FPGA for computing the pixel purity index algorithm on hyperspectral images

Carlos González, Daniel Mozos
Departament of Computer Architecture and Automatics
Computer Science Faculty
Complutense University of Madrid
Emails: carlosgonzalez@fdi.ucm.es, mozos@fis.ucm.es

Javier Resano
Departament of Computer Architecture
High Polytechnic Center
University of Zaragoza
Email: jresano@unizar.es

Antonio Plaza
Departament of Computer Technology and Communications
Polytechnic School of Cáceres
University of Extremadura
Email: aplaza@unex.es

***Abstract*—The pixel purity index algorithm is employed in remote sensing for analyzing hyperspectral images. A single pixel usually covers several different materials, and its observed spectrum can be expressed as a linear combination of a few pure spectral signatures. This algorithm tries to identify these pure spectra. In this paper, we present a Field Programmable Gate Array implementation of the algorithm, which has been implemented on a Virtex-II PRO XC2VP30 and on a Virtex-4 XC4VFX60 FPGAs and evaluated using the well-known "Cuprite" image (a standard benchmark in hyperspectral imaging applications). Our experimental results demonstrate that a hardware version of the PPI algorithm can significantly outperform an equivalent software version of the algorithm and retain excellent pure spectral extraction accuracy. In addition, the proposed architecture is easily scalable depending of the available resources and is more than three times faster than a recently developed FPGA implementation of the same algorithm due to the architectural improvements.**

I. Introduction

Hyperspectral imaging is a new technique in remote sensing that generates images with hundreds of spectral bands, at different wavelength channels, for the same area on the surface of the Earth (see Figure 1). Hyperspectral imaging is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airborne or satellite sensor. The concept of hyperspectral imaging originated at NASAs Jet Propulsion Laboratory in California, which developed instruments such as the Airborne Imaging Spectrometer (AIS), then called AVIRIS, for Airborne Visible Infra-Red Imaging Spectrometer.

Most of the pixels collected by hyperspectral imagers contain the resultant mixed spectrum from the reflected surface radiation of subpixel constituent materials within the pixel. Mixed pixels exist for several reasons. First, if the spatial resolution of the sensor is not high enough to separate different pure signature materials at a macroscopic level, these can jointly occupy a single pixel, and the resulting spectral measurement is a composite of the individual spectra. Second, mixed pixels can also result when distinct materials are combined into a homogeneous mixture. This circumstance also occurs independent of the spatial resolution of the sensor.

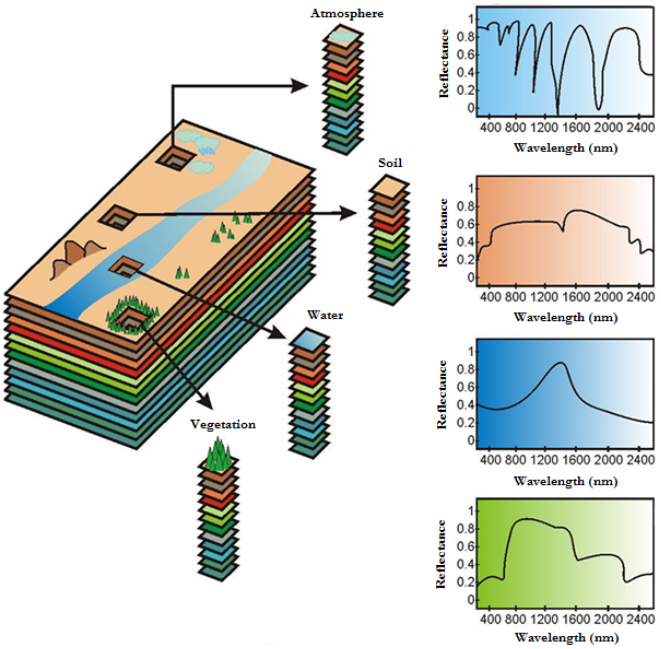

Fig. 1. The concept of hyperspectral imaging.

Over the last years, many algorithms have been developed with the purpose of finding "spectral endmembers" [1], which are assumed to be pure signatures in remotely sensed hyperspectral data sets. Such pure signatures can then be used to estimate the abundance or concentration of materials in mixed pixels, thus allowing sub-pixel analysis which is crucial in many remote sensing applications due to current sensor optics and configuration. The pixel purity index (PPI) algorithm has been widely used in hyperspectral image analysis for endmember extraction due to its publicity and availability in ITTVIS Environment for Visualizing Images (ENVI) software originally developed by Analytical Imaging and Geophysics (AIG). This algorithm is very time consuming, a fact that has generally prevented its exploitation in valid response times in a wide range of applications, including environmental monitoring, military applications or hazard and threat assess-

ment/tracking.

The trend in remote sensing missions has always been towards using hardware devices with smaller size, lower cost, more flexibility, and higher computational power. On-board processing, as a solution, allows for a good reutilization of expensive hardware resources. Instead of storing and forwarding all captured images, remote sensing data interpretation can be performed on-orbit prior to downlink, resulting in a significant reduction of communication bandwidth as well as simpler and faster subsequent computations to be performed at ground stations. In this regard, FPGAs combine the flexibility of traditional microprocessors with the power and performance of application-specific integrated circuits (ASICs). Therefore, FPGAs are a promising candidate for on-board remote sensing data processing [3].

II. The Pixel Purity Index Algorithm

Since the details of the specific steps to implement ENVI's PPI are not available in the literature, the PPI algorithm described below is only based on the limited published results and our own interpretation [4]. Nevertheless, except a final manual supervision step (included in ENVIs PPI) which is replaced by step 4, both our approximation and the PPI in ENVI 4.0 produce very similar results. The inputs to the algorithm are a hyperspectral data cube $\mathbf{F}$ with D dimensions; the number of random skewers to be generated during the process, K; and a cut-off threshold value, t_v, used to select as final endmembers only those pixels that have been selected as extreme pixels at least t_v times throughout the PPI process.

The algorithm is given by the following steps:

1) Initialization: Generate a set of K unit vectors called "skewers" $\{\mathbf{skewer}_j\}_{j=1}^{K}$ randomly, where K is a sufficiently large positive integer.

2) PPI Calculation: For each $\mathbf{skewer}_j$, all the data sample vectors are projected onto $\mathbf{skewer}_j$ to find sample vectors al its extreme positions and form an extrema set for this particular $\mathbf{skewer}_j$, denoted by $S_{extrema}(\mathbf{skewer}_j)$. Despite the fact that a different $\mathbf{skewer}_j$ generates a different extrema set $S_{extrema}(\mathbf{skewer}_j)$, it is very likely that some sample vectors may appear in more than one extrema set. Define an indicator function of a set S, $I_S(\mathbf{r})$ by

$$I_S(\mathbf{r}) = \begin{Bmatrix} 1 \text{ if } \mathbf{r} \in S \\ 0 \text{ if } \mathbf{r} \notin S \end{Bmatrix} \tag{1}$$

and

$$N_{PPI}(\mathbf{r}) = \sum_{j=1}^{k} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{r}) \tag{2}$$

where $N_{PPI}(\mathbf{r})$ is defined as the PPI score of sample vector r.

3) Candidate Selection: Find the PPI scores $N_{PPI}(\mathbf{r})$ for all the sample vectors defined by (2).

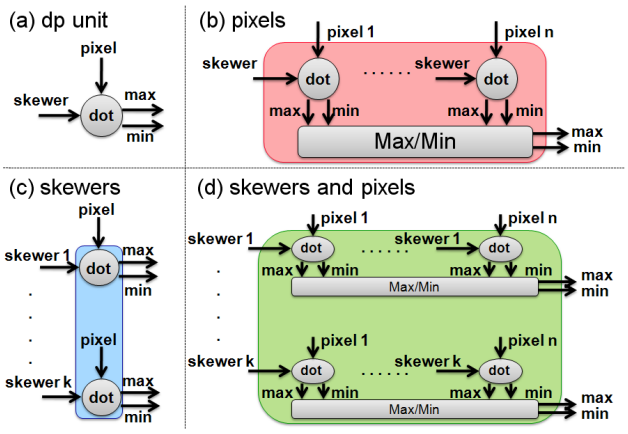

Fig. 2. (a) Dot-product (dp) unit. (b) Parallelization strategy by pixels. (c) Parallelization strategy by skewers. (d) Parallelization strategy by skewers and pixels.

4) Endmember Extraction: Let t_v be a threshold value set for the PPI score. Extract all the sample vectors with $N_{PPI}(\mathbf{r}) \geq t_v$.

In this algorithm, the most time consuming part is the calculation of the projections (using a dot-product). Fortunately, the PPI algorithm is well suited for parallel implementation. The computation of skewer projections are independent and can be performed simultaneously, leading to many ways of parallelization.

III. Parallel Design Strategies

The PPI algorithm computes a very large number of dot-products, all of which can be performed simultaneously. In [5], [6], two parallel architectures, based on a 2-D processor array, to implement the PPI are proposed. Although these works have demonstrated the efficiency of a hardware implementation on a reconfigurable board, these solutions are not scalable and easily portable. The proposed architecture can be easily adapted to different platforms and is scalable depending on the amount of available resources because the required resources grow proportionally with the number of dot-product units and the clock cycle remains constant.

If we consider a simple dot-product unit such as the one displayed in Figure 2(a) as the baseline for parallel computations, then we can perform the parallel computations by pixels [Figure 2(b)], by skewers [Figure 2(c)], or by pixels and skewers [Figure 2(d)]. If we parallelize the computations by pixels, additional hardware is necessary to compare all the maxima and minima between them. As we increase the number of parallel computations, a greater area would be required for maxima/minima computations and the critical path would be longer, hence the clock cycle would be higher. Another possible way to parallelize the extreme projections stage is to compute K dot-products at the same time for the same pixel, where K is the number of skewers [Figure 2(c)]. If we increase the number of skewers in this case, the required area would grow proportionally with the number of

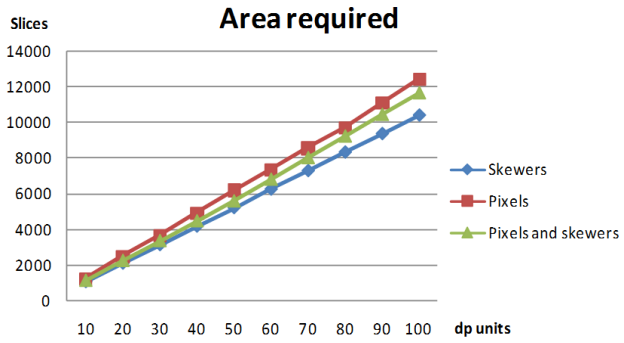

Fig. 3. Area required (in slices) by dot-processing units in different parallel implementation strategies of the PPI.

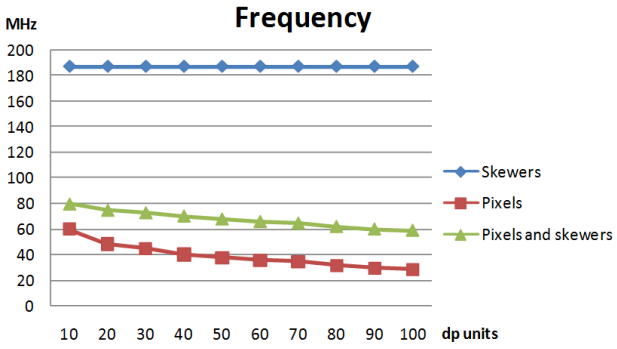

Fig. 4. Clock cycle required (in MHz) by dot-processing units in different parallel implementation strategies of the PPI.

dot-product units and the clock cycle would remain constant. Finally, the parallelization strategy in Figure 2(d) is a mixed solution which provides no further advantage with respect to the parallelization by skewers and has the same problems that parallelization by pixels.

To emphasize the advantages of the considered parallelization strategy over other possible alternatives, Figures 3 and 4 compare the three parallelization strategies. We have implemented the three parallelization strategies in the considered analysis scenario and further evaluated the number of slices required by the dot-processing units (see Figure 3) and the clock cycle required by the same units (see Figure 4). Parallelization by skewers offers significant advantages with regards to the other considered implementation strategies.

Apart from the aforementioned advantages with regard to other possible strategies, another reason for this selection is that the parallelization strategy based on skewers fits very well how the image data reaches the system. In our case, our goal is to make an on-line processing of the hyperspectral images bearing in mind that hyperspectral sensors capture the image data in a pixel by pixel fashion. Therefore, parallelization by skewers is the one that best fits the data entry mechanism since each pixel can be processed immediately as collected.

IV. Proposed Architecture

The general structure of the pipeline architecture designed for implementing the PPI is summarized in Figure 5. We can

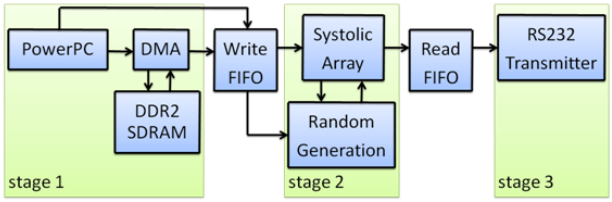

Fig. 5. Fig. 3. General scheme of the pipelined architecture.

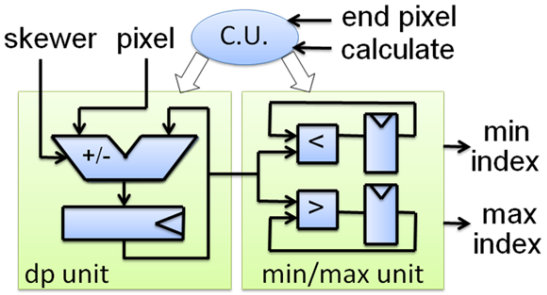

Fig. 6. Hardware architecture of a dot-product processor.

distinguish three stages which are communicated using FIFOs: The first stage provides the necessary data for the system, the second stage calculates the projections and finally the endmembers are sent via a RS232 port by the third stage. Therefore, all stages work in parallel.

The PPI algorithm computes a very large number of dot-products, and all these dot-products can be performed simultaneously. A possible way to parallelize them is to have a system able to compute K dot-products at the same time against the same pixel. We use a systolic array design to implement this strategy. Here, a systolic cycle consists in computing a dot-product between a pixel and a skewer, and to store the index of the pixel if the dot-product is higher or smaller than a previously computed max or min value. In this context both a pixel and a skewer are vectors of D values. A dot-product calculation (dp) is performed as follows:

$$dp = \sum_{i=1}^{D} pixel[i] \times skewer[i] \quad (3)$$

Skewer values can be limited to a very small set of integers when D is large, as in the case of hyperspectral images. A particular and interesting set is $\{1, -1\}$ since it avoids the multiplications. Figure 6 shows the architecture of a dot-product processor using this idea. The dp unit computes the dot-product by adding positive or negative pixel values. The min/max unit performs a comparison between a min and a max value and stores the pixel number if the dot-product exceeds one of these two extreme values.

V. Experimental Results

The systolic array has been designed using VHDL and the Embedded Development Kit environment to specify the complete system. It has been implemented on a Virtex-II PRO

TABLE I
SPECTRAL ANGLE-BASED SIMILARITY SCORES BETWEEN THE ENDMEMBERS EXTRACTED BY DIFFERENT IMPLEMENTATIONS OF PPI AND THE SELECTED USGS REFERENCE SIGNATURES

USGS Mineral	ENVI software	PPI approximation	our FPGA-based PPI
Alunite	0.084	0.084	0.084
Buddingtonite	0.071	0.068	0.068
Calcite	0.089	0.089	0.089
Kaolinite	0.136	0.132	0.132
Muscovite	0.092	0.081	0.081

TABLE II
COMPARISON OF DIFFERENT IMPLEMENTATIONS OF THE PPI ALGORITHM

Algorithm	Platform	Procesing time (seconds)
C++ PPI aproximation	AMD Athlon 2.6 GHz	3068
FPGA implementation in [5]	Virtex-II XC2V6000-6 (33792 slices)	62
Proposed FPGA implementation	Virtex-II PRO XC2VP30 (13696 slices)	31
Proposed FPGA implementation	Virtex-4 XC4VFX60 (25280 slices)	17

XC2VP30 and on a Virtex-4 XC4VFX60 commercial FPGAs because they have the same architecture and similar area than radiation-hardened FPGAs that have been certified by several international agencies for remote sensing applications and are commonly used in airborne and space borne Earth Observation platforms.

The hyperspectral dataset used in these experiments is the well-known Cuprite scene, collected by NASA Jet Propulsion Laboratory's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS). The scene comprises a relatively large area (350 lines by 350 samples) and 224 spectral bands. The site is well understood mineralogically, and has several exposed minerals of interest including alunite, buddingtonite, calcite, kaolinite and muscovite. Reference ground signatures of the above minerals, available in the form of a U.S. Geological Survey library (USGS) will be used to assess endmember signature purity in this work.

We first conducted an experiment-based cross examination of endmember extraction accuracy to assess the spectral similarity between the USGS library spectra and the corresponding endmembers extracted by the considered implementation of the PPI algorithm. Table I shows the spectral angle distance (SAD) between the most similar endmembers detected by the original ENVI implementation, the PPI algorithm approximation in C++, and our FPGA-based implementation after evaluating 10^4 skewers. It should be noted that the SAD between a pixel vector $\mathbf{f}_i$ selected by the PPI and a reference spectral signature $\mathbf{s}_j$ is given by:

$$\text{SAD}(\mathbf{f}_i, \mathbf{s}_j) = \cos^{-1} \frac{\mathbf{f}_i \cdot \mathbf{s}_j}{\|\mathbf{f}_i\| \cdot \|\mathbf{s}_j\|} \quad (4)$$

where smaller SAD values indicate higher spectral similarity. For all the cases our approach obtains the best results.

For illustrative purposes, we have performed a comparison of our proposed FPGA design with previous implementations in terms of computation time. Table II shows the platform used and the computing time for three different implementations: a SW PPI implementation, an FPGA-based implementation presented in [4], and the FPGA-based implementation proposed in this work. The FPGA-based implementation in [4] was more than 49 times faster than the PPI approximation for the AVIRIS Cuprite image, while our FPGA implementation of the PPI shows a speedup of 3,5 with regard to that implementation in [4].

VI. CONCLUSION

On-board data processing of hyperspectral imagery has been a long-awaited goal by the remote sensing community. The number of applications requiring a response in real-time has been growing exponentially in recent years, and FPGAs offer the necessary flexibility and performance to carry out this process. In this paper, we have described a FPGA implementation of the pixel purity index algorithm, one of the most well-known approaches for hyperspectral data analysis in the remote sensing community. Our experimental results, conducted on a on a Virtex-II PRO XC2VP30 and on a Virtex-4 XC4VFX60 architecture, demonstrate that our architecture is easily scalable and portable with highly satisfactory endmember extraction accuracy. Further, our proposed hardware version of the PPI algorithm can significantly outperform (in terms of computation time) a SW approximation of the algorithm and a recently developed FPGA implementation. This work leaves the door open to the execution in near real-time of remote sensing applications such as wildland fire detection, oil spill maping and chemical and biological standoff detection.

ACKNOWLEDGMENT

This research was supported by TIN2009-09806 and AYA2009-13300-C03-02.

REFERENCES

[1] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, pp. 650–663, 2004.

[2] J. Boardman, "Automating spectral unmixing of AVIRIS data using convex geometry concepts," *Summaries of Airborne Earth Science Workshop,* JPL Publication 93-26, pp. 111–114, 1993.

[3] T. A. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. V. Kindratenko and D. A. Buell, "The promise of high-performance reconfigurable computing," *IEEE Computer*, vol. 41, pp. 69–76, 2008.

[4] C.-I Chang and A. Plaza, "A fast iterative algorithm for implementation of Pixel Purity Index," *IEEE Geoscience and Remote Sensing Letters*, vol. 3, pp. 63–67, 2006.

[5] D. Valencia, A. Plaza, M. A. Vega-Rodriguez and R. Perez, "FPGA design and implementation of a fast pixel purity index algorithm for endmember extraction in hyperspectral imagery," *Proceedings of SPIE*, vol. 5995, pp. 599508.1–599508.10, 2005.

[6] D. Lavernier, J. Theiler, J. Szymanski, M. Gokhale and J. Frigo, "FPGA implementation of the Pixel Purity Index algorithm," *Proceedings of SPIE*, vol. 4693, pp. 30–41, 2002.