Porting a PCA-based hyperspectral image dimensionality reduction algorithm for brain cancer detection on a manycore architecture

R. Lazcano, D. Madronal, Ruben Salvador, Karol Desnos, Maxime Pelcat, R. Guerra, H. Fabelo, S. Ortega, S. Lopez, G. M. Callico, et al.

Porting a PCA-based Hyperspectral Image Dimensionality Reduction Algorithm for Brain Cancer Detection on a Manycore Architecture

R. Lazcano[1], D. Madroñal[1], R. Salvador[1], K. Desnos[2], M. Pelcat[2], R. Guerra[3], H. Fabelo[3], S. Ortega[3], S. López[3], G. M. Callicó[3], E. Juárez[1], C. Sanz[1]

[1]Centre of Software Technologies and Multimedia Systems (CITSEM), Technical University of Madrid (UPM), Spain
[2]IETR, INSA Rennes, CNRS UMR 6164, UEB, France
[3]Research Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria (ULPGC), Spain

{raquel.lazcano, daniel.madronal, ruben.salvador, eduardo.juarez, cesar.sanz}@upm.es
{kdesnos, mpelcat}@insa-rennes.fr
{rguerra, hfabelo, sortega, seblopez, gustavo}@iuma.ulpgc.es

***Abstract*— This paper presents a study of the parallelism of a Principal Component Analysis (PCA) algorithm and its adaptation to a manycore MPPA (Massively Parallel Processor Array) architecture, which gathers 256 cores distributed among 16 clusters. This study focuses on porting hyperspectral image processing into manycore platforms by optimizing their processing to fulfill real-time constraints, fixed by the image capture rate of the hyperspectral sensor. Real-time is a challenging objective for hyperspectral image processing, as hyperspectral images consist of extremely large volumes of data and this problem is often solved by reducing image size before starting the processing itself. To tackle the challenge, this paper proposes an analysis of the intrinsic parallelism of the different stages of the PCA algorithm with the objective of exploiting the parallelization possibilities offered by an MPPA manycore architecture. Furthermore, the impact on internal communication when increasing the level of parallelism is also analyzed.**

Experimenting with medical images obtained from two different surgical use cases, an average speedup of 20 is achieved. Internal communications are shown to rapidly become the bottleneck that reduces the achievable speedup offered by the PCA parallelization. As a result of this study, PCA processing time is reduced to less than 6 seconds, a time compatible with the targeted brain surgery application requiring 1 frame-per-minute.

Keywords— Dimensionality Reduction; Hyperspectral Imaging; Massively Parallel Processing; Real-time processing

I. INTRODUCTION

Hyperspectral imaging (HI) collects both spatial and spectral information from across the electromagnetic spectrum, covering a wide range of wavelengths. This new technology aims at identifying elements in an image by distinguishing among their spectral signatures, which represent the reflectance measured by the sensor for each wavelength [1]. Although the original application field for this technology was remote sensing [2] [3], its use has spread over several research fields, such as astronomy, security, forensics and medicine [4]-[7].

Regarding the medical field, the ability to distinguish among materials has become crucial for cancer detection applications. This technology has already been applied in two different scenarios: ex-vivo and in-vivo studies –i.e., with images captured from a resected sample and directly taken from the patient, respectively.

Related literature shows an increasing research interest concerning the performance of in-vivo HI processing during medical procedures to assist surgeons in discerning between tumor tissues and healthy tissues [8] [9]. Furthermore, to help surgeons in determining the margins of the tumor during surgery, a real-time analysis of the hyperspectral image becomes compulsory, considering this *real-time* as the time needed for the hyperspectral sensor to capture a new image. As nowadays hyperspectral sensors usually present a push-broom scanning mechanism, real-time in this context can be set to a maximum of 1 picture per minute. Neurosurgeons have stated that a processing time of one image per minute is sufficient to assist them during an operation [8]. Processing a diagnostic helping image in less than 1 minute is not possible with the existing alternative to HI, which is the Intraoperative Magnetic Resonance Imaging (iMRI) that usually needs more than thirty minutes to acquire one image [10].

Hyperspectral sensors generate large amounts of data, which makes meeting real-time constraints challenging. As a result, it is advisable to reduce the volume of data before beginning with high-level processing. Therefore, a dimensionality reduction stage is often performed as an essential step during image preprocessing. This dimensionality reduction method is usually accomplished through a principal components transformation [11], which selects and retains the most relevant information for classification.

Principal Component Analysis (PCA) is the most widely used technique in remote sensing applications, specifically in those using hyperspectral images. In this kind of images, the adjacent bands are profoundly correlated, thus providing no new information. PCA reduces the volume of information by removing the dependencies among the different bands. To do so, an eigenvector decomposition of the covariance matrix of the original data is computed [11].

Sequential implementations of this algorithm do not have the performance required to achieve the real-time constraint of this use case. Hence, to reach real-time performance, this study analyzes the intrinsic parallelism of the PCA algorithm and exploits the resulting parallelized model to minimize the time needed for an image to be processed. As this processing requires an extensive usage of computational resources, High Performance Computing (HPC) architectures are targeted.

HPC platforms are evaluated based on two criteria: processing time and energy consumption. Although the former has frequently been the metric chosen to assess HPC platforms, the latter is gaining importance as a first-class performance criterion.

Even though current medical applications rarely work under energy requirements, it is not difficult to foresee future clinical applications where portable and real-time HI processing becomes a crucial tool to support medical decisions. In that sense, manycore processors are today some of the most efficient architectures [12] [13] for the task. For instance, the Kalray Massively Parallel Processor Array (MPPA) in its Bostan version (MPPA-256-N) requires only 5W in average operating mode [14].

The main contribution of this paper is the study and implementation of a PCA algorithm and the evaluation of its performance on an HPC MPPA manycore architecture. Additionally, this research also aims at studying the effect of the internal communications within the manycore architecture when the degree of parallelism is increased. This paper extends the results of [15] with a new discussion section to compare the current work with the state-of-the-art. In addition, the results of a new parallel approach and a better exploitation of the platform parallelism are analyzed. At last, a new data set has been described and employed to obtain the results.

The rest of the paper is structured as follows. First, Section II describes the target MPPA platform together with the studied PCA algorithm. Secondly, Section III focuses on the implementations. Afterwards, Section IV shows the experimental results, and Section V provides a comparison with some state-of-the-art implementations. Finally, Section VI draws the main conclusions of this research work.

II. Hardware and Algorithm

A. MPPA-256-N Kalray Platform

The HPC platform selected for this research is the Kalray MPPA-256-N, whose structure is shown in Fig. 1. This platform is a single-chip manycore processor that gathers 256 cores organized in 16 clusters running at up to 600MHz. It also contains two quad-core Input/Output (I/O) subsystems responsible for handling the communications between a host processor and the clusters, which are interconnected by a Network-On-Chip (NoC).

Each cluster gathers 2 MB of memory shared among the 16 cores within the cluster. In addition, there is also a Direct Memory Access (DMA) engine managing the communications between this memory and the NoC, and a Resource Management (RM) core responsible for starting the NodeOS operating system and handling events and interrupts for the whole cluster.

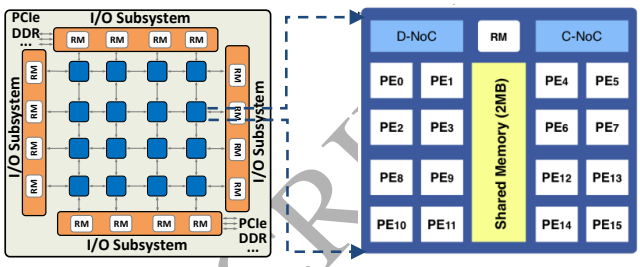

Fig. 1. The Kalray MPPA-256-N: chip (left) and cluster (right) structures

B. Principal Component Analysis

As mentioned in Section I, Principal Component Analysis is the most well-known and widely used technique for data shrinking in HI applications, proving to be a powerful tool for hyperspectral image processing [16].

As hyperspectral images are composed of spectral information gathered from an extensive number of narrow bands, this information is frequently deeply correlated, thus containing a large amount of redundancies. Hence, these redundancies should be eliminated, reducing the image size and, therefore, its processing cost.

Specifically, PCA reduces the data volume by converting the original data into a subspace of smaller dimension where the image is rearranged as a decreasing function of its spectral information – i.e., accumulating the useful spectral information in the first bands–. To do so, PCA computes the covariance matrix of the original data, extracts its associated eigenvectors and projects the image onto these eigenvectors. Finally, the algorithm finishes by selecting the number of bands –or principal components– to retain. Algo. 1 provides the pseudocode of the algorithm, which is divided in four stages:

i. *Image preprocessing*: It is the first step of the algorithm, and it centers the image by computing and removing the average of each spectral band of the original image, composed by N pixels per M spectral bands. It shall be noted that the monochrome image associated to a frequency band is treated as a vector, ignoring the spatial relationship among pixels.

ii. *Covariance computation*: This stage computes the covariance matrix associated to the original image multiplying the preprocessed image by its transpose.

iii. *Eigenvector decomposition*: This step extracts the eigenvectors associated to the covariance matrix computed in the previous stage.

iv. *Projection and reduction*: This stage combines steps 4 and 5, and it projects the original image onto the set of eigenvectors to store then the first P principal components, where P represents the number of principal components –or bands– to retain.

Algorithm 1: Principal Component Analysis

Input: Hyperspectral image Y (N x M matrix)

Step 1: X = Remove the average of each band of Y

Step 2: Covariance matrix C = $X^T \cdot X$

Step 3: E = Eigenvector decomposition of C

Step 4: Projection Q = Y · E

Step 5: Reduce Q to P principal components

Output: Reduced hyperspectral image Q' (N x P matrix)

Algo. 1. PCA algorithm

Related with the eigenvector decomposition, traditionally the conventional method for extracting them has involved computing the inverse matrix and finding the roots of its characteristic polynomial [17]. However, for extensively large matrices –e.g. hyperspectral images– this procedure is not feasible, therefore other methods shall be considered.

In [17], Panju summarizes some of the iterative methodologies for addressing this issue. These approaches work by refining approximations of the eigenvectors in each iteration and, consequently, their convergence depends on the criterion set for the approximation accuracy. However, iterative methods usually work in detriment of real-time, as they are very demanding in terms of processing time. In order to minimize this effect, Jacobi method has been selected in this research due to its high degree of parallelism.

Jacobi method [18][19][20] presents another interesting advantage: besides computing the eigenvalues of the input matrix, it also extracts the eigenvectors associated to them. As the latter are the ones that are of interest for PCA algorithm, a method that does not need extra computation for calculating them is a good match for the studied problem.

This methodology only applies to real and symmetric matrices, and it aims at approximating the original image to a diagonal matrix by applying planar rotations in successive iterations. González et al. provide an extensive description of this method in [21], including the mathematical basis.

This method applies rotations to the largest off-diagonal element with the objective of zeroing it. It should be noted that, at each step, it is possible to undo the zeros reached in previous iterations. However, it has been demonstrated that the overall effect is the magnitude decrement of the nonzero elements, as the sum of the squares of all the off-diagonal elements is proven to be reduced after each iteration. These iterations are repeated until all the off-diagonal elements become smaller than the provided stop condition (ε), which is an input parameter of the algorithm.

The convergence of this algorithm has been demonstrated for two different strategies [18], regarding the order in which the elements are chosen to be zeroed.

1) Classical method: As described before, this method zeroes the largest off-diagonal element in each rotation.

2) Cyclic method: This method zeroes the off-diagonal elements in a given order, e.g. row by row.

The first method has been proven to guarantee the least number of rotations, but the second one is typically faster, as it avoids the location of the largest element in each iteration, which is a quadratic order operation.

Specifically, each Jacobi iteration performs the following steps:

- First, the next off-diagonal element to be zeroed is selected, following one of the two described methods (classical or cyclic Jacobi).
- Secondly, the Jacobi rotation matrix (P) is calculated, which is similar to the model shown in equation (1). The dimensions of this matrix equal those of the covariance matrix C, which is the input for the Jacobi method.
- The α value depends on the element selected to be zeroed –C_{ij}, where i represent the rows of the covariance matrix and j represents the columns–, so it must be recalculated at the beginning of each iteration. In each iteration, equations (2) to (5) are applied to calculate this value and, hence, the Jacobi rotation matrix.

$$P = \begin{pmatrix} 1 & \cdots & \cdots & 0 \\ \vdots & cos\,\alpha & sin\,\alpha & \vdots \\ \vdots & -sin\,\alpha & cos\,\alpha & \vdots \\ 0 & \cdots & \cdots & 1 \end{pmatrix} \quad (1)$$

$$m = \frac{2 \cdot C_{ij}}{C_{jj} - C_{ii}} \quad (2)$$

$$t = \frac{-1 + \sqrt{1 + m^2}}{m} \quad (3)$$

$$\cos\alpha = \frac{1}{\sqrt{1 + t^2}} \quad (4)$$

$$\sin\alpha = t \cdot \cos\alpha \quad (5)$$

- Once the matrix P has been computed, the operation provided in (6) is performed. As a result, the off-diagonal element is now zeroed in C_1, and also its symmetric counterpart, as C has to be symmetric. For the next iteration, C_1 will be considered as the input of Jacobi algorithm.

$$C_1 = P_1^T \cdot C \cdot P_1 \quad (6)$$

Each iteration repeats the described steps, zeroing one element at a time. However, the algorithm does not stop when all the elements have been chosen to be zeroed once, due to the fact that, in each iteration, several previous zeros can be undone. That is the reason why a stop factor is needed.

Once the last iteration finishes –*K*– and the convergence is reached, the eigenvalues are stored in the diagonal of C_K, as shown in (7), where P_i are the successive Jacobi rotation matrices.

$$C_K = P_i^T \cdot \cdots \cdot P_1^T \cdot C \cdot P_1 \cdot \cdots \cdot P_i \qquad 1 < i < K \qquad (7)$$

Likewise, the eigenvectors associated to these eigenvalues can be calculated as depicted in (8), where the eigenvectors are placed in the columns of E.

$$E = P_1 \cdot P_2 \cdot P_3 \cdot \cdots \cdot P_K \qquad (8)$$

As each rotation affects only a couple of rows and columns, several rotations can be calculated simultaneously, thus providing a parallel method.

Specifically, in each iteration the elements that are changed in the input matrix are the ones belonging to the concerned rows and columns, so several elements can be processed in parallel if they do not share any of those positions. For instance, elements C_{23} and C_{45} could be processed in parallel; on the contrary, C_{23} and C_{34} could not be simultaneously zeroed.

Next section will provide a detailed description of the adaptation of this method –along with the adaptation of the rest of the operations involved in PCA algorithm– to the architecture under study.

III. Implementation

A. Database

The conducted experiments have been tested upon hyperspectral images extracted from the HELICoiD project database [22]. The in-vivo human brain surface images were captured during neurosurgical operations performed at the University Hospital Doctor Negrin of Las Palmas de Gran Canaria (Spain) and at the University Hospital of Southampton (UK).

To capture these images, the HELICoiD setup described in [23] has been used. Two hyperspectral sensors compose this setup: one in the visible and near infrared spectral range (VNIR), covering from 400 nm to 1000 nm, and the other in the near infrared range (NIR), covering from 900 nm to 1700 nm of the electromagnetic spectrum. Both cameras are attached to a push-broom scanning unit, together with an illumination system that provides a cold light to protect the exposed brain surface from the heat generated by the lamp. As the cameras take the images with a push-broom mechanism, each image needs from 1 to 2 minutes to be captured –depending on the spatial size of the image–, which can be considered as the definition of the surgery real-time constraint.

Specifically, two different hyperspectral images have been used to assess the algorithm. A preprocessing stage has spatially limited the image to the area of interest and it has reduced the spectral resolution to 128 bands [24]. As a result, the first image –hereafter case 1– presents a spatial resolution of 377 lines and 329 samples –i.e., 124033 pixels–, while the second –hereafter case 2– contains 479 lines and 552 samples –i.e., 264408 pixels–. Fig. 2 gathers both hyperspectral images, case 1 on the left and case 2 on the right.

Considering that these images are stored as float numbers (4 Bytes), the required memory for storing each image is 60.6 MB for case 1 –i.e. 124033 pixels x 128 bands– and 129.1 MB for case 2 –i.e. 264408 pixels x 128 bands–, respectively.

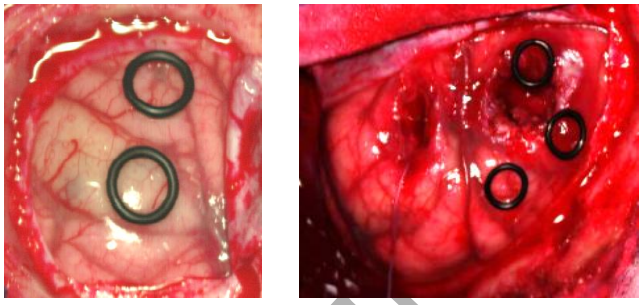

Fig. 2. RGB representation of the hyperspectral images extracted from the HELICoiD database: case 1 (left) and 2 (right)

B. Target environment

As mentioned in Section I, the aim of this work is to achieve real-time while processing hyperspectral images during a surgical procedure. Due to the large amount of information contained in these images, the computational complexity is such that sequential implementations of the algorithms do not provide enough performance to fulfill these requirements; thus, HPC platforms become a necessity.

The Kalray MPPA-256-N appears to be an optimal solution, as it is particularly competitive in terms of energy efficiency, which is a parameter of growing interest. Specifically, the target environment is a workstation that includes an MPPA-256-N chip, whose simplified block diagram when connected to a host PC is provided in Fig. 3. The MPPA architecture presents three different levels: the host module, the I/O interface and the 256 processing units, which are organized in 16 clusters. The host module is responsible for managing the global functioning, and it communicates with the I/O interface through a PCI express (PCIe) connection; it also presents the largest memory space, with more than 10 GB of available memory. Similarly, the I/O interface handles both the communications with the host –through the PCIe connection– and with the clusters –through a NoC interface–; as for the available memory, it contains a 4 GB external DDR. Finally, the processing cores are responsible for the processing itself. As this chip contains 256 processing cores, the potential level of parallelism is very high. Nevertheless, this platform also presents an important restriction for hyperspectral image processing, which is the reduced amount of memory within each cluster –a 2 MB block of shared memory among the 16 internal cores of each cluster–. Furthermore, some of this memory is reserved for both program code and operating system, so the available memory for storing data cannot represent more than 1.5 MB, approximately.

As hyperspectral images gather extremely large volumes of data, this is an important limitation. As described before, the images used during this work require 60.6 MB –case 1– and 129.1 MB –case 2–. Therefore, it seems obvious that, to process an entire image, the algorithm must be split into several iterations. Subsequently, iterating the processing of the image highlights another important limitation, which is the

communication between the I/O subsystems and the clusters. This communication is very demanding in terms of performance, so it seems obvious that the processing time will be proportional to the number of iterations needed to complete the algorithm execution.

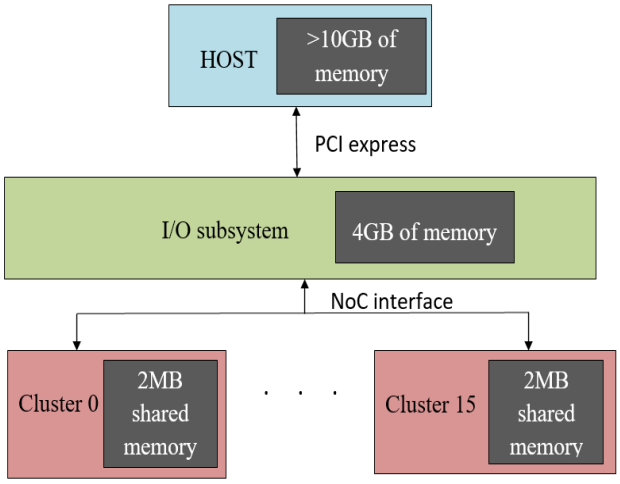

Fig. 3. MPPA-256-N simplified block diagram to show the memory hierarchy

C. MPPA-256-N implementation

Taking into consideration the previous features, this section provides a detailed description of the implementation of each part of the PCA algorithm, highlighting its intrinsic parallelism. It shall be noted that this implementation is an extended work of that presented in [15].

1) Preprocessing: To compute the covariance matrix of the original data, first the image ought to be mean-centered. To do so, the average of each spectral band is calculated and removed. This operation is band-wise parallelizable: the original matrix is divided among the clusters, and the internal cores compute and remove the average of the bands received in each cluster. Due to the high dimensionality of the image and the cluster memory limitation, each cluster can only compute the average of one band at a time; therefore, each iteration processes 16 bands simultaneously. As a result, each core computes one portion of the average of the band; then, the master thread –i.e., the core executing the *main* function– computes the global average and subtracts it.

2) Covariance matrix computation: The resulting matrix is then multiplied by its transpose, thus generating the covariance matrix. As neither of these matrices fit into a cluster, this computation needs to be iterated; further, due to the dimensions of the matrices involved in the multiplication, this operation is the main bottleneck of the algorithm. Considering that the memory needed to store a row –i.e., a band– of the matrix is 0.48 MB in case 1 and 1 MB in case 2, there are two different cases:

- For the smallest image –case 1–, two bands fit into a cluster, so each cluster can compute one element of the covariance matrix at a time. This means that, in each iteration, 16 elements of the covariance matrix can be computed simultaneously. As the dimensions of the resulting matrix are 128×128, 1024 iterations are required to complete this computation.
- On the other hand, for the largest image, even two bands exceed the memory restriction, so at least two clusters must be used to compute just one element of the covariance matrix. To do so, instead of sending a whole row to each cluster, all the bands are divided in half, and each cluster only receives the halves that should be multiplied (as a matrix multiplication is just a concatenation of dot products). With this method, the number of iterations is thus doubled.

In both cases, the cores within each cluster compute their corresponding share of the vector multiplication –as they are a concatenation of dot products– and send it to the I/O subsystem, which adds them and stores the resulting value.

Nevertheless, it should be noted that, as multiplying a matrix by its transpose generates a symmetric matrix, only the upper triangle of the covariance matrix needs to be computed, thus reducing the iterations in half for both cases.

Furthermore, it is also worth noting that, in each iteration, two different transmissions are required, one for each of the bands –or half-bands– intervening in the multiplication. Considering that, as mentioned before, the communications are one of the setbacks of the architecture under study, the impact of this process on the overall execution time is presumably very high, as the time consumed in the data transfers would be much higher than the one consumed in the processing itself.

For that reason, another method has also been proposed for avoiding these data transfers. This method consists of taking advantage of the host of the platform to remove the memory restriction and, thus, all the I/O-cluster communications. Although the processing time will certainly increase –as the multiplication will not be parallelized–, the time saved in the communications will hypothetically compensate this increase. These two solutions will be evaluated in Section IV.

3) Eigenvector decomposition: This stage deals with the implementation of the Jacobi method introduced in Section II. Among all the existing variations, the cyclic Jacobi [21] has finally been implemented. As described in Section II, the main variation of this method is that, instead of choosing the elements to be zeroed by finding the largest off-diagonal element, it chooses the next element in a given order, e. g., row by row. As the covariance matrix (128×128) fits into a cluster, the use of only one of them for implementing this step removes all the internal communications, as all the cores wihtin a cluster share the same memory. In addition, as Jacobi method is highly parallelizable, all cores of the chosen cluster have been used for implementing this step. As a result, this method has been implemented as follows:

- The master thread –i.e., the core executing the *main* function– is responsible for handling the search of the next element to be zeroed. In each iteration, this thread verifies that the stop condition is not fulfilled, chooses a maximum of 15 different new elements to be processed in parallel, calculates the rotation matrix and sends it to the processing cores. This process is repeated until the stop condition is reached by all the off-diagonal elements. Once this happens, this thread sorts the eigenvalues in a descending order, as well as their associated eigenvectors.
- Likewise, the processing cores perform the operation shown in (9), where *i* represents the current iteration, P_i is the rotation matrix of the iteration *i*, C_{i-1} is the covariance matrix modified in the previous iteration and C_i is the resulting matrix –i.e., the covariance matrix with several elements already zeroed–. It shall be noted that (9) is a generalization of equation (6) described in Section II.

$$C_i = P_i^T \cdot C_{i-1} \cdot P_i \quad (9)$$

4) Projection and principal components selection: In this step, the original matrix is projected onto the set of eigenvectors and the first P bands –i.e., principal components– are stored. To reduce the complexity of the projection, instead of using the whole set of eigenvectors only the subset of the first P ones is utilized. Specifically, as for this application only the first principal component is required [25], the set of eigenvectors is thus reduced from 128 to 1. This also reduces the projection complexity, since instead of multiplying two matrices, just one matrix –the original data– by a vector –the first eigenvector– multiplication is required. Related with the parallelization, the method is similar to the one applied in step 2, but much simpler. As the eigenvector fits into a cluster, it is broadcasted to all of them and then the original matrix is split and sent to the clusters in a pixel-wise order iteratively –i.e., in groups of 128 elements–, until all the matrix is multiplied by the eigenvector. From the point of view of the parallelization within the clusters, each one receives blocks of 1024 pixels, so each core projects 64 pixels onto the eigenvector and returns the results to the I/O subsystem.

Consequently, concerning the MPPA-256-N resource usage, the PCA algorithm is divided into two blocks: steps 1, 2 and 4 use all the available resources (16 clusters and 16 cores per cluster), while step 3 only uses the 16 cores of one cluster. It should be noted that, for the alternative solution to step 2, all platform resources are idle, as the processing is performed in the host.

IV. RESULTS

In this section, the results obtained while exploring the proposed solutions are evaluated in terms of processing time. First, the sequential version of the PCA algorithm on the MPPA-256-N is assessed; then, the parallelism is exploited, locating the main bottlenecks. After that, the communication cost is evaluated in order to analyze its effects on the overall computation time. Finally, the second proposed approach for computing the covariance matrix is evaluated. It should be noted that the results presented in this section have been measured when setting the stop factor, ε, to the maximum value that provides a final result with a relative error lower than 1% when compared to a Matlab version.

A. First approach

In order to address the analysis of the obtained results, the different stages of the PCA algorithm described before are characterized. To do so, as the MPPA architecture presents two different levels of parallelism, first the sequential implementation will be studied; after that, the first level of parallelism will be evaluated by distributing the computational load among the 16 clusters, but using just 1 core in each one. Finally, the second level of parallelism is assessed by using all the cores of each cluster, instead of just one.

Table 1 provides the execution time for each of the stages when they are processed in a sequential way –i.e., using only one core of one cluster–. As expected, the main bottleneck of the algorithm is the covariance computation stage, as it consumes more than a 92% of the global execution time.

TABLE I. AVERAGE TOTAL EXECUTION TIME (MS) - SEQUENTIAL

Experiment	Step 1	Step 2	Step 3	Steps 4-5	Global
Case 1	900.7	40,870.5	1,759.8	840.2	44,305.2
Case 2	1,908.9	81,503.4	1,849.7	1,759.8	87,321.8

Likewise, Table II presents the execution time of each stage of the algorithm, when their computational load is divided among the 16 available clusters –i.e., using the 16 clusters, but just 1 core of each of them–. As a result, it can be noticed that, although the execution time has decreased, the global speedup achieved is rather low, as ideally it should grow up to 16 but in reality it is, approximately, 1.5. This limitation is basically due to data broadcasting: as hyperspectral images contain extremely high volumes of information, the transmission of the images to the clusters could be such that it could overcome any speedup the processing itself may achieve. To check whether this hypothesis is correct, first the second level of parallelization is studied and, after that, the effect of communications is evaluated.

TABLE II. AVERAGE TOTAL EXECUTION TIME (MS) – FIRST LEVEL OF PARALLELISM

Experiment	Step 1	Step 2	Step 3	Steps 4-5	Global
Case 1	402.7	26,699.4	1,759.8	380.3	29,244.2
Case 2	860.4	52,990.1	1,849.7	839.6	56,540.2

Table III gathers the equivalent results to those presented in Tables I and II, but exploiting the whole architecture, i.e., using the 16 cores of the 16 clusters. As can be observed, the speedup achieved is almost negligible when compared to the previous one, since it only accelerates the computation by a factor of 1.1, approximately. Again, the reason to explain this behavior is the communications: supposing that the previous

hypothesis was correct, it explains this new behavior, as it means that the processing itself cannot be further parallelized, and hence all the measured time is being wasted for communications. To prove whether this hypothesis is true, next subsection evaluates the impact of communications.

TABLE III. AVERAGE TOTAL EXECUTION TIME (MS) – SECOND LEVEL OF PARALLELISM

Experiment	Step 1	Step 2	Step 3	Steps 4-5	Global
Case 1	393.3	25,726.1	350.9	369.9	26,840.2
Case 2	831.9	51,039.5	370.3	779.8	53,021.5

Before getting into the communication assessment, the behavior of the Jacobi algorithm should be highlighted. As described in Section III, Jacobi has only been implemented using one cluster due to the communications cost, so there is no difference in the time measured for Jacobi in Tables I and II. As for the results presented in Table III, it can be observed that, in this case, the Jacobi algorithm is accelerated by a factor of 5, approximately. This result supports the communications hypothesis: as described before, the input of the Jacobi algorithm –i.e., the covariance matrix– fits into a cluster; as a result, this step is the only one in which there are not any communications that could cause a delay in the processing of the algorithm. Additionally, there are two main reasons as to why the speedup for Jacobi step is still not near the ideal –16–. Firstly, the Jacobi algorithm is an iterative process that aims at zeroing one off-diagonal element at a time, and the order in which these elements are chosen affects the number of iterations, as each iteration can undo the zeroes achieved in the previous one and, hence, the number of iterations needed to reach the convergence criterion may vary. Secondly, the Jacobi algorithm has data dependencies, so its convergence may vary depending on the nature of the data –e.g., their dynamic range.

B. Communications assessment

This subsection deals with the evaluation of communications, so as to check whether they become a bottleneck or not. To do so, the measurements previously shown have been divided in two: transmission (hereafter, *Tx*) and processing (hereafter, *Px*). These measurements are gathered in Tables IV and V –case 1 and case 2, respectively–. As can be observed, each table contains the three configurations described before: the sequential execution (1 cluster – 1 core), the first level of parallelism (16 clusters – 1 core) and the second one (16 clusters – 16 cores).

TABLE IV. DETAILED SYSTEM EXECUTION TIME (MS) – CASE 1

Cluster-Core		1-1	16-1	16-16	Speedup
Step 1	Tx	329.8	329.8	329.8	1
	Px	541.4	50.9	5.1	106.2
Step 2	Tx	25,609.9	25,609.9	25,609.9	1
	Px	15,257.3	1,089.4	89.9	169.7
Step 3	Tx	0.34	0.34	0.34	1
	Px	1,759.8	1,759.8	350.9	5.0
Steps 4-5	Tx	329.8	329.8	329.8	1
	Px	470.9	30.9	4.3	109.5
Global	Tx	26,269.8	26,269.8	26,269.8	1
	Px	18,029.4	2,931.1	470.5	38.3

The most important conclusion that can be extracted from these tables is that, indeed, the time needed for the data to be transmitted to the clusters is extremely high. The most representative example of this behavior is the covariance matrix computation (step 2): as shown in both tables, even in the sequential execution the transmission time already consumes more than 60% of the covariance execution time. As a result, when the processing is fully parallelized, more than 99% of the covariance execution time is dedicated to data transmission.

The main reason why the communications delay is so large is that the I/O interface is executed sequentially, that is, the communications are not parallelized. This is shown in both tables, where there are no speedups for any communications. For instance, in step 1, the whole image is needed. As a result, when only one cluster is used, the entire image is sent to it. Conversely, when several clusters are involved, the image is distributed among them –i.e., the same amount of data is transmitted–, but the transmission time remains the same because the I/O core sends the information sequentially. Furthermore, each transmission is blocking, so until the transmission to the first cluster has finished, the second transmission cannot begin.

TABLE V. DETAILED SYSTEM EXECUTION TIME (MS) – CASE 2

Cluster-Core		1-1	16-1	16-16	Speedup
Step 1	Tx	721.9	721.9	721.9	1
	Px	1,177.4	86.1	11.2	105.2
Step 2	Tx	50,796.8	50,796.8	50,796.8	1
	Px	30,706.2	2,193.3	178.2	172.3
Step 3	Tx	0.34	0.34	0.34	1
	Px	1,849.7	1,849.7	370.3	5.0
Steps 4-5	Tx	721.9	721.9	721.9	1
	Px	999.6	68.0	9.2	108.6
Global	Tx	52,240.9	52,240.9	52,240.9	1
	Px	34,732.9	4,247.1	642.3	54.1

Finally, related with the speedups achieved on the processing itself, it can be observed that, in general, they are rather large –more than 100, without taking into consideration the Jacobi step, as it is not distributed among the 16 clusters–, which proves that the parallelization potential of this architecture is very high.

As a result, the initial hypothesis has been proven correct: although the potential parallelism offered by an architecture

such as the MPPA-256-N is rather large, the communications within the chip quickly become the main bottleneck of data consuming algorithms, as PCA.

C. Second approach

The previous subsection has proven right the initial hypothesis about the communications. Now, as a proof of concept, the second approach proposed in Section III is analyzed.

This approach aims at proving that the main bottleneck of the system is the communications by removing them from the most consuming stage of the algorithm, which is the covariance matrix computation. To do so, instead of performing computation on the clusters, the host is used for the task, where the memory limitation is lower and, thus, there is no need for communicating data.

Specifically, Table VI gathers the results obtained when applying this solution, comparing them with those obtained in the previous one. As only the covariance matrix computation is involved, just the results of step 2 are provided. Additionally, the global execution times for both cases and in all the configurations are also presented, as well as the speedup obtained in each configuration when the mentioned computation is moved to the host.

As can be noticed, performing this operation on the host clearly provides considerably better results in terms of processing time, as, for instance, a speedup of more than 35 is achieved when comparing a sequential execution –1 cluster, 1 core– with the equivalent version in the host. Specifically, the speedup achieved with this change is such that even using all the resources of the chip provides far worse results than those generated by the sequential version of the system –but with the covariance computation in the host–, as the speedup achieved with the latter is still more than 22.

TABLE VI. COVARIANCE MATRIX COMPUTATION COMPARISON (MS)

Configuration		Case 1		Case 2	
		Step 2	Global	Step 2	Global
1-1	Clusters	40,870.5	44,305.2	81,503.4	87,321.8
	Host	1,061.1	4,518.9	2,254.7	7,737.2
	Speedup	**38.5**	**9.8**	**36.1**	**11.3**
16-1	Clusters	26,699.4	29,244.2	52,990.1	56,540.2
	Host	1,061.1	3,602.2	2,254.7	5,794.4
	Speedup	**25.2**	**8.1**	**23.5**	**9.8**
16-16	Clusters	25,726.1	26,840.2	51,039.5	53,021.5
	Host	1,061.1	2,196.6	2,254.7	4,644.7
	Speedup	**24.2**	**12.2**	**22.6**	**11.4**

To conclude this section, Fig. 4 provides a graphical comparison, for case 1, of the processing times for the four different implementations described before: (i) the sequential implementation –1 cluster, 1 core–; (ii) the first parallelization approach –16 clusters, 1 core–; (iii) the second parallelization –16 clusters, 16 cores–; and iv) the one that moves the covariance computation to the host, while the rest of the algorithm maintains the configuration of the second approach.

Fig. 4 also provides the speedups of each implementation when compared to the sequential one; as a result, it can be observed that the speedup achieved when moving the covariance computation to the host is, approximately, 20; on the other hand, the speedup achieved when using only the MPPA resources is 12.5.

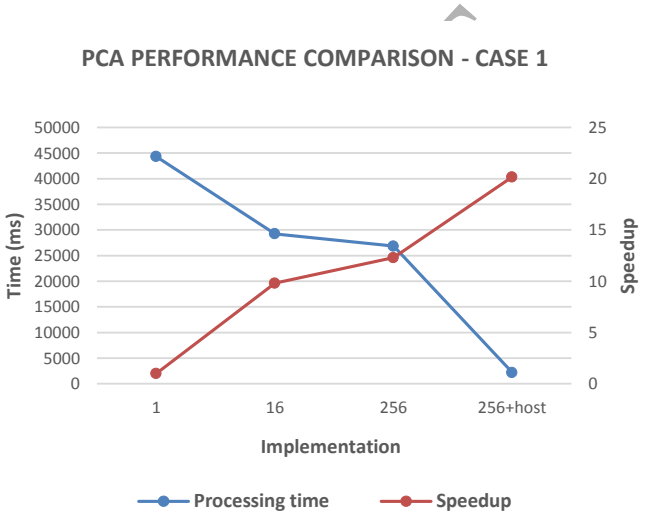

Fig. 4. PCA performance comparison for case 1: execution time and speedup for different implementations

Finally, to graphically observe the generated results, Fig. 5 displays the one-band representation of the image obtained when applying the PCA algorithm to both cases 1 and 2 described before. As can be seen, these images contain a spatial representation of the most relevant spectral information, which drastically reduces data dimensionality.

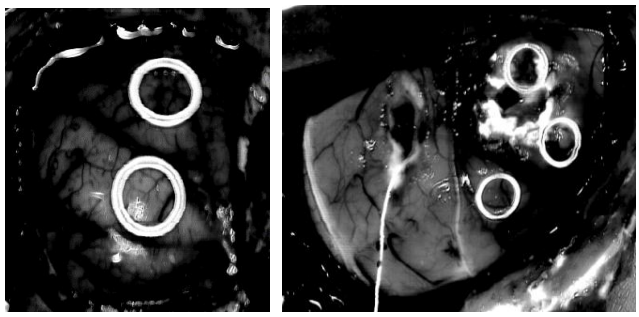

Fig. 5. PCA result (one band representation of the original image) for case 1 (left) and 2 (right)

V. DISCUSSION

After analyzing the results obtained during this research work, the present section compares the performance achieved using the MPPA architecture with studies presented in [26] and [27]. The former deals with the implementation of PCA on a Xilinx Virtex-7 XC7VX690T FPGA, while the latter uses the dataflow programming language RVC-CAL [28] to highlight

the intrinsic parallelism of the algorithm and automatically distribute the workload among the available cores of an Intel Core i7-4790 –which gathers four processors running at 3.6 GHz– with 32 GB of RAM.

Before starting the comparison itself, it should be noted that, as HI technology initially aimed remote sensing, when working with hyperspectral images the most common datasets used in the related literature are the well-known AVIRIS Cuprite scene –available online[1] and hereafter AV_Cuprite–, and the AVIRIS Jasper Ridge scene –hereafter AV_JRidge–. Both images have been widely used for validating the accuracy of dimensionality reduction algorithms, such as PCA. They are composed of several reference ground signatures of well-known minerals, which can be also extracted from the United States Geological Survey (USGS[2]) database. These two images have been the ones used for evaluation purposes in [26].

However, the pervasive development of HI has led to its application to several other fields, such as the medical one. In this line, this work and the one in [27] use the same input dataset as introduced in Section III, which has been obtained from the HELICoiD project database [22].

As a result, these implementations cannot be compared directly, so the comparison will be performed in terms of the size of the images. In [26], the images have an approximate size of 50 MB (AV_Cuprite, 350×350×224) and 140MB (Av_JRidge, 614×512×224) –lines×samples×bands–. Likewise, the images used in this work need 60.6 MB (case 1, 377×329×128) and 129.1 MB (case 2, 479×552×128) –again lines×samples×bands– to be stored.

From the dimensions of the images of both datasets can be inferred that, in the first one, the image is stored with a precision of 2 bytes, while in the second one the precision required is 4 bytes. In other words, the precision in the second dataset is twice the one applied in the first one, which is an important feature when comparing the implementations.

For each implementation, Table VII provides the global processing load of each image, as well as its size. Furthermore, it also provides a generic metric to simplify the comparison among them, which is the time needed for each implementation to process 1 MB of information.

As can be observed, the processing time grows linearly with the size of the image, as the time needed for processing 1 MB of information remains constant regardless of the implementation. Additionally, as shown in Table VII, the FPGA implementation achieves a processing cost of almost 30 ms per MB of information, while the MPPA implementation reaches a processing rate of 36 ms per MB. On the other hand, the Intel-i7 implementation (RVC-CAL) obtains a processing rate of 10 ms per MB.

This comparison shows, again, that the memory limitation plays a crucial role in the processing of data consuming algorithms, as the x86 architecture, which is the one with the least memory limitations thanks to its complex cache architecture, achieves the largest processing rate. The reason of this result is also that its cores run at 3.6 GHz, while those of the MPPA run at 600 MHz.

However, the x86 architecture cannot compete with both FPGAs and manycore platforms in terms of power consumption. For this metric, the MPPA-256-N used during this research clearly outperforms both the Xilinx Virtex-7 XC7VX690T FPGA and the Intel Core i7-4790, as they consume 5W [14], 30-40W [29] and 84W[3], respectively in average conditions. As power consumption is an important feature for clinical applications, trying to minimize this value is very important for designing a portable prototype.

Consequently, the implementation developed during this research work can be considered competitive when compared to those described before, as it outperforms its competitors in terms of power consumption. Furthermore, when compared to the FPGA implementation, it can be observed that the results are equivalent, considering that the precision used in this implementation is twice the precision of the FPGA one.

TABLE VII. PCA TIME COMPARISON (MS)

Implementation	Database	Global processing time	Size (MB)	Processing time per MB
FPGA	AV_Cuprite	1,490.0	50.0	29.8
FPGA	AV_JRidge	4,170.0	140.0	29.8
INTEL-i7	HELICoiD_C1	614.8	60.6	10.1
INTEL-i7	HELICoiD_C2	1,265.1	129.1	9.8
MPPA	HELICoiD_C1	2,196.6	60.6	36.2
MPPA	HELICoiD_C2	4,644.7	129.1	36.0

VI. CONCLUSION

This paper has presented an analysis of the parallelism of a PCA algorithm, including its adaptation to the manycore architecture MPPA-256-N from Kalray, and its comparison with other state-of-the-art studies. The proposed implementation aims at adapting the algorithm to process hyperspectral image analysis in real-time, so as to help surgeons in locating brain tumors during surgical procedures.

On the one hand, the obtained results show that, as expected in such a data consuming algorithm, communications among processing units quickly become the main bottleneck of the system. Consequently and, as a proof of concept, two different implementations have been carried out: first, the most consuming stage of the algorithm has been located and has been parallelized over the clusters and then it has been processed sequentially in the host of the platform. The analysis of these implementations has proven that, indeed, communications are the main bottleneck, as they can consume up to a 99% of the processing time. Therefore, a trade-off between the level of parallelism and the increase in internal communications must be met.

[1] http://aviris.jpl.nasa.gov
[2] http://speclab.cr.usgs.gov/spectral-lib.html
[3] http://ark.intel.com/products/80806/Intel-Core-i7-4790-Processor-8M-Cache-up-to-4_00-GHz

On the other hand, the exploitation of the available resources on the MPPA-256-N demonstrates that, in favorable cases, the intrinsic parallelism of the algorithm can be fully exploited. Speedups up to 170 have been achieved on the processing of certain stages of the algorithm on a 256-core implementation. Although these speedups decrease to 20 due to communication delays, the real-time objective of one hyperspectral image per minute is still reached for the application at hand. In these conditions, the PCA algorithm consumes less than a 10% of the available processing time, keeping processing resources for other hyperspectral image analysis tasks.

This implementation has been compared to state-of-the-art implementations of the same algorithm on an FPGA and an x86 architecture, the proposed solution proving to be competitive and providing better power efficiency rates.

As a final conclusion, experimental results have demonstrated that manycores are promising architectures for medical hyperspectral image processing. Future studies will investigate the possibilities to increase the effective communication bandwidth, so as to achieve speedups closer to 256x, which is the ideal speedup achievable with this architecture. Furthermore, OpenMP and OpenCL implementations of the algorithm will be compared to the current POSIX results.

ACKNOWLEDGEMENTS

This research has been funded by the EU FET HELICoiD (HypErspectraL Imaging Cancer Detection) project (FP7-ICT-2013.9.2 (FET Open) 618080).

REFERENCES

[1] D. Manolakis and G. Shaw, “Detection algorithms for hyperspectral imaging applications,” IEEE Signal Processing Magazine, vol. 19, no. 1, pp. 29–43, 2002.

[2] M. Govender, K. Chetty and H. Bulcock, “A review of hyperspectral remote sensing and its application in vegetation and water resource studies,” Water Sa, vol. 33, no. 2, 2007.

[3] C. Gomez, R.A.V. Rossel and A.B. McBratney, “Soil organic carbon prediction by hyperspectral remote sensing and field vis-NIR spectroscopy: An Australian case study,” Geoderme, vol. 146, no. 3, pp. 403-411, August 2008.

[4] E.K. Hege, D. O’Connell, W. Johnson, S. Basty and E.L. Dereniak, “Hyperspectral imaging for astronomy and space surviellance,” in Optical Science and Technology, SPIE’s 48th Annual Meeting, International Sociesty for Optics and Photonics, pp. 380-391, January 2004.

[5] D. Manolakis, D. Marden and G.A. Shaw, “Hyperspectral image processing for automatic target detection applications,” Lincoln Laboratory Journal, vol. 14, no. 1, pp. 79-116, January 2003.

[6] G.J. Edelman, E. Gaston, T.G. Van Leeuwen, P.J. Cullen and M.C.G. Aalders, “Hyperspectral imaging for non-contact analysis of forensics traces,” Forensic science international, vol. 223, no. 1, pp. 28-39, November 2012.

[7] M.E. Martin, et al, “Development of an advanced hyperspectral imaging (HIS) system with applications for cancer detection,” Annals of biomedical engineering, vol. 34, no. 6, pp. 1061-1068.

[8] H. Fabelo et al, “HELICoiD project: a new use of hyperspectral imaging for brain cancer detection in real-time during neurosurgical operations,” Hyperspectral Imaging Sensors: Innovative Applications and Sensor Standards 2016, Proc. SPIE 986002 (2016).

[9] G. Lu and B. Fei, “Medical hyperspectral imaging: a review,” Journal of Biomedical Optics, vol. 19, no. 1, 2014.

[10] J. Ramm-Pettersen et al, “Intra-operative MRI facilitates tumour resection during trans-sphenoidal surgery for pituitary adenomas,” Acta neurochirurgica, vol. 153, no. 7, pp. 1367–1373, 2011.

[11] C. Rodarmel and J. Shan, “Pincipal component analysis for hyperspectral image classification,” Surveying and Land Information Science, vol. 62, no. 2, p. 115, 2002.

[12] M. Castro, F. Dupros, E. Francesquini, J.F. Méhautk, P.O.A. Navaux, “Energy efficient seismic wave propagation simulation on a low-power manycore processor,” IEEE Xplore Digital Library, 26th International Symposium on Computer Architecture and High Performance Computing, p. 8, October 2014.

[13] E. Francesquini, et al, “On the energy efficiency and performance of irregular application executions on multicore, NUMA and manycore platforms,” Journal of Parallel and Distributed Computing, vol. 76, pp. 32-48, February 2015.

[14] B.D. de Dinechi et al., “A clustered manycore processor architecture for embedded and accelerated applications,” High Performance Extreme Computing Conference (HPEC), pp. 1-6, 2010.

[15] R. Lazcano et al, “Parallelism Exploitation of a Dimensionality Reduction Algorithm Applied to Hyperspectral Images” Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on, October 2016.

[16] J. M. Bioucas-Dias et al., “Hyperspectral remote sensing data analysis and future challenges,” IEEE Geoscience and Remote Sensing Magazine, vol. 1, no. 2, pp. 6-36, 2013.

[17] M. Panju, “Iterative methods for computing eigenvalues and eigenvectors,” Waterloo Mathematics Review, pp. 9-18, 2011.

[18] G. H. Golub and C. F. van Loan, “Matrix computations”, 3rd edition, John Hopkins University Press, Baltimore, 1996.

[19] A. Quarteroni, R. Sacco, and F. Saleri, “Numerical mathematics,” 2nd edition, New York, NY, USA: Springer, pp. 183-238, 2007.

[20] G. E. Forsythe, and P. Henrici, “The cyclic Jacobi method for computing the principal values of a complex matrix,”, Transactions of the American Mathematical Society, vol. 94, no. 1, pp. 1-23, 1960.

[21] C. González, S. López, D. Mozos, and R. Sarmiento, “FPGA implementation on the HySime algorithm for the determination of the number of endmembers in hyperspectral data,” IEEE Journal on Selected Topics in Applied Earth Observations and Remote Sensing, vol. 8, no. 6, pp. 2870-2883, 2015.

[22] S. Kabwama et al, “Intra-operative Hyperspectral Imaging for Brain Tumour Detection and Delineation,” XXXI Design of Circuits and Integrated Systems Conference (DCIS), 2016.

[23] R. Salvador et al, “Demonstrator of the HELICoiD tool to detect in real time brain cancer,” Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on, pp. 1-2, IEEE, October 2016.

[24] H. Fabelo et al, “A Novel Use of Hyperspectral Images for Human Brain Cancer Detection using in-Vivo Samples,” Proceedings of the 9th International Joint Conference on Biomedical Engineering Systems and Technologies, pp. 311-320, 2016.

[25] K. Huang, S. Li, X. Kang and L. Fang, “Spectral-Spatial Hyperspectral Image Classification Based on KNN,” Sensing and Imaging, vol. 17, no. 1, pp. 1-13, 2016.

[26] D. Fernandez, C. Gonzalez, D. Mozos and S. Lopez, “FPGA implementation of the Principal Component Analysis algorithm for dimensionality reduction of hyperspectral images”, Journal of Real-Time Image Processing, pp. 1-12, 2016.

[27] R. Lazcano et al., “Parallelism exploitation of a PCA algorithm for hyperspectral images using RVC-CAL”, SPIE Remote Sensing, International Society for Optics and Photonics, 2016.

[28] J. Eker and J. W. Janneck, “Cal language report,” Tech. Rep. UCB/ERL M03/48, University of California at Berkeley, 2003.

[29] K.K. Matam et al., "Evaluating energy efficiency of floating point matrix multiplication on FPGAs." High Performance Extreme Computing Conference (HPEC), pp. 1-6, 2013.

Raquel Lazcano received her B.Sc. degree in Communication Electronics Engineering from Universidad Politécnica de Madrid (UPM), Spain, in 2014, and her M.Sc. degree in Systems and Services Engineering for the Information Society from Universidad Politécnica de Madrid (UPM), Spain, in 2015. She is currently a student of the PhD degree in Systems and Services Engineering for the Information Society at the Electronic and Microelectronic Design Group (GDEM), UPM. In 2015, she stayed 4 months at the Institute of Electronics and Telecommunications of Rennes (IETR), at the National Institute of Applied Sciences (INSA), France, as an interchange student of the M.Sc. degree. Her research interests include high-performance multicore processing systems, real-time hyperspectral image processing and the automatic optimization of the parallelism in real-time systems. She is author or co-author of 9 contributions to technical conferences.

Daniel Madroñal received his B.Sc. degree in Communication Electronics Engineering from Universidad Politécnica de Madrid (UPM), Spain, in 2014, and his M.Sc. degree in Systems and Services Engineering for the Information Society from Universidad Politécnica de Madrid (UPM), Spain, in 2015. He is currently a student of the PhD degree in Systems and Services Engineering for the Information Society at the Electronic and Microelectronic Design Group (GDEM), UPM. In 2015, he stayed 4 months at the National Institute of Applied Sciences (INSA), France, as an interchange student of the M.Sc. degree. His research interests include high-performance multicore processing systems, real-time hyperspectral image processing and the automatic optimization of the energy consumption in high-performance systems. He is author or co-author of 9 contributions to technical conferences.

Rubén Salvador (PhD'2015–MSc'2007 Industrial Electronics, UPM) is currently an Assistant Professor at the department of Telematics and Electronics Engineering and member of the Electronics and Microelectronics design group (GDEM), at the Research Center on Software Technologies and Multimedia Systems for Sustainability (CITSEM), Universidad Politécnica de Madrid (UPM). Previously, he was a Research Assistant at the Center of Industrial Electronics (CEI, UPM, 2006-2011) where he obtained his PhD (cum laude). Before that, he was a researcher at the Intelligent Vehicle Systems division of the University Institute for Automobile research (INSIA, UPM, 2005-2006). In 2009 he was a visiting research student at the Department of Computer Systems, Brno University of Technology.

He is author or coauthor of more than 6 indexed journals and 20 refereed conference papers and one book chapter. He serves as technical program committee member of international conferences like ReCoSoC and DASIP and is a

reviewer of several international journals and conferences. Besides, he is an IEEE and ACM member. His current research interests are focused in the embedded systems domain including reconfigurable, heterogeneous and parallel computing for FPGAs and manycore accelerators, with a focus on system self-adaptation, HW/SW dynamic reconfiguration techniques, evolvable hardware and approximate computing. He is interested in design tools and specification methods for highly parallel and energy efficient accelerators for embedded and high performance computing systems. The applications targeted span the biomedical and space fields, using hyperspectral image processing and machine learning techniques for diagnostic imaging systems and system self-adaptation in harsh environments.

Karol Desnos is a Teaching Assistant at the National Institute of Applied Science (INSA) of Rennes and Maître de Conférences. He carries is research under the supervision of Pr. Jean-François Nezan and Dr. Maxime Pelcat, and in collaboration with Texas Instrument France. Karol Desnos received his Master of Engineering in Electronics and Computer Engineering from the INSA of Rennes in 2011 and his Ph. D in 2014. In Fall 2012, Karol Desnos was a visiting researcher at the University of Maryland in the research group led by Pr. Shuvra Bhattacharyya. Since 2012, he is a member of the Multicore Association and takes part in the working group on the Multicore Task Management API (MTAPI). His research interests focus on dataflow models of computation and associated implementation techniques for heterogeneous MPSoCs.

Maxime Pelcat is an associate professor at the Department of Electrical and Computer Engineering at the National Institute of Applied Sciences (INSA) in Rennes. He holds a joint appointment in the Institute of Electronics and Telecommunications of Rennes (IETR), a CNRS research unit. His main research interests include dataflow models, multimedia and telecommunication processing, and programming of distributed embedded systems. Maxime Pelcat is a member of the HiPEAC network. He obtained his Ph.D. in signal processing from INSA in 2010. The Ph.D. thesis resulted from a collaboration of INSA Rennes and Texas Instruments, Nice, and followed a contract as research engineer. Previously, after one year in the Audio and Multimedia department at Fraunhofer-Institute IIS in Erlangen, Germany, he worked as a contractor at France Telecom Research and Development.

Raúl Guerra was born in Las Palmas de Gran Canaria, Spain, in 1988. He received the industrial engineer degree by the University of Las Palmas de Gran Canaria in 2012. In 2013 he received the master degree in telecommunications technologies imparted by the Institute of Applied Microelectronics, IUMA. He has been funded by this institute to do his PhD research in the Integrated System Design Division. His current research interests include the parallelization of algorithms for multispectral and hyperspectral images processing and hardware implementation.

Himar Fabelo, received the telecommunication engineer degree and the master degree in telecommunication technologies from the University of Las Palmas de Gran Canaria, Spain, in 2013, and 2014, respectively. Since then, he has conducted his research activity in the Integrated System Design Division at the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, in the field of electronic and bioengineering. In 2015 he started to work as a coordination assistant and researcher in the HELICoiD European project, co-funded by the European Commission. His current research interests are in the use of classification algorithms together with hyperspectral images to detect human brain cancer using in-vivo hyperspectral images. He obtained the best paper award in the Jornadas de Computación Empotrada (JCE2015) in September of 2015 with the paper entitled “HELICoiD Demonstrator for Intraoperative Brain Cancer Detection using Hyperspectral Images”.

Samuel Ortega, received the telecommunication engineer degree and the research master degree in telecommunication technologies from the University of Las Palmas de Gran Canaria, Spain, in 2014, and 2015, respectively. Since then, he has conducted his research activity in the Integrated System Design Division at the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria, in the field of electronic and bioengineering. In 2015 he started to work as a coordination assistant and researcher in the HELICoiD European project, co-funded by the European Commission. His current research interests are in the use of machine learning algorithms in medical applications using hyperspectral images.

Sebastian Lopez was born in Las Palmas de Gran Canaria, Spain, in 1978. He received the Electronic Engineer degree by the University of La Laguna in 2001, obtaining regional and national awards for his CV during his degree. He got his PhD degree by the University of Las Palmas de Gran Canaria in 2006, where he is currently an Associate Professor, developing his research activities at the Integrated Systems Design Division of the Institute for Applied Microelectronics (IUMA). He is a member of the IEEE Geoscience & Remote Sensing and Consumer Electronics Societies as well as an associate editor of the IEEE Transactions on Consumer Electronics. Additionally, he currently serves as an active reviewer of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing (JSTARS), IEEE Transactions on Geoscience and Remote Sensing, IEEE Geoscience and Remote Sensing Letters, IEEE Transactions on Circuits and Systems for Video Technology, the Journal of Real Time Image Processing, Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), and the IET Electronics Letters, among others. He is also a program committee member of different international conferences including the SPIE Conference on Satellite Data Compression, Communication and Processing, IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), and SPIE Conference of High Performance Computing in Remote Sensing. Furthermore, he has been designated as the program co-chair of the last two aforementioned conferences for their upcoming editions in 2013. He has published more than 60 papers in international journals and conferences. His research interests include real-time hyperspectral imaging systems, reconfigurable architectures, video coding standards, and hyperspectral compression systems.

Gustavo M. Callicó received the Telecommunication Engineer degree in 1995 and the Ph.D. degree and the European Doctorate in 2003, all from the University of Las Palmas de Gran Canaria (ULPGC) and all with honours. From 1996 to 1997 he was granted with a national research grant from the Educational Ministry and in 1997 he was hired by the ULPGC as an electronic lecturer. In 1994 he joined the Institute for Applied Microelectronics (IUMA) and from 2000 to 2001 he did a stay at the Philips Research Laboratories (NatLab) in Eindhoven, The Netherlands, as a visiting scientist, where he developed his Ph.D. thesis. He is currently an Associate Professor at the ULPGC and develops his research activities at the Integrated Systems Design Division of the IUMA. He has more than 90 publications in national and international journals and conferences and has participated in 17 research projects funded by the European Community, the Spanish Government and international private industries. He is a member of the Consumer Electronics Society as well as a member of the Publications Review Committee of the IEEE Transactions on Consumer Electronics. Additionally, he currently serves as an active reviewer of the ACM Transactions on Design Automation of Electronic Systems, the Electronics and Telecommunications Research Institute (ETRI), the EURASIP Journal on Embedded Systems, the SPIE Journal of Electronic Imaging, the SPRINGER Journal of Real-Time Image Processing and the ELSEVIER Journal of Microprocessors and Microsystems: Embedded Hardware Design. His current research fields include real-time super-resolution algorithms, synthesis-based design for SOCs, real-time hyperspectral imaging systems and circuits for multimedia processing and video coding standards.

Eduardo Juárez (M'96) received the Ingeniero de Telecomunicación degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain in 1993 and the Docteur ès Sciences Techniques degree from the École Polytechnique Fédéral de Lausanne (EPFL), Switzerland in 2003. In 1994, he joined the Digital Architecture Group (GAD) of the UPM as a researcher. In 1998, he joined the Integrated Systems Laboratory (LSI) of the EPFL as an Assistant. In 2000, he joined Transwitch Corp., Switzerland, as a Senior System Engineer. In 2004, he joined the Electronic and Microelectronic Design Group (GDEM) as a post-doctoral researcher. Since 2013, he has been a researcher in the Research Centre of Software Technologies and Multimedia Systems (CITSEM). His current interests are related to the field of low power hyperspectral embedded imaging systems for health applications.

C. Sanz (S'87-M'88-SM'13) received both his Ingeniero de Telecomunicación degree (1989) and his Doctor Ingeniero de Telecomunicación degree (1998) from the Universidad Politécnica de Madrid (Spain). Since 2008, he has been the director of the E.U.I.T. de Telecomunicación and currently leads the Electronic and Microelectronic Design Group (GDEM) involved in R&D projects with Spanish and European companies and public institutions. His areas of interest are microelectronic design applied to image coding, digital TV, digital video broadcasting and hyperspectral imaging.