for an adaptive working point trade-off between performance, energy efficiency, and fault tolerance in low-cost COTS devices is something not commonly found in the literature.

In this paper, a distributed and hardware-accelerated implementation of a linear unmixing algorithm for on-board hyperspectral image processing is presented. Although hyperspectral image unmixing is an application that has been following the classical execution flow (i.e., on-board compression, and on-Earth decompression and processing), the use of a small and low-cost computing cluster with SoPC-based nodes, emulating real satellite deployment, provides a competitive alternative for distributed edge computing on CubeSats. Each node uses the ARTICo³ architecture [18], a hardware-based processing architecture for high-performance embedded systems, and its DPR infrastructure to support user-driven run-time adaptation of computing performance, energy consumption and fault tolerance. In addition, the ARTICo³ toolchain has been used to automate the implementation of the mixed hardware/software system, leveraging High-Level Synthesis (HLS) engines, and thus rendering low development times, a key factor in CubeSat deployments. As entry point for the design flow, a high-level description of the Fast UNmixing (FUN) algorithm [19] in C code has been modified to fully exploit data-level parallelism and enable a two-way scalable execution

pattern: on the one hand, in the number of ARTICO³ accelerators per computing node; on the other hand, in the number of SoPC-based nodes involved in the processing stage. Although the data-parallel extension of the FUN algorithm has been implemented and evaluated using the ARTICO³ architecture, it is platform-agnostic and, thus, it can be deployed on any computing platform as long as the parallelism holds. Moreover, the proposed block-based partitioning scheme can provide enhanced fault tolerance to the final implementation, making it possible to extract nearly all endmembers from an input dataset even if the data associated to one of the blocks get corrupted (e.g., due to radiation in space).

In summary, the main contributions of this paper are:

- A data-parallel linear unmixing algorithm for on-board hyperspectral image processing based on the FUN algorithm (originally conceived for on-Earth processing).
- A low-cost, networked, and hardware-accelerated implementation of that algorithm with two scalability levels and run-time adaptation capabilities in terms of computing performance and energy efficiency.

The rest of this paper is organized as follows. Section 2 presents an overview of the related work. The FUN algorithm and the ARTICO³ framework, which constitute the technological foundations of the presented work, are presented on Section 3. Section 4 gives an overview of the implementation details, including algorithmic modifications, hardware-optimization techniques, and network-distribution approaches. Section 5 shows the experimental validation and the obtained results, and Section 6 provides the conclusions and future work.

2. Related Work

This section is divided into two subsections. The first addresses the importance and continuous development in unmixing algorithms for hyperspectral images, whereas the second addresses the recent tendency for on-board processing based on distributed and scalable systems.

2.1. Unmixing of Hyperspectral Images

High spectral resolution in hyperspectral images makes it possible to identify materials by analyzing the spectral signatures of each pixel. This analysis can be compromised if there is not enough spatial resolution, a problem typically found in airborne and low-cost sensors. Hence, the spectral unmixing of the pixels has proven to be necessary to address the aforementioned problem [20].

In remote-sensing scenarios, hyperspectral unmixing has been traditionally done in Earth facilities after compression/decompression on the satellite link. In order to achieve real-time execution performance, and due to high computational loads, High-Performance Computing systems are commonly used [21,22]. In particular, CPUs, GPUs or CPU/GPU architectures have been used in most solutions. For instance, a real-time implementation of endmember extraction was reported in Reference [23], and, although part of the unmixing chain (i.e., abundance calculation) was not present, it marked the path to follow. One year later, the same group presented the first real-time implementation of the complete unmixing chain, using only one GPU and combining different algorithms for each part of the chain [24]. However, one of the weak points of the proposed implementation was the accuracy of the endmember extraction stage. In Reference [19], a new linear unmixing algorithm that provided better accuracy and performance results than state-of-the-art alternatives was proposed. Optimized implementations of the algorithm were presented in Reference [25] for GPUs and Reference [26] for FPGAs. However, none of these implementations considers on-board processing.

In fact, the concept of on-board processing of hyperspectral images usually refers to data compression on the edge. In this context, where flexibility and reliability become as important as real-time execution, FPGAs have been the most-used devices. Many theoretical approaches have proposed different compression algorithms, but their evaluation is not done on space-compliant devices [27,28]. Hence, no experimental data or performance metrics under space constraints can be

found in those works. However, it is also possible to find solutions throughout the literature using GPUs or FPGAs showing potential real-time results [8,9,29–31].

In this work, as opposed to other state-of-the-art solutions, the data-parallel and scalable approach of the modified FUN algorithm enables the implementation on a multi-SoPC cluster. As such, the cluster emulates a satellite constellation, where the hyperspectral linear unmixing is done on the edge (avoiding the downlink transmission overhead).

2.2. Distributed On-Board Processing

In the last few years, on-board processing has become a remarkable research trend, mainly motivated by the limited bandwidth of the communication links between satellites and Earth facilities. In parallel, the search for highly efficient and fast solutions for edge processing has also been a hot spot in the field. Activities combining both research lines can be found in the literature. For instance, a high-efficiency system based on Zynq SoPC devices for hyperspectral image classification was presented in Reference [32]. Other on-board application scenarios have covered Synthetic Aperture RADAR (SAR) image processing [33,34] or Support Vector Machines (SVM) for cloud detection [35].

The appearance of CubeSats can be also considered a major breakthrough in the field. The fact that these devices are usually low-cost, and therefore limited in performance, has encouraged the research around Space Information Networking (SIN), where on-board processing in satellite applications is enhanced by the clustering of satellites [12]. With this approach, satellites operate as nodes of the same network, sharing resources in a distributed-computing approach.

One of the main challenges in this type of solutions is to achieve scalable behavior when adding more processing nodes to the system. In this regard, communications play an important role, since their overheads may not be negligible. Different satellite network architectures have been proposed taking into account efficiency and flexibility regarding new technologies, rendering overheads in the order of nanoseconds [36,37]. Moreover, satellite communications have been also envisioned for FPGA usage [38], being flexibility, reconfigurability and reliability the main parameters assessed.

Going back to the on-board processing capabilities of CubeSats, the use of hybrid and reconfigurable computing has been found the most suitable, especially when taking into account factors such as power consumption, price, reliability, and flexibility. In the literature, it is possible to find many research works related to on-board processing in SmallSats, with applications ranging in a wide spectrum that covers, for instance, signal processing for interferometric satellites using FPGAs [39], or machine learning for image processing on a Xilinx FPGA [40]. Nevertheless, there is still a need for scalable execution in satellite clusters.

With regard to alternative state-of-the-art solutions, the proposed implementation provides not only low cost, reduced energy consumption, and flexibility, but also scalability at both the device and node level. These features are required for the distributed on-board processing scenario, and have been enhanced by using the ARTICo³ architecture, which provides a run-time adaptable working point trading off between energy consumption, performance, and fault tolerance.

3. Technology Background

In this section, descriptions of both the linear unmixing algorithm and the hardware-based processing architecture used in this work are presented.

3.1. FUN

Linear unmixing is a key processing technique for hyperspectral images. It assumes that the effect of secondary reflections and scattering effects are negligible and, thus, each pixel in the image can be represented as a linear combination of a subset of elements whose spectrum is considered pure. These elements are called endmembers, and their presence in each pixel of the image is weighted by the so-called abundances. The main purpose of linear unmixing algorithms is to obtain both the endmembers and their abundances from a hyperspectral image.

The Fast UNmixing (FUN) algorithm [19] allows the simultaneous estimation of the number of endmembers and endmember extraction. The FUN algorithm selects the first endmember as the pixel of the hyperspectral image with the largest orthogonal projection to the centroid pixel. The centroid pixel is an artificial pixel that averages all the information present in the image. Afterwards, it sequentially performs orthogonal projections of the hyperspectral image in the direction spanned by the last extracted endmembers. This process is performed in such a way that information of the hyperspectral image that can be represented by the already-extracted endmembers is subtracted from the image, and the pixel with more remaining information is selected as the next endmember. After selecting each endmember, the FUN algorithm estimates the percentage of information that cannot be represented with the already selected endmembers using stop factor s . If the value obtained is smaller than input threshold α , the algorithm finishes, thus obtaining the total number of endmembers present in the image plus the extracted endmembers.

The orthogonal projections of the hyperspectral image can be obtained using different methods. The FUN algorithm employs a modified version of the Gram–Schmidt method, which features low computational complexity and allows the reuse of previously computed information, speeding up the overall process, without using too-complex matrix calculations.

The FUN algorithm, as other geometrical approaches to hyperspectral unmixing (e.g., OSP [41], N-FINDR [42], VCA [43]), selects highly characteristic pixels as endmembers. In the presence of outliers, the set of extracted endmembers would contain actual endmembers (the algorithm would still find them) and, most probably, the outliers. Although this situation can be avoided when processing data in ground facilities (e.g., through direct supervision made by an expert), it becomes a critical issue to be addressed in on-board processing scenarios. In this context, a preprocessing stage is required to remove the outliers from input hyperspectral data (e.g., by using filtering or interpolation). In the rest of this paper, it is assumed that no outliers are present in the input datasets and, thus, no preprocessing stage is required.

The pseudocode shown in Algorithm 1 describes the process followed by the FUN algorithm to extract the endmembers. Lines 1 to 9 of this pseudocode correspond to the initialization of the algorithm by selecting the first endmember, e_1 . In Line 12, the information of each pixel of the image that can be represented by the last extracted endmember is subtracted from the image. Hence, x_i contains the information of the i -pixel that cannot be represented by the already-extracted endmembers. The amount of information remaining in each pixel, s_i^2 , is measured in Line 13. The pixel with the maximum amount of remaining information is the next candidate to be endmember. The index of this pixel, i_{\max} , is calculated in Line 15 of this pseudocode. After doing so, the amount of remaining information in this pixel is used for evaluating the stopping condition, as described in Line 16. If the amount of remaining information is high according to the stopping condition, and more endmembers are required, the i_{\max} -pixel is selected as the next endmember (Lines 19 to 22) and the process is repeated. Otherwise, the process finishes (Line 17). Please note that the stopping condition is the same as the one presented in Reference [19], but implemented in a more computationally efficient way, as proposed in Reference [25].

After extracting the endmembers, the FUN algorithm computes their abundances. This process is done by generating an orthonormal set of vectors using the obtained endmembers as the starting point. The pseudocode shown in Algorithm 2 describes the process followed by the FUN algorithm to obtain the abundances. Gram–Schmidt orthogonalization is performed P times (lines 5 to 19) over the endmembers, using a different order each time, to fill the U matrix (line 16). Abundances are then computed by projecting the hyperspectral image using the orthonormalized vectors contained in U (line 20). It is important to highlight that not only Algorithm 1 (as seen in the previous paragraph), but also Algorithm 2 corresponds to the computationally efficient implementation of the FUN algorithm presented in Reference [25]. In this implementation, several functions were optimized to reduce the number of complex and time-consuming mathematical operations. For instance, the indexing of the endmembers in Line 9 is simplified using a linear access (i.e., $x \leftarrow e_{k+j-1}$) instead of using integer

division, even though it slightly increases memory overhead by replicating endmember matrix E in Line 1.

Algorithm 1 FUN algorithm: endmember extraction

Inputs:

$$M = [m_1 \dots m_{pixels}]$$

▷ Input hyperspectral image

 α

▷ Stop factor

Outputs:

$$E = [e_1 \dots e_P]$$

▷ Endmembers

 P

▷ Number of endmembers

```

1:  $X \leftarrow M$ ;  $X = [x_1 \dots x_{pixels}]$                                 ▷ Auxiliary copy of the hyperspectral image
2:  $E \leftarrow []$                                                     ▷ Endmembers matrix
3:  $Q \leftarrow []$                                                     ▷ Gram Schmidt orthogonalization of Endmembers
4:  $U \leftarrow []$                                                     ▷ Gram Schmidt orthogonalization (normalized) of Endmembers
5:  $e_1 \leftarrow x_{init}$ ;  $E \leftarrow [E \ e_1]$                             ▷ Select first endmember according to initialization criteria
6:  $q_1 \leftarrow e_1$ ;  $Q \leftarrow [Q \ q_1]$ 
7:  $u_1 \leftarrow e_1 / (e_1 \bullet e_1)$ ;  $U \leftarrow [U \ u_1]$ 
8:  $P \leftarrow 1$ 
9:  $exit \leftarrow 0$ 
10: while  $exit = 0$  do
11:   for  $i \leftarrow 1$  to  $pixels$  do
12:      $x_i \leftarrow x_i - (x_i \bullet q_P) \cdot u_P$ 
13:      $s_i^2 \leftarrow x_i \bullet x_i$ 
14:   end for
15:    $(s_{max}^2, i_{max}) \leftarrow \text{GETMAX}(S^2)$ ;  $S^2 = [s_1^2 \dots s_{pixels}^2]$ 
16:   if  $s_{max}^2 \cdot 100^2 \leq \alpha^2 \cdot (m_{i_{max}} \bullet m_{i_{max}})$  then
17:      $exit \leftarrow 1$ 
18:   else
19:      $P \leftarrow P + 1$ 
20:      $e_P \leftarrow m_{i_{max}}$ ;  $E \leftarrow [E \ e_P]$ 
21:      $q_P \leftarrow x_{i_{max}}$ ;  $Q \leftarrow [Q \ q_P]$ 
22:      $u_P \leftarrow x_{i_{max}} / (x_{i_{max}} \bullet x_{i_{max}})$ ;  $U \leftarrow [U \ u_P]$ 
23:   end if
24: end while

```

Algorithm 2 FUN algorithm: abundances computation

Inputs:	
$M = [m_1 \dots m_{pixels}]$	▷ Input hyperspectral image
$E = [e_1 \dots e_p]$	▷ Endmembers
P	▷ Number of endmembers
Outputs:	
$A = [a_1 \dots a_{pixels}]$	▷ Abundances
1: $E \leftarrow [E \ E]$	
2: $Q \leftarrow []$	
3: $U \leftarrow []$	
4: $U^* \leftarrow []$	
5: for $k \leftarrow 2$ to $P + 1$ do	
6: $q_1 \leftarrow e_k$; $Q \leftarrow [Q \ q_1]$	
7: $u_1^* \leftarrow e_k / (e_k \bullet e_k)$; $U^* \leftarrow [U^* \ u_1^*]$	
8: for $j \leftarrow 2$ to P do	
9: $x \leftarrow e_{k+j-1}$	
10: for $i \leftarrow 1$ to $j - 1$ do	
11: $x \leftarrow x - (x \bullet q_i) \cdot u_i^*$	
12: end for	
13: $q_j \leftarrow x$; $Q \leftarrow [Q \ q_j]$	
14: $u_j^* \leftarrow x / (x \bullet x)$; $U^* \leftarrow [U^* \ u_j^*]$	
15: end for	
16: $U \leftarrow [U \ u_p^*]$	
17: $Q \leftarrow []$	
18: $U^* \leftarrow []$	
19: end for	
20: $A \leftarrow U^t \cdot M$	

3.2. ARTICO³

ARTICO³ [18] is a hardware-based high-performance embedded processing architecture that enables user-driven adaptation at runtime, creating a dynamic solution space in which tradeoffs between computing performance, energy consumption, and fault tolerance can be established. The architecture relies on the use of Dynamic and Partial Reconfiguration (DPR) in Xilinx FPGAs to provide software-like flexibility while maintaining hardware-like performance during execution. The top-level block diagram of the ARTICO³ architecture is shown in Figure 1.

ARTICO³-based computing platforms work in a processor–coprocessor scheme, where the application code is executed in the host microprocessor, and only those program sections that exhibit significant levels of data parallelism (called kernels) are offloaded to the hardware accelerators. This execution model, based on data independences between blocks, provides transparent scalability in terms of computing performance when coupled with DPR-based hardware replication, since the available number of processing elements (i.e., hardware accelerators) for a given kernel can be altered on demand even during application execution. In addition, the architecture can benefit from module

replication not only to increase computing performance (different blocks working on different data, SIMD-like execution), but also to increase fault tolerance using DMR or TMR (different blocks working on the same data, with an embedded voter unit to mask faults).

The ARTICo³ architecture is part of a framework that also includes a toolchain to transparently generate dynamically reconfigurable systems from the descriptions of both hardware accelerators and host application. The design-time support of the framework requires users to provide an already partitioned hardware/software system, where host code is specified in C/C++ and kernels are specified in low-level HDL (VHDL, Verilog) or C/C++ to be used with High-Level Synthesis (HLS) tools. Using these elements as inputs, the toolchain automatically performs three tasks: instantiates the user-defined kernel logic in a standard wrapper, generates the on-chip DMA-powered communication infrastructure, and builds both hardware and software components to obtain the required binaries that are used in the target platform.

From the programming point of view, user code interfaces with the accelerators using a runtime library, which transparently handles (and thus, hides from the user) two complex processes: FPGA reconfiguration, and parallel execution management. This is supported by the ARTICo³ programming model, which is based on a reduced API to favor user-friendly reconfigurable computing.

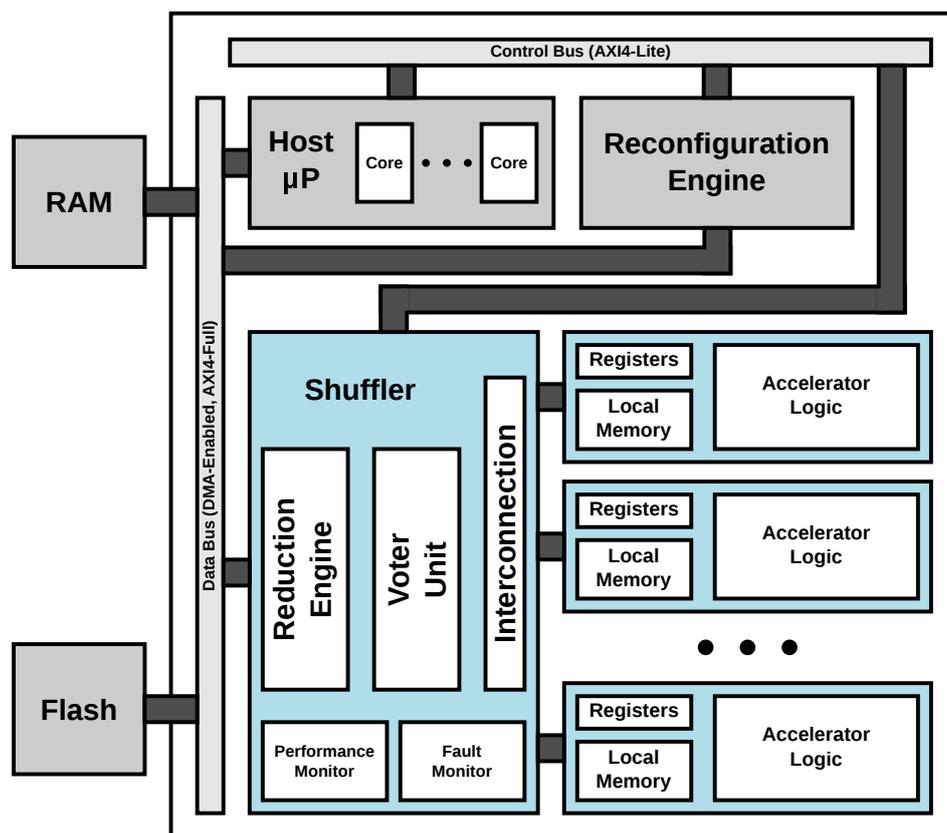


Figure 1. Top-level block diagram of the ARTICo³ architecture.

4. Implementation Details

As stated in previous sections, ARTICo³-based high-performance embedded computing relies on user-driven application partitioning in sequential host code and data-parallel hardware kernels. As a result, the first step that needs to be addressed is the profiling of the application in order to identify and extract potential data-level parallelism. A sequential C-based and single-core implementation of the original FUN algorithm has been developed and analyzed in a Zynq-7000 device. The obtained results can be seen in Table 1, and show that, among the two parts of the linear

unmixing algorithm (i.e., endmember extraction and abundances computation), endmember extraction is the most time-consuming (75%).

Table 1. FUN-based hyperspectral image unmixing: application profiling (Zynq-7000 ARM Cortex-A9 core @ 666.67 MHz).

	Endmember Extraction	Abundance Computation
Execution Time (ms)	1723.31	552.44
Execution Fraction (%)	75.4	24.6

Taking these results as starting point, it seems reasonable to focus on optimizing the endmember extraction process. The rest of this section details the proposed modifications to the algorithm to exploit data-level parallelism (as required by ARTICo³ itself), the design-time decisions made to balance data precision, execution performance, and area overhead in the generated HLS-based hardware accelerators (to fit in resource-constrained ARTICo³ slots and still provide accurate results), and the extension from single-node to networked multi-FPGA solutions (to add a second level of scalability to the hardware-accelerated deployment).

4.1. Parallelization Approach

The original FUN algorithm uses the full hyperspectral image to extract the underlying endmembers. In the reference C code, this fact generates huge overheads in terms of processing latency (clock cycles), since several operations are implemented as nested loops that depend on the size of the input hyperspectral cube. Moreover, this dependency also imposes huge memory requirements that, for large hyperspectral images, may result in nonfeasible hardware-accelerated solutions that do not fit within the available FPGA resources.

In order to not only mitigate these problems, but also to enable data-level parallelism (and thus potentially scalable execution), a reduction-based parallelization approach has been proposed for the original algorithm, and it is one of the main contributions of this paper. First, the input hyperspectral image is split in fixed-size hyperspectral subimages. The total number of subimages may change, since it depends on the size of the input image. Then, the FUN algorithm is used independently in each of those subimages to extract partial endmembers. An iterative process follows in which fixed-size artificial hyperspectral subimages are made up from the pool of resulting partial endmembers, and the FUN algorithm is used again to achieve dimensional reduction, up to the point where only one fixed-size artificial subimage remains, and the actual endmembers are obtained. In summary, the main idea of the proposed approach is to iteratively extract endmembers from the endmembers in the partial subimages until all the actual ones have been found.

The pseudocode of the reduction-based FUN algorithm for endmember extraction can be seen in Algorithm 3. Notice that, when the same endmembers are extracted in two consecutive rounds (i.e., there is no dimensional reduction), the algorithm forces the generation of artificial subimage blocks with double the size while keeping the maximum number of endmembers that have to be extracted per block. This failsafe mechanism, which is meant to solve situations where the same endmembers are present in two or more data-independent subimages, ensures no deadlocks occur during the reduction process. Note also that, in any other situation and in order to avoid information loss, the maximum number of endmembers that can be extracted from each subimage block during the reduction process equals the number of pixels in that subimage. To better understand this, consider a scenario where all endmembers are present in one of the initial subimages: the algorithm needs to allow all these pixels to reach the final reduction round.

Algorithm 3 Reduction-based FUN algorithm: endmember extraction

Inputs:	
$M = [m_1 \dots m_{pixels}]$	▷ Input hyperspectral image
P_{\max}	▷ Maximum number of endmembers to extract
α	▷ Stop factor
$pixels_{\text{block}}$	▷ Number of pixels per parallel block
Outputs:	
$E = [e_1 \dots e_p]$	▷ Endmembers
P	▷ Number of endmembers

1:	$P \leftarrow 0$	
2:	$[X_1 \dots X_N] \leftarrow \text{SPLIT}(M, pixels_{\text{block}})$	▷ Divide input image in subimages
3:	while $N > 1$ do	▷ Repeat until there is only one subimage block remaining
4:	$E \leftarrow []$	
5:	for $i \leftarrow 1$ to N do	
6:	$E_{\text{aux}} \leftarrow \text{ENDMEMBERS}(X_i, pixels_{\text{block}}, \alpha)$	▷ Extract endmembers from subimage
7:	$E \leftarrow [E \ E_{\text{aux}}]$	▷ Generate artificial image
8:	end for	
9:	$P_{\text{aux}} \leftarrow \text{SIZE}(E)$	
10:	if $P_{\text{aux}} = P$ then	▷ Avoid deadlock (no dimensional reduction)
11:	$[X_1 \dots X_N] \leftarrow \text{SPLIT}(E, 2 \cdot pixels_{\text{block}})$	▷ Divide artificial image in subimages
12:	else	
13:	$[X_1 \dots X_N] \leftarrow \text{SPLIT}(E, pixels_{\text{block}})$	▷ Divide artificial image in subimages
14:	end if	
15:	$P \leftarrow P_{\text{aux}}$	
16:	end while	
17:	$E \leftarrow \text{ENDMEMBERS}(E, P_{\max}, \alpha)$	▷ Extract actual endmembers
18:	$P \leftarrow \text{SIZE}(E)$	▷ Get number of actual endmembers

The accuracy of the proposed algorithm has been evaluated in two scenarios: on the one hand, using synthetic hyperspectral images (256 bands, 128 lines, 128 samples) with different noise levels, number of endmembers, and abundance distributions (see Table 2); on the other hand, using well-known real hyperspectral datasets [44] (see Table 3). In both scenarios, $pixels_{\text{block}}$ has been set to 32, which is the maximum number of pixels that can be stored in the local memory inside ARTICo³ accelerators. An experimental evaluation (see Table 4) shows that the impact of $pixels_{\text{block}}$ on the extraction effectiveness is negligible when this parameter is large enough (i.e., when $pixels_{\text{block}} \geq 16$). The distance between extracted and real endmembers has been measured using the spectral angle, which can be calculated using Equation (1):

$$\text{Spectral Angle} = \arccos \frac{e_{\text{real}} \bullet e_{\text{extracted}}}{\|e_{\text{real}}\| \cdot \|e_{\text{extracted}}\|} \quad (1)$$

where e_{real} represents known endmembers (a pixel with all its spectral bands) and $e_{\text{extracted}}$ represents endmembers obtained with the algorithm. Perfect matches between real and extracted endmembers

would render a spectral angle of 0° , whereas for completely unrelated pixels, the spectral angle would tend to 90° . It is important to note that, for real hyperspectral images, if the algorithm finds P endmembers and the image is known to have only P_{real} , with $P_{\text{real}} < P$, the spectral angle is computed as the minimum value from all possible combinations (i.e., the extracted P_{real} endmembers that are closer to the real ones).

As it can be seen, the reduction-based endmember extraction provides results that are close to the ones obtained using the original FUN algorithm, which in turn are not far from the actual endmembers present in the image. A comparison with state-of-the-art alternatives (Table 3) shows that deviations in the spectral angle are acceptable either because it is still below the value obtained with some of the most widely used unmixing algorithms (e.g., VCA or NMF), or because (even with comparable results) the algorithm is too complex to enable a runtime adaptive hardware implementation as the one presented in this work (e.g., DgS-NMF).

Table 2. Endmember extraction accuracy (synthetic hyperspectral images with known endmembers).

Image Characteristics			Mean Spectral Angle ($^\circ$)		
SNR (dB)	Endmembers	Abundances	FUN	Reduction-Based FUN	
20	4	Gauss Spheric	1.5115	2.6709	
		Legendre Polynomial	1.5433	1.4739	
		Dirichlet	3.7219	2.0508	
	8	Gauss Spheric	2.8381	2.4991	
		Legendre Polynomial	3.6176	3.8983	
		Dirichlet	2.5348	2.7655	
	12	Gauss Spheric	3.7033	5.4387	
		Legendre Polynomial	5.3731	5.4038	
		Dirichlet	2.8855	4.3962	
	40	4	Gauss Spheric	0.0798	0.2163
			Legendre Polynomial	0.3327	0.1521
			Dirichlet	0.1254	0.4491
8		Gauss Spheric	0.4679	0.3806	
		Legendre Polynomial	0.6072	0.8048	
		Dirichlet	0.3019	1.2459	
12		Gauss Spheric	0.4252	1.1471	
		Legendre Polynomial	0.3992	0.3853	
		Dirichlet	0.3895	0.8428	
60		4	Gauss Spheric	0.0658	0.0726
			Legendre Polynomial	0.0367	0.2022
			Dirichlet	0.0682	0.0895
	8	Gauss Spheric	0.0459	0.0936	
		Legendre Polynomial	0.0737	0.0975	
		Dirichlet	0.1463	0.1672	
	12	Gauss Spheric	0.0876	0.0860	
		Legendre Polynomial	0.1705	0.1953	
		Dirichlet	0.1384	0.2409	

Table 3. Endmember extraction accuracy (real hyperspectral images with known endmembers).

Image	Endmembers	Mean Spectral Angle ($^\circ$)				
		FUN	Reduction-Based FUN	VCA [44]	NMF [44]	DgS-NMF [44]
Samson	3	3.7004	2.6444	4.8301	5.1337	2.8934
Jasper Ridge	4	7.5999	7.0384	19.8759	10.1069	3.0997
Urban	4	4.4637	7.6292	23.2564	10.8633	4.8988
Cuprite	12	5.9887	5.637	–	7.5401	5.9931

Table 4. Impact of partitioning depth ($pixels_{block}$) on endmember-extraction accuracy (evaluation based on synthetic images with 256 bands, 256 lines, 256 samples, and 60 dB of SNR).

Image Characteristics		Mean Spectral Angle (°)						
Endmembers	Abundances	4	8	16	32	64	128	256
4	Gauss Spheric	0.2329	0.1933	0.1933	0.1933	0.1933	0.1933	0.1933
	Legendre Polynomial	3.3223	1.0612	0.1554	0.0682	0.0547	0.0546	0.0543
	Dirichlet	0.0595	0.0592	0.0592	0.0592	0.0592	0.0592	0.0592
16	Gauss Spheric	–	–	0.0838	0.0871	0.1374	0.1374	0.0875
	Legendre Polynomial	–	–	0.7113	0.703	0.7154	0.7149	0.6541
	Dirichlet	–	–	0.0838	0.0852	0.0852	0.085	0.085

4.2. Hardware Tradeoffs

The reduction process detailed in Algorithm 3 has a sequential component (i.e., dimensional reduction) and a data-parallel section (i.e., endmember extraction for each subimage, Lines 5 to 8), which has been selected as the functionality to be moved to hardware. As a consequence of this decision, the transparent scalability provided by the ARTICo³ architecture, and enabled by the use of a configurable number of hardware accelerators, can be used to dynamically adapt the loop unrolling depth during application execution.

While the ARTICo³ toolchain supports both HDL and C/C++ descriptions of the kernels, only the HLS-based entry point has been used to implement the FUN endmember extraction, since a C-based version of the whole linear unmixing chain was already available (i.e., the one used during the initial application profiling stage). As stated in the previous section, low-level FPGA limitations make it mandatory to develop resource-constrained, yet performance-oriented, hardware accelerators. In the following, two different HLS-based solutions are analyzed taking into account their execution latency and resource utilization. Both solutions have been implemented using Vivado HLS with the configuration parameters shown in Table 5.

The first solution uses single-precision (i.e., 32-bit) floating-point arithmetic, and relies on tool-based automatic optimizations (mainly, datapath pipelining). The second solution, on the other hand, uses half-precision (i.e., 16-bit) floating-point arithmetic, and relies on tool-based automatic optimizations to improve performance (datapath pipelining) as well as on user-driven code optimization techniques to improve the performance/area ratio (manual loop unrolling). Experimental results showed no functional difference between both solutions when processing normalized input subimages (i.e., whose pixel values range from 0 to 1), since the same endmembers were obtained as output.

Table 6 shows the accelerator latency bounds for both solutions. It is important to highlight that the latency of the accelerator is data-dependent (although it will always be between the reported bounds), since the algorithm contains a loop whose trip count depends on the number of pixels with relevant information. Hence, the results shown in Table 6 refer either to an execution that finds one endmember (minimum value) or to an execution that finds $pixels_{block}$ endmembers (maximum value). Table 7, on the other hand, shows the resource-utilization reports. Note that these reports include: combinational logic expressed as Look-Up Tables (LUTs); sequential logic expressed as Flip-Flops (FFs); dedicated logic for arithmetic operations, expressed as Digital Signal Processing blocks (DSPs); and memory elements, expressed as Block RAM (BRAMs). When comparing both alternatives, it is possible to see that the combination of half-precision floating-point arithmetic and manual loop unrolling reduces the memory footprint, but increases the rest of the resources. However, both implementations can fit in regular ARTICo³ slots, and the slightly superior resource utilization of the half-precision implementation is an affordable cost given the performance increase that can be achieved (roughly speaking, 2× resources generate 4× performance).

Table 5. High-Level Synthesis (HLS)-Based hardware accelerator: configuration parameters.

Parameter	Description	Value
$pixels_{block}$	Number of input pixels	32
N_z	Number of bands per pixel	256
P_{max}	Maximum number of endmembers to extract	32
α	Stop factor	1.0

Table 6. Accelerator latency: single-precision versus half-precision floating-point implementations.

	Implementation #1	Implementation #2
Precision	Single—32 bits	Half—16 bits
Optimization	Automatic (directives)	Automatic (directives) + Manual (code)
Latency (cycles)	200331—1 endmember 1588512—32 endmembers	50917—1 endmember 384509—32 endmembers

Table 7. Resource utilization: Single precision versus Half precision floating point implementations.

	Implementation #1	Implementation #2
Precision	Single—32 bits	Half—16 bits
Optimization	Automatic (directives)	Automatic (directives) + Manual (code)
LUTs	3089	4296
FFs	2612	4172
DSPs	10	18
BRAMs	8.5	4

4.3. Network Infrastructure

The ARTICo³ architecture provides performance scalability on a single node, relying on the application developer to decide how many accelerators to load for a given kernel. In order to extend the scalability to a multi-FPGA context, thus enabling a two-way scalable system, a networked approach has been implemented. The proposed solution is built upon a small cluster of parallel-processing elements, referred to as nodes, in a master/slave(s) approach (see Figure 2), where data-level parallelism can be further exploited by dividing the computational workload between all available nodes.

In this work, data distribution and execution synchronization are made using MPI, a well-known parallel programming API used in high-performance systems with distributed memory, which has also been used in the past to prototype communications in space [45]. However, it is important to highlight that the proposed distribution methods can be implemented using any other type of communication primitives. Regarding data distribution, two different methods have been proposed and evaluated: the application developer can decide on whether to maximize parallelism, or to minimize data transactions (communication packets). Both alternatives start with the master dividing the input hyperspectral image in as many blocks as slaves are involved in the processing stage, but they differ in the next steps.

In the maximum parallelization approach, each slave further divides the image in blocks whose size can be directly processed by the core reduction algorithm ($pixels_{block}$). Then, partial endmembers are extracted from each of these blocks. The whole set of partial endmembers are then sent back to the master, which regroups them before sending new artificial image blocks to the slaves for another round of endmember extraction. This procedure continues until one block remains and all endmembers are finally obtained. In the minimum data transactions approach, on the other hand, each slave performs a local reduction on its own, only sending back to the master the reduced endmembers contained in the corresponding input subimage. The master then performs a final reduction round to obtain the actual endmembers present in the original hyperspectral image.

Although the maximum parallelization approach should ideally be better, communication latency discourages its use due to the excessive data traffic over the network. As it can be seen in Figure 3, the approach that targets minimum data transactions between master and slave nodes has provided better experimental results (100 ms faster on average). All network experiments performed in this work use the MPICH implementation of the MPI API, since it has already been proven that it renders better communication performance than other alternatives such as OpenMPI [46].

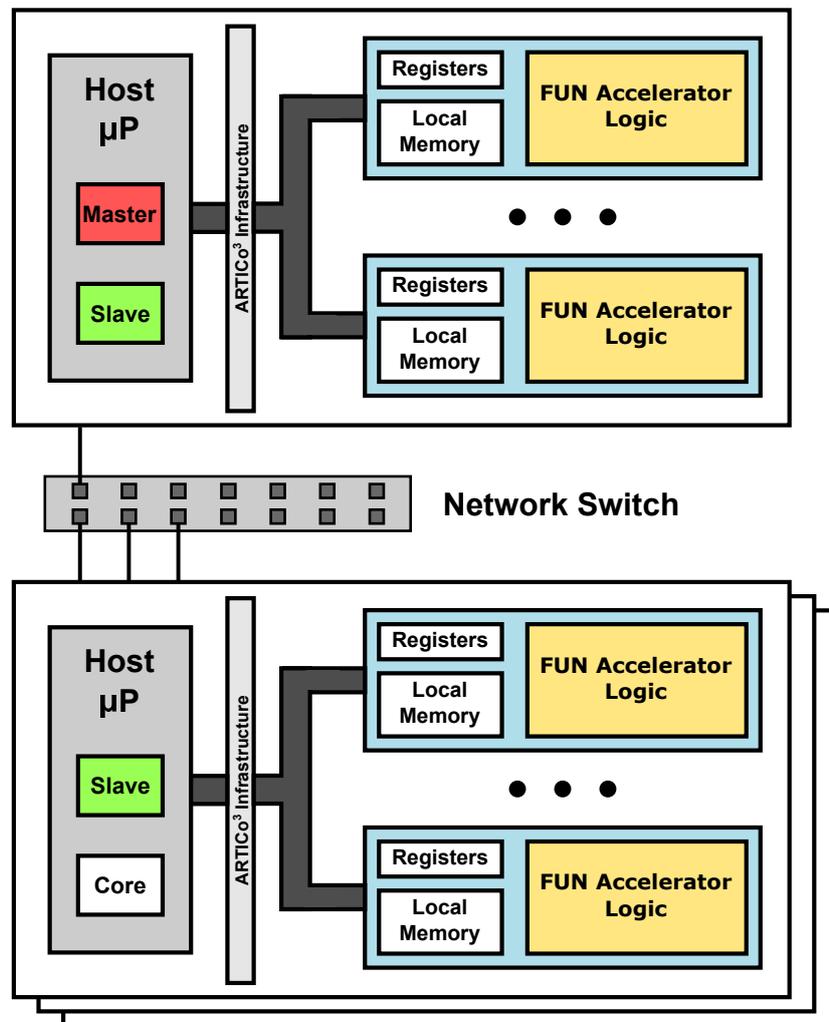


Figure 2. Network setup: ARTICo³-based computing cluster.

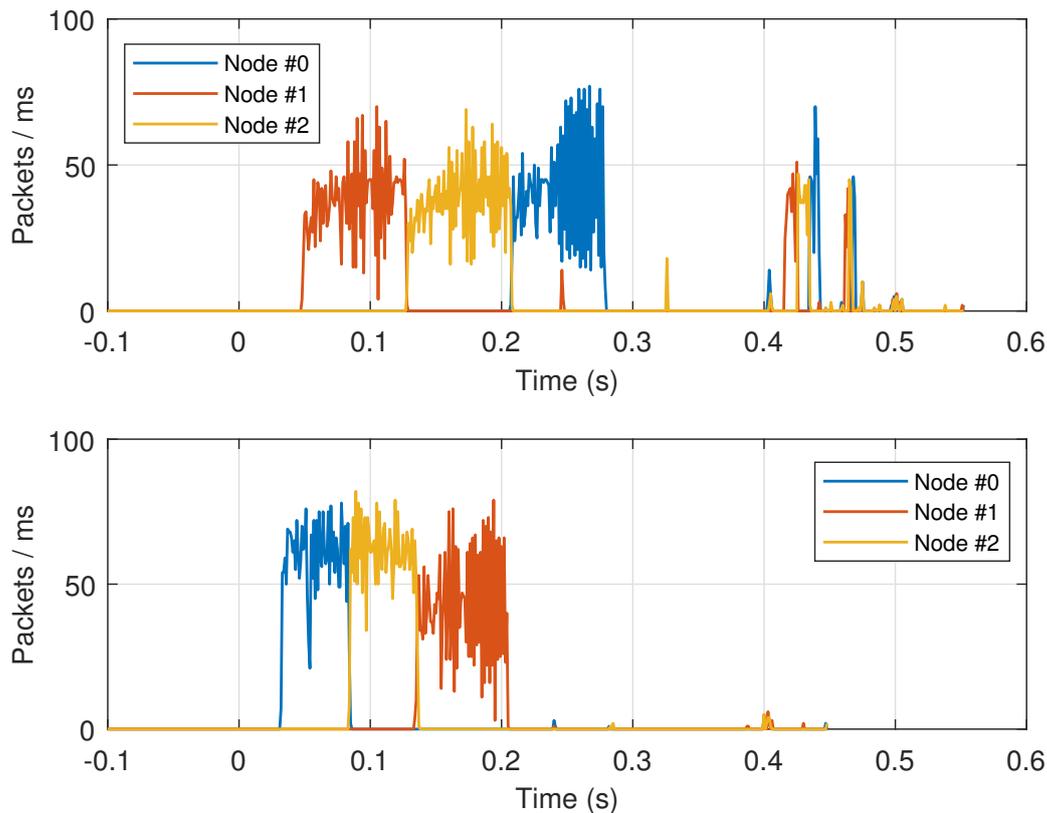


Figure 3. MPI-based communication overheads (four nodes, hyperspectral images with 128 samples, 128 lines and 256 bands). The top graph shows maximum parallelization, whereas the bottom graph shows minimum data transactions (i.e., communication). The cluster operates on a private network with a dedicated Ethernet switch (1 Gbps).

5. Experimental Results

Two experimental setups have been used to evaluate the runtime scalability of the ARTICo³-based endmember extraction. Single-node scalability has been tested using an in-house custom Zynq-7000 board (XC7Z020-1CLG484) with integrated power measurement circuitry and thus, energy efficiency metrics have also been obtained. Multi-FPGA scalability, on the other hand, has been evaluated only in terms of computing performance using up to 8 commercial MicroZed development boards (XC7Z020-1CLG400) arranged in a small Ethernet-based computing cluster. Both scenarios implement the reduction-based FUN endmember extraction using half-precision floating point hardware accelerators and, in the multi-FPGA setup, the data distribution approach for minimum communication over the network.

In addition, all tests reported in this section have been performed using three synthetic hyperspectral images, since they enable the evaluation of a wider range of scenarios (e.g., different number of endmembers, different input sizes, etc.) than any of the represented by the real hyperspectral datasets used to validate the parallelization approach (see Section 4.1): one with a size of $256 \times 128 \times 128$ and 10 endmembers, one with a size of $256 \times 256 \times 256$ and 16 endmembers, and a final one with a size of $256 \times 512 \times 512$ and 16 endmembers. The size of the input hyperspectral images is expressed as $N_z \times N_y \times N_x$, being z bands, y lines, and x samples.

Table 8 reports the resource utilization for the ARTICo³ infrastructure (per node), as well as for the final ARTICo³ kernel (FUN + wrapper logic per accelerator). Since this kernel does not require

any configuration register, the overhead introduced by wrapping the HLS-generated HDL is almost negligible in terms of LUTs and FFs, while there is a sharp increase in the BRAM count due to the local memory inside the hardware accelerator.

Table 8. Resource utilization: ARTICo³ infrastructure overhead versus FUN kernel.

Component	ARTICo ³	FUN Kernel
Info	–	64 KiB memory 0 registers C + HLS
LUTs	4158	4502
FFs	2366	4207
DSPs	–	18
BRAMs	–	20

5.1. Standalone ARTICo³

The ARTICo³ solution space for single-node deployments represents all possible combinations of computing performance, energy consumption, and fault-tolerance level (i.e., hardware redundancy) when changing the number of accelerators and their configuration for a given kernel. However, the results presented in this section are only focused on computing performance and energy consumption.

Table 9 shows the obtained results for the single-node scenario, where a single-core software version of the reduction-based FUN algorithm has also been implemented to complement the analysis. As it can be seen, for small hyperspectral images, the software version is equivalent (in terms of computing performance) to the ARTICo³-powered one using a single hardware accelerator. However, energy consumption is almost 50% less for the hardware-accelerated solution. For large hyperspectral images, on the other hand, the ARTICo³-based solution outperforms the software in terms of computing performance (even when having only one accelerator), while maintaining a good energy-efficiency ratio.

Although execution time is reduced when increasing the number of hardware accelerators for a fixed input image, it is possible to notice that the scalability factor is not linear. This is due to the combination of kernels changing from computing- to almost memory-bounded behavior (in the solution space), and the memory-management overheads introduced by the ARTICo³ runtime. Nevertheless, energy consumption is also reduced, a fact that would still motivate the use of more hardware accelerators for processing in a real-world scenario.

Table 9. Execution performance and energy consumption—ARTICo³ (single node @ 166.67 MHz) versus software-based implementation (single-node, 1 ARM core @ 666.67 MHz).

Nz × Ny × Nx	Execution Time (s)					Energy Consumption (J)				
	SW	1 acc	2 accs	3 accs	4 accs	SW	1 acc	2 accs	3 accs	4 accs
256 × 128 × 128	1.05	1.05	0.75	0.66	0.58	1.44	0.98	0.79	0.76	0.75
256 × 256 × 256	4.33	4.35	3.11	2.66	2.38	5.88	3.95	3.09	2.85	2.65
256 × 512 × 512	69.44	46.4	29.12	22.97	19.75	92.99	38.42	27.17	23.16	21.2

5.2. Networked ARTICo³

In this second scenario, the small-size high-performance embedded computing cluster has been used. The cluster has eight nodes, each of them able to host one ARTICo³ instance with up to four hardware accelerators. As a result, it is possible to evaluate the two-way scalability of the system (changing the number of nodes, or changing the number of hardware accelerators per node). Figure 4 shows a picture of the Ethernet-based multi-FPGA embedded computing cluster.

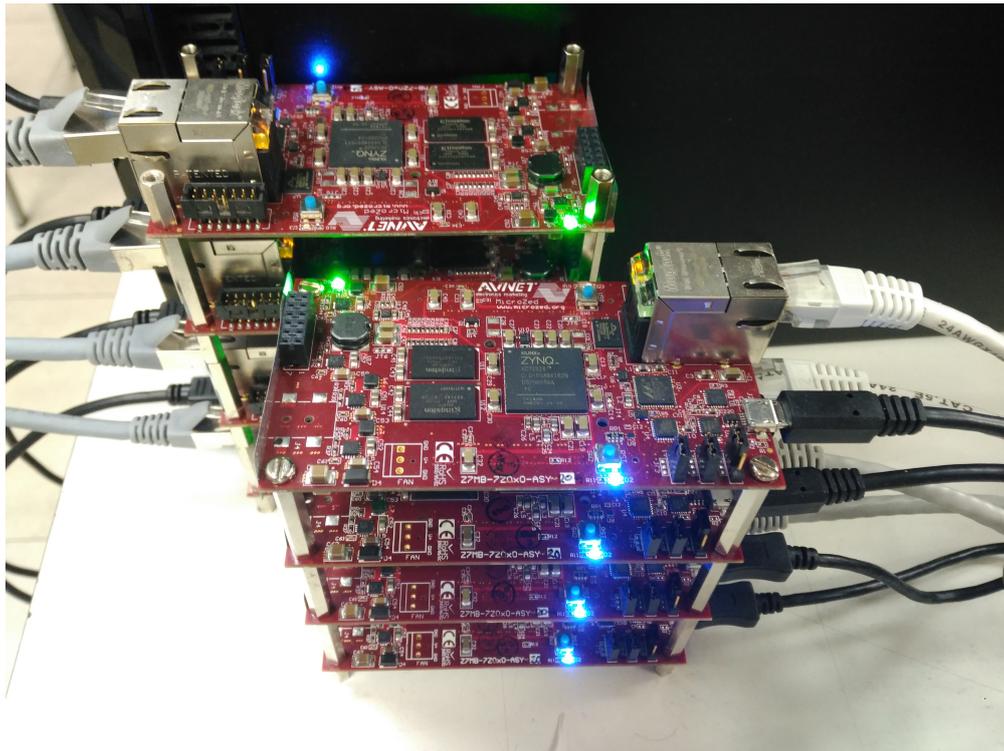


Figure 4. ARTICo³-based multi-FPGA computing cluster. This high-performance embedded computing setup features eight MicroZed boards with a XC7Z020-1CLG400C device.

The first set of tests performed with the cluster aims to show the aforementioned two-way scalability of the system. Table 10 reports the execution times for different configurations of input image sizes, number of ARTICo³ nodes, and number of hardware accelerators per node. It also reports, for each image size, the overall improvement of each configuration with respect to the solution with one node and one hardware accelerator (1×). These results show that, for the same number of hardware accelerators, it is preferable to favor scalability in the number of nodes rather than in the number of accelerators per node, since the per-node memory-management overhead is slightly larger than the internode communication overhead. Notice that execution times when using one node, a scenario that should be similar to the one reported in the previous section, are slightly larger due to the overhead generated by the communication API.

Table 10. Execution performance—ARTICo³ (multiple nodes @ 166.67 MHz).

Nz × Ny × Nx	# Nodes	Execution Time (s)			
		1 acc	2 accs	3 accs	4 accs
256 × 128 × 128	1	1.18 (1×)	0.84 (1.4×)	0.73 (1.6×)	0.67 (1.8×)
	2	0.72 (1.6×)	0.56 (2.1×)	0.49 (2.4×)	0.44 (2.7×)
	4	0.46 (2.6×)	0.41 (2.9×)	0.37 (3.2×)	0.34 (3.5×)
	8	0.38 (3.1×)	0.31 (3.8×)	0.29 (4.1×)	0.27 (4.4×)
256 × 256 × 256	1	4.72 (1×)	3.35 (1.4×)	2.87 (1.6×)	2.66 (1.8×)
	2	2.83 (1.7×)	2.12 (2.2×)	1.88 (2.5×)	1.85 (2.6×)
	4	1.99 (2.4×)	1.51 (3.1×)	1.38 (3.4×)	1.45 (3.3×)
	8	1.4 (3.4×)	1.35 (3.5×)	1.15 (4.1×)	1.11 (4.3×)
256 × 512 × 512	1	49.04 (1×)	30.67 (1.6×)	24.29 (2×)	21.14 (2.3×)
	2	25.88 (1.9×)	16.9 (2.9×)	14.17 (3.5×)	12.37 (4×)
	4	14.69 (3.3×)	10.18 (4.8×)	8.56 (5.7×)	7.8 (6.3×)
	8	9.12 (5.4×)	6.78 (7.2×)	6.52 (7.5×)	5.55 (8.8×)

The second set of tests performed with the multi-FPGA setup has been devised with a twofold objective: on the one hand, to compare the hardware-accelerated solution with a software-based alternative; on the other hand, to quantify the communication overheads in the system.

Using MPI terminology, the hardware version uses one slave process with up to four ARTICo³ accelerators per node (with a master process running in one of the nodes), while the software version uses two slave processes per node to maximize the number of processing elements working potentially in parallel (although one node has the master process and one slave process instead). In addition, processing is carried out completely in the slave processes for the hardware version, while for the software version the master process also performs computations (again, to maximize the number of processing elements working potentially in parallel, and perform a fairer comparison against the hardware-based implementation). As a result, total execution times in Table 11 equal communication time plus processing time for software, but not for hardware (ARTICo³ finishes processing in each slave while the master is still sending or receiving data through the network). This phenomenon resembles the memory-bounded behavior in single-node deployments, where the memory bandwidth sets the maximum performance boundary for accelerators where DMA-enabled data transfers are more time-consuming than actual kernel execution.

Results from Table 11 show that, for small hyperspectral images, the ARTICo³-based solution is slower than the software-based alternative. However, this behavior is inverted when processing large images, even if communication time is still longer for the hardware-based solutions. It is important to highlight that, when only considering processing time, performance scales almost linearly for both ARTICo³-powered and software versions. This, together with the partitioning that generates smaller images to be processed in each node, makes it better to scale in the number of nodes than in the number of accelerators per node.

Table 11. Execution performance—ARTICo³ (multiple nodes, four accelerators @ 166.67 MHz per node) versus software-based implementation (multiple nodes, two ARM cores @ 666.67 MHz per node).

Nz × Ny × Nx	# Nodes	ARTICo ³			Software		
		Comm. (s)	Exec. (s)	Total (s)	Comm. (s)	Exec. (s)	Total (s)
256 × 128 × 128	1	0.06	0.59	0.65	0.08	0.35	0.43
	2	0.18	0.27	0.46	0.15	0.21	0.36
	4	0.29	0.14	0.34	0.22	0.09	0.31
	8	0.27	0.07	0.28	0.24	0.04	0.28
256 × 256 × 256	1	0.22	2.41	2.64	0.19	1.6	1.79
	2	0.74	1.12	1.83	0.64	0.83	1.47
	4	1.15	0.59	1.32	0.82	0.45	1.27
	8	1.07	0.3	1.09	0.91	0.26	1.17
256 × 512 × 512	1	0.89	19.36	21.25	0.44	25.93	26.37
	2	2.87	9.77	12.36	1.4	13.8	15.2
	4	4.52	4.94	7.87	3.69	6.63	10.32
	8	5.51	2.42	5.58	3.45	3.45	6.9

6. Conclusions and Future Work

In this paper, a two-way scalable and runtime-adaptive implementation of a linear unmixing algorithm for on-board hyperspectral image processing has been presented. This implementation deploys a modified version of the FUN algorithm with explicit data-level parallelism over a low-cost cluster of ARTICo³-powered SoPC that emulates a collaborative constellation of CubeSats.

Regarding the algorithm, the proposed modifications enable seamless parallelization without compromising functional correctness, as demonstrated by experimental evaluation. In addition, the hardware-oriented optimizations applied to the algorithm itself do not affect the functionality of the algorithm either.

From an architectural point of view, the proposed implementation scheme provides transparent hardware acceleration and management, with two degrees of freedom to dynamically adapt the working point in a solution space defined by computing performance, energy efficiency and fault tolerance. These three elements are key aspects in low-cost deployments of COTS-based CubeSats, making the proposed solution highly relevant in this context.

Experimental results demonstrate the feasibility of the proposed approach and widen the solution space of the accelerated application, providing an additional degree of freedom (number of network nodes used for processing) that can be used to maintain performance levels, while also enabling fault tolerance by means of hardware redundancy.

The communication infrastructure has proven to be one of the major bottlenecks in the proposed approach. Thus, an optimized communication infrastructure, paired with efficient data-transmission mechanisms, is envisioned as future activity to further enhance system performance. In the long run, and after optimizing network communications, the goal is to generalize the proposed platform to support general-purpose computing in constellation-based on-board processing. To this end, intelligent load balancing and distribution algorithms are to be implemented to transparently manage the system from the application developer's point of view.

Author Contributions: Conceptualization, A.O. (Alberto Ortiz), A.R., R.G., S.L., A.O. (Andrés Otero), R.S. and E.d.l.T.; Investigation, A.O. (Alberto Ortiz) and A.R.; Methodology, A.O. (Alberto Ortiz) and A.R.; Software, A.O. (Alberto Ortiz), A.R. and R.G.; Supervision, S.L., A.O. (Andrés Otero), R.S. and E.d.l.T.; Validation, A.O. (Alberto Ortiz) and A.R.; Visualization, A.O. (Alberto Ortiz) and A.R.; Writing—original draft, A.O. (Alberto Ortiz), A.R., R.G. and A.O. (Andrés Otero); Writing—review & editing, A.O. (Alberto Ortiz), A.R., R.G., A.O. (Andrés Otero) and E.d.l.T.

Funding: This work was partially supported by the Spanish Ministry of Economy and Competitiveness under the project PLATINO, with reference numbers TEC2017-86722-C4-1-R and TEC2017-86722-C4-2-R. The authors would like to thank the Spanish Ministry of Education, Culture and Sport for its support under the FPU grant program.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; and in the decision to publish the results.

References

1. Transon, J.; D'Andrimont, R.; Maignard, A.; Defourny, P. Survey of hyperspectral Earth Observation applications from space in the Sentinel-2 context. *Remote Sens.* **2018**, *10*, 157. [[CrossRef](#)]
2. Fabelo, H.; Ortega, S.; Kabwama, S.; M. Callico, G.; Bulters, D.; Szolna, A.; F. Pineiro, J.; Sarmiento, R. HELICoiD project: A new use of hyperspectral imaging for brain cancer detection in real-time during neurosurgical operations. *Proc. SPIE* **2016**, *9860*. [[CrossRef](#)]
3. Plaza, A.J.; Chang, C.I. *High Performance Computing in Remote Sensing*; CRC Press: Boca Raton, FL, USA, 2007.
4. Wu, Z.; Shi, L.; Li, J.; Wang, Q.; Sun, L.; Wei, Z.; Plaza, J.; Plaza, A. GPU Parallel Implementation of Spatially Adaptive Hyperspectral Image Classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 1131–1143. [[CrossRef](#)]
5. Bernabé, S.; Jiménez, L.L.; García, C.; Plaza, J.; Plaza, A. Multicore Real-Time Implementation of a Full Hyperspectral Unmixing Chain. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 744–748. [[CrossRef](#)]
6. Torti, E.; Danese, G.; Leporati, F.; Plaza, A. A Hybrid CPU–GPU Real-Time Hyperspectral Unmixing Chain. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 945–951. [[CrossRef](#)]
7. González, C.; Bernabé, S.; Mozos, D.; Plaza, A. FPGA Implementation of an Algorithm for Automatically Detecting Targets in Remotely Sensed Hyperspectral Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 4334–4343. [[CrossRef](#)]
8. Báscones, D.; González, C.; Mozos, D. FPGA Implementation of the CCSDS 1.2.3 Standard for Real-Time Hyperspectral Lossless Compression. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 1158–1165. [[CrossRef](#)]
9. Tsigkanos, A.; Kranitis, N.; Theodorou, G.A.; Paschalis, A. A 3.3 Gbps CCSDS 123.0-B-1 Multispectral and Hyperspectral Image Compression Hardware Accelerator on a Space-Grade SRAM FPGA. *IEEE Trans. Emerg. Top. Comput.* **2018**. [[CrossRef](#)]

10. Sweeting, M.N. Modern Small Satellites—Changing the Economics of Space. *Proc. IEEE* **2018**, *106*, 343–361. [[CrossRef](#)]
11. Selva, D.; Krejci, D. A survey and assessment of the capabilities of Cubesats for Earth observation. *Acta Astronaut.* **2012**, *74*, 50–68. [[CrossRef](#)]
12. Yu, Q.Y.; Meng, W.X.; Yang, M.C.; Zheng, L.M.; Zhang, Z.Z. Virtual multi-beamforming for distributed satellite clusters in space information networks. *IEEE Wirel. Commun.* **2016**, *23*, 95–101. [[CrossRef](#)]
13. Poghosyan, A.; Golkar, A. CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Prog. Aerosp. Sci.* **2017**, *88*, 59–83. [[CrossRef](#)]
14. Chin, K.B.; Brandon, E.J.; Bugga, R.V.; Smart, M.C.; Jones, S.C.; Krause, F.C.; West, W.C.; Bolotin, G.G. Energy Storage Technologies for Small Satellite Applications. *Proc. IEEE* **2018**, *106*, 419–428. [[CrossRef](#)]
15. George, A.D.; Wilson, C.M. Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites. *Proc. IEEE* **2018**, *106*, 458–470. [[CrossRef](#)]
16. Pérez, A.; Suriano, L.; Otero, A.; de la Torre, E. Dynamic reconfiguration under RTEMS for fault mitigation and functional adaptation in SRAM-based SoPCs for space systems. In Proceedings of the 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Pasadena, CA, USA, 24–27 July 2017; pp. 40–47. [[CrossRef](#)]
17. Dörflinger, A.; Fiethe, B.; Michalik, H.; Fekete, S.P.; Keldenich, P.; Scheffer, C. Resource-efficient dynamic partial reconfiguration on FPGAs for space instruments. In Proceedings of the 2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Pasadena, CA, USA, 24–27 July 2017; pp. 24–31. [[CrossRef](#)]
18. Rodríguez, A.; Valverde, J.; Portilla, J.; Otero, A.; Riesgo, T.; de la Torre, E. FPGA-Based High-Performance Embedded Systems for Adaptive Edge Computing in Cyber-Physical Systems: The ARTICo³ Framework. *Sensors* **2018**, *18*, 1877. [[CrossRef](#)] [[PubMed](#)]
19. Guerra, R.; Santos, L.; López, S.; Sarmiento, R. A New Fast Algorithm for Linearly Unmixing Hyperspectral Images. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 6752–6765. [[CrossRef](#)]
20. Villa, A.; Chanussot, J.; Benediktsson, J.A.; Jutten, C. Spectral Unmixing for the Classification of Hyperspectral Images at a Finer Spatial Resolution. *IEEE J. Sel. Top. Signal Process.* **2011**, *5*, 521–533. [[CrossRef](#)]
21. Bernabé, S.; Botella, G.; Martín, G.; Prieto-Matias, M.; Plaza, A. Parallel Implementation of a Full Hyperspectral Unmixing Chain Using OpenCL. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 2452–2461. [[CrossRef](#)]
22. Ke, J.; Guo, Y.; Sowmya, A. A Fast Approximate Spectral Unmixing Algorithm Based on Segmentation. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, USA, 21–26 July 2017; pp. 260–266. [[CrossRef](#)]
23. Sánchez, S.; Plaza, A. Fast determination of the number of endmembers for real-time hyperspectral unmixing on GPUs. *J. Real-Time Image Process.* **2014**, *9*, 397–405. [[CrossRef](#)]
24. Sánchez, S.; Ramalho, R.; Sousa, L.; Plaza, A. Real-time implementation of remotely sensed hyperspectral image unmixing on GPUs. *J. Real-Time Image Process.* **2015**, *10*, 469–483. [[CrossRef](#)]
25. Martel, E.; Guerra, R.; López, S.; Sarmiento, R. A GPU-Based Processing Chain for Linearly Unmixing Hyperspectral Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 818–834. [[CrossRef](#)]
26. Guerra, R.; Martel, E.; Khan, J.; López, S.; Athanas, P.; Sarmiento, R. On the Evaluation of Different High-Performance Computing Platforms for Hyperspectral Imaging: An OpenCL-Based Approach. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 4879–4897. [[CrossRef](#)]
27. Shen, H.; Pan, W.D.; Wu, D. Predictive Lossless Compression of Regions of Interest in Hyperspectral Images With No-Data Regions. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 173–182. [[CrossRef](#)]
28. Zhang, L.; Zhang, L.; Tao, D.; Huang, X.; Du, B. Compression of hyperspectral remote sensing images by tensor approach. *Neurocomputing* **2015**, *147*, 358–363. [[CrossRef](#)]
29. Guerra, R.; Barrios, Y.; Díaz, M.; Santos, L.; López, S.; Sarmiento, R. A New Algorithm for the On-Board Compression of Hyperspectral Images. *Remote Sens.* **2018**, *10*, 428. [[CrossRef](#)]
30. Giordano, R.; Guccione, P. ROI-Based On-Board Compression for Hyperspectral Remote Sensing Images on GPU. *Sensors* **2017**, *17*, 1160. [[CrossRef](#)] [[PubMed](#)]
31. Hihara, H.; Moritani, K.; Inoue, M.; Hoshi, Y.; Iwasaki, A.; Takada, J.; Inada, H.; Suzuki, M.; Seki, T.; Ichikawa, S.; et al. Onboard Image Processing System for Hyperspectral Sensor. *Sensors* **2015**, *15*, 24926–24944. [[CrossRef](#)] [[PubMed](#)]

32. Ma, N.; Wang, S.; Ali, S.M.; Cui, X.; Peng, Y. High Efficiency On-Board Hyperspectral Image Classification with Zynq SoC. *MATEC Web Conf.* **2016**, *45*, 05001. [[CrossRef](#)]
33. Villano, M.; Krieger, G.; Moreira, A. Onboard Processing for Data Volume Reduction in High-Resolution Wide-Swath SAR. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 1173–1177. [[CrossRef](#)]
34. Yang, C.; Li, B.; Chen, L.; Wei, C.; Xie, Y.; Chen, H.; Yu, W. A Spaceborne Synthetic Aperture Radar Partial Fixed-Point Imaging System Using a Field-Programmable Gate Array Application-Specific Integrated Circuit Hybrid Heterogeneous Parallel Acceleration Technique. *Sensors* **2017**, *17*, 1493. [[CrossRef](#)] [[PubMed](#)]
35. Jallad, A.M.; Mohammed, L.B. Hardware Support Vector Machine (SVM) for satellite on-board applications. In Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, UK, 14–18 July 2014; pp. 256–261. [[CrossRef](#)]
36. Bao, J.; Zhao, B.; Yu, W.; Feng, Z.; Wu, C.; Gong, Z. OpenSAN: A Software-defined Satellite Network Architecture. *SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 347–348. [[CrossRef](#)]
37. Tang, C.; Hu, X.; Zhou, S.; Liu, L.; Pan, J.; Chen, L.; Guo, R.; Zhu, L.; Hu, G.; Li, X.; He, F.; Chang, Z. Initial results of centralized autonomous orbit determination of the new-generation BDS satellites with inter-satellite link measurements. *J. Geod.* **2018**. [[CrossRef](#)]
38. Hofmann, A.; Glein, R.; Frank, L.; Wansch, R.; Heuberger, A. Reconfigurable on-board processing for flexible satellite communication systems using FPGAs. In Proceedings of the 2017 Topical Workshop on Internet of Space (TWIOS), Phoenix, AZ, USA, 15–18 January 2017; pp. 1–4. [[CrossRef](#)]
39. Glumb, R.; Lapsley, M.; Mantica, P.; Maurer, P.; Reinhard, J.; Kirsch, T. High-Performance On-board Signal Processing for Interferometric CubeSats. In *AIAA SPACE and Astronautics Forum and Exposition*; AIAA SPACE Forum, American Institute of Aeronautics and Astronautics: Reston, VA, USA, 2017. [[CrossRef](#)]
40. Manning, J.; Langerman, D.; Ramesh, B.; Gretok, E.; Wilson, C.; George, A.; MacKinnon, J.; Crum, G. Machine-Learning Space Applications on SmallSat Platforms with TensorFlow. In Proceedings of the 32nd Annual AIAA/USU Conference on Small Satellites, Logan, UT, USA, 4–9 August 2018.
41. Harsanyi, J.C.; Chang, C. Hyperspectral image classification and dimensionality reduction: An orthogonal subspace projection approach. *IEEE Trans. Geosci. Remote Sens.* **1994**, *32*, 779–785. [[CrossRef](#)]
42. Winter, M.E. N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data. *Proc. SPIE* **1999**, 3753. [[CrossRef](#)]
43. Nascimento, J.M.P.; Dias, J.M.B. Vertex component analysis: A fast algorithm to unmix hyperspectral data. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 898–910. [[CrossRef](#)]
44. Zhu, F.; Wang, Y.; Fan, B.; Xiang, S.; Meng, G.; Pan, C. Spectral Unmixing via Data-Guided Sparsity. *IEEE Trans. Image Process.* **2014**, *23*, 5412–5427. [[CrossRef](#)] [[PubMed](#)]
45. Shames, P. A Message Transfer Service for Space Applications. In Proceedings of the SpaceOps 2002 Conference, Houston, TX, USA, 10–19 October 2002; p. 15.
46. Hablot, L.; Gluck, O.; Mignot, J.C.; Genaud, S.; Primet, P.V.B. Comparison and tuning of MPI implementations in a grid context. In Proceedings of the 2007 IEEE International Conference on Cluster Computing, Austin, TX, USA, 17–20 September 2007; pp. 458–463. [[CrossRef](#)]



© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).