FPGA Design and Implementation of a Fast Pixel Purity Index Algorithm for Endmember Extraction in Hyperspectral Imagery

David Valencia[a], Antonio Plaza[a], Miguel A. Vega-Rodríguez[b], and Rosa M. Pérez[a]

[a]Neural Networks and Signal Processing Group (GRNPS)
[b]Computer Architecture and Logic Design Group (ARCO)
Computer Science Department, University of Extremadura
Avda. de la Universidad s/n, E-10071 Cáceres, Spain
E-mail: {davaleco, aplaza, mavega, rosapere}@unex.es

ABSTRACT

Hyperspectral imagery is a class of image data which is used in many scientific areas, most notably, medical imaging and remote sensing. It is characterized by a wealth of spatial and spectral information. Over the last years, many algorithms have been developed with the purpose of finding "spectral endmembers," 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. One of the most popular endmember extraction algorithms has been the pixel purity index (PPI), available from Kodak's Research Systems ENVI software package. 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 assessment/tracking (including wildland fire detection, oil spill mapping and chemical and biological standoff detection). Field programmable gate arrays (FPGAs) are hardware components with millions of gates. Their reprogrammability and high computational power makes them particularly attractive in remote sensing applications which require a response in near real-time. In this paper, we present an FPGA design for implementation of PPI algorithm which takes advantage of a recently developed fast PPI (FPPI) algorithm that relies on software-based optimization. The proposed FPGA design represents our first step toward the development of a new reconfigurable system for fast, onboard analysis of remotely sensed hyperspectral imagery.

Keywords: Field programmable gate arrays (FPGAs), Pixel purity index (PPI), Hyperspectral, Endmember extraction.

1. INTRODUCTION

The term "spectral endmember" is well-known in the remote sensing community, and can be defined as an idealized, pure spectral signature for a class[1]. This concept is mainly used in hyperspectral imaging, and should be distinguished from the concept of "pure pixel" traditionally adopted in other types of studies, such as those based on multispectral sensors which typically collect information in only a few spectral bands. As opposed to that concept, the term "pixel" in hyperspectral imaging refers to an L-dimensional vector given by hundreds of spectral values corresponding to different wavelengths[2]. It should be noted that an endmember is generally not a pixel. It is a spectral signature that is completely specified by the spectrum of a single material substance. Accordingly, a pixel vector in hyperspectral imaging is generally called "pure" if its spectral signature is an endmember. The Pixel Purity Index (PPI) algorithm developed by Boardman et al.[3] has been one of the most widely used endmember extraction algorithms for hyperspectral data exploitation. One of the main reasons for the success of PPI in the remote sensing community is its publicity and availability in the Kodak's Research Systems ENVI software package[4]. Due to this propriety, its detailed implementation has never been available in the literature, and thus, several implementations based on limited published results have been proposed. The overall nature of the algorithm is supervised. First, a pixel purity score is calculated for each point in the image cube by generating k random unit L-dimensional vectors called "skewers." All the points in the original L-dimensional space comprised by the input data are then projected onto the skewers, and the ones falling at the extremes

Chemical and Biological Standoff Detection III, edited by James O. Jensen, Jean-Marc Thériault,
Proc. of SPIE Vol. 5995, 599508, (2005) · 0277-786X/05/$15 · doi: 10.1117/12.631270

of each skewer are tallied. After many repeated projections to different random skewers, those pixels selected a number of times above a certain cut-off threshold, *t*, are declared "pure" and loaded into an interactive "*L*-dimensional visualization tool" (available as a built-in companion piece in ENVI software) and manually rotated until a desired number of endmembers, *p*, are visually identified as extreme pixels in the *L*-dimensional data cloud.

The PPI algorithm suffers from several drawbacks, as indicated in a previous study[5]. A relevant shortcoming is the algorithm's sensitivity to parameters *k* (number of skewers) and *t* (cut-off threshold value). Since the skewers are randomly generated, it is generally required that a very large number of iterations be generated to arrive to a satisfactory endmember set in terms of signature purity. Another major issue is the algorithm's computational complexity. For instance, the PPI algorithm available in ENVI 4.0 version took more than 50 minutes to project every data sample vector of the well-known "Cuprite" hyperspectral scene that will be used later on in experiments (the image consists of 614 samples, 512 lines, 224 spectral bands and a total size of 137 MB) onto 10^4 skewers in a PC with AMD Athlon 2.6 GHz processor and 512 MB of RAM. This very high number of skewers was required to arrive to a reasonable endmember set. Another shortcoming of the PPI is the requirement of human intervention to manually select a final set of endmembers by visual inspection. Most importantly, the PPI is not an iterative process and does not guarantee its convergence in finite runs in spite of that it may converge asymptotically as claimed by the authors[3].

Recently, Chang and Plaza[6] have proposed a fast software-based implementation of the PPI, called fast PPI (FPPI). It has several advantages over the PPI. First, it makes use of a newly developed concept, virtual dimensionality $(VD)^2$ to estimate the number of endmembers, *p*, required to be generated. This allows for the replacement of the two parameters, *k* and *t* used in the PPI so that the issue of their sensitivity to number of runs and cut-off threshold value is resolved. Second, the FPPI takes advantage of the automatic target generation process $(ATGP)^2$ implemented in the automatic target detection and classification algorithm $(ATDCA)^2$ to generate an appropriate set of initial endmembers that can reduce a significant number of iterations required for the PPI. Third, it provides a new iterative rule and a stopping rule for the algorithm. Most importantly, unlike the PPI which requires a visualization tool to manually select a final set of endmembers, the FPPI is completely automatic and unsupervised. It is terminated as long as the implemented stopping rule is met. Even though the FPPI is able to produce a response in about three minutes for the same hyperspectral scene described above in the same computing environment (even though a Matlab-based implementation was used), certain applications require that the response to the endmember extraction problem for a scene is provided in real time. Such is the case of military applications, or those dealing with environmental monitoring or hazard/threat assessment and tracking, including wildland fire detection, oil spill mapping and chemical and biological standoff detection[1].

In this paper, we develop a hardware-based version of the FPPI algorithm to increase its computational performance by resorting to FPGA reconfigurable boards. The proposed FPGA design represents our first step toward the development of a system for onboard analysis of hyperspectral imagery. It should be noted that most currently available parallel processing systems in remote sensing are directed towards the exploration of massive data archives which have been already transmitted to Earth, while a highly desirable goal is to have the data processed in real time and before its transmission to Earth. This may allow scientists to overcome an existing limitation in many remote sensing and observatory systems, i.e., the bottleneck introduced by the bandwidth of the downlink connection from the observatory platform. The proposed methodology was designed in Handel-C[7] language for high-level prototyping, using Celoxica's DK 3.1 tool[8]. Then, it was implemented on a Virtex-II FPGA from Xilinx, Inc.[9] The remainder of the paper is organized as follows. Section 2 describes the FPPI algorithm and our proposed parallelization strategy. Section 3 describes a systolic-array based implementation for the FPPI. Section IV briefly discusses the Handel-C code. Section V presents an experimental assessment of both software- and hardware-optimized versions of FPPI with regards to the original PPI algorithm available in ENVI software. Section VI concludes with some remarks and future research lines.

2. FPPI ALGORITHM AND PARALLELIZATION STRATEGIES

Before describing our parallel algorithm, we first provide a step-by-step description of the original FPPI algorithm. Although the algorithm makes use of the VD concept[2] and the ATGP algorithm[5] to improve the original PPI, both techniques have not been implemented in parallel in this work. As a result, we assume that the number of endmembers, *p*, to be extracted by the algorithm is known in advance, and that a set of *p* intelligently generated skewers is already available. Future work should be directed towards the parallelization of both techniques. With the above assumptions in mind, a step-by-step description of the FPPI algorithm in[6] is provided below.

FPPI Algorithm

1. *Initialization:*
   *A*ssume that p, the number of endmembers required to generate is estimated by the VD concept, and let $\{\boldsymbol{skewer}_j^{(0)}\}_{j=1}^{p}$ be an initial set of p skewers generated by ATGP algorithm.

2. *Iterative rule:*
   At iteration $k \geq 0$, for each $\boldsymbol{skewer}_j^{(k)}$ project all the data sample vectors onto this particular $\boldsymbol{skewer}_j^{(k)}$ to find those sample vectors that are at its extreme positions to form an extrema set, denoted by $S_{extrema}(\boldsymbol{skewer}_j^{(k)})$. Despite the fact that a different $\boldsymbol{skewer}_j$ generates a different extrema set $S_{extrema}(\boldsymbol{skewer}_j^{(k)})$, it is very likely that some sample vectors may appear in more than one extrema set. Define an indicator function of a set S, $I_S(\boldsymbol{r})$ by the following expression:

$$I_S(\boldsymbol{r}) = \begin{cases} 1; \text{if } \boldsymbol{r} \in S \\ 0; \text{if } \boldsymbol{r} \notin S \end{cases} \text{ and } N_{PPI}(\boldsymbol{r}) = \sum\nolimits_j I_{S_{extrema}(\boldsymbol{skewer}_j^{(k)})}(\boldsymbol{r}). \quad (1)$$

3. *Pure pixel candidate selection:*
   Find the sample vectors, denoted by $\{\boldsymbol{r}_j^{(k)}\}_{j=1}^{p}$, that produce the first p largest values of $N_{PPI}(\boldsymbol{r})$.

4. *Stopping rule:*
   Find the p largest values of N_{PPI} in the joint set of $\{N_{PPI}(\boldsymbol{r}_j^{(k)})\}_{j=1}^{p}$ Y $\{N_{PPI}(\boldsymbol{skewer}_j^{(k)})\}_{j=1}^{p}$ and let $\{N_{PPI}(\boldsymbol{skewer}_j^{(k+1)})\}_{j=1}^{p}$ be those vectors corresponding to the obtained p largest values of N_{PPI}, denoted by $\{\boldsymbol{e}_j^{(k+1)}\}_{j=1}^{p}$. If $\{N_{PPI}(\boldsymbol{skewer}_j^{(k+1)})\}_{j=1}^{p} = \{N_{PPI}(\boldsymbol{skewer}_j^{(k)})\}_{j=1}^{p}$, that is, $\{\boldsymbol{e}_j^{(k+1)}\}_{j=1}^{p} = \{\boldsymbol{e}_j^{(k)}\}_{j=1}^{p}$, then there is no more endmember has been replaced. In this case, the algorithm is terminated. Otherwise, let $k \leftarrow k+1$ and go to step 2.

```
while (not end)
   for every skewer in the VD do
      max_value = MINIMUM_VALUE
      pos = INITIAL_POINT (x,y)
      for every line do
         for every column in a line do
            result = dot-product(sample,skewer);
            if result > max_value then
               Tally extreme pixel and its spatial coordinates (x,y)
            endif
         endfor
      endfor
      for each pixel in the list do
         increment PPI count of the pixel at spatial coordinates (x,y)
      endfor
   endfor
   change set of skewers according to results
   evaluate stopping rule
   if (stopping rule = true) then
      end = true
   endif
endwhile
```

Table 1. Pseudo-code description of steps 2-4 in the FPPI algorithm.

In order to implement steps 2-4 of the FPPI algorithm in parallel, we first investigated which parts of the algorithm are likely to represent the most significant fraction of the execution time. Obviously, projecting a large number of data sample vectors onto a set of skewers via dot products is a highly parallelizable operation since there are no data dependences involved. Therefore, our first approximation to parallel designed was aimed at speeding up computational performance of the highly time-consuming process accomplished in step 2. Table 1 provides a pseudo-code of the calculations carried out by this step. After analyzing the code in Table 1, we can adopt two main strategies for parallelization. A first option would be to partition the dataset (and the main loop), thus maintaining the spectral identity of each pixel to avoid having individual pixels split among processing elements. This option is particularly appealing in remote sensing applications, where the minimum calculation unit is generally the spectral signature contained in a pixel vector *as a whole*. On other hand, partitioning the data in terms of spectral information may also introduce a significant communication overhead and additional data dependences, as demonstrated in our recent study[10]. As a result, we generally prefer to keep the spectral identity of both skewers and pixels instead of breaking them into several processing elements. This is expected to introduce less requirements in terms of FPGA resources. With the above ideas in mind, Table 2 shows a pseudo-code description of the parallel code used in this work.

```
while (not end)
    for every partition of the image do
       for each sample in partition do in parallel
          for each skewer in the set do in parallel
             Result_ij=dot-product_ij(sample_i,skewer_j)
          endfor
       endfor
       Calculate local extrema and tally spatial coordinates (x,y)
    endfor
    calculate global extrema and tally spatial coordinates (x,y)
    change set of skewers according to results
    if (old skewer set = new skewer set) then
       end = true
    endif
endwhile
```

Table 2. Pseudo-code description of a parallel implementation of steps 2-4 in the FPPI algorithm.

In the following section, we describe our design strategy to implement the parallel code shown in Table 2, aimed at enhancing replicability and reusability of slices in the FPGA (e.g., resources in a Virtex FPGA are divided into slices) through the use of systolic array design.

3. SYSTOLIC ARRAY-BASED DESIGN

One of the main advantages of systolic array-based implementations[11] is that they are able to provide a systematic design procedure that allows for the derivation of a element processing structure and an interconnection pattern (even in a very first approximation), which can then be easily ported to real hardware configuration via Handel-C[7] and EDIF[8] descriptions. Through this procedure, we can also calculate the data dependences in a very straightforward manner, along with subsequent shift operations that need to be implemented in the hardware structure, thus improving over available design procedures which are mainly based on trial-and-error practices. The adopted procedure also allows for a more practical and easy-to-handle strategy to implement algorithms in hardware, which does not really require detailed knowledge on hardware description languages. The systematic design used in this work is based on the general idea that an algorithm can be described by a tuple $A = \left(I^N, C, D, X, Y\right)$, where I^N is a set of indexes, C represents the volume of calculations, D denotes the data dependences, and X, Y are the inputs and outputs to the algorithm, respectively. An algorithm denoted by A would be equivalent to another algorithm $A' = \left(I'^N, C', D', X', Y'\right)$ if and only if the following properties are satisfied: i) $X' = X$ and $Y' = Y$; ii) $C' = C$; and iii) A biyective function $T = \begin{bmatrix} \pi \\ S \end{bmatrix}$ exists, such that $D' = T(D)$ and $I'^N = T\left(I^N\right)$. It should be noted that π is defined in the first *M* indexes of A', and provides

information about the corresponding operation that takes place in each node, i.e., $\pi : I^N \rightarrow I'^M$. Similarly, S is defined in the remaining indexes of A', and provides information on the processing cells which are going to execute the operation, i.e., $S : I^N \rightarrow I'^{N-M}$. It should be noted that π must preserve data dependences, i.e., $\pi \cdot d_i > 0$ with $D = (d_1, d_2, \cdots, d_N)$, and S must preserve the biyective property in T. Taking into account the above framework for systolic algorithm design, we only need to introduce minor modifications to the main loop in Table 2 in order to make it suitable for parallel implementation under the proposed systematic methodology. Specifically, the algorithm should be expanded to a 3-dimensional space[11], as shown by Table 3.

```
for i = 1 to Number of Pixels in partition do
   for k=1 to Number of skewers in partition do
      Result(i,k,0)=0
      for j=0 to number of bands do
         Sample(i,0,j)=Pixel(i,j)
         Skewer(0,k,j)=Skewer(k,j)
         Result(i,k,j)=Result(i,k,j-1)+ Skewer(i,k,j)*Sample(i,k,j)
       endfor
    endfor
endfor
```

Table 3. Modification of the main loop in the parallel FPPI algorithm to adapt it to a systolic array-based implementation.

From the code outlined above, we can obtain a set of data dependences and the values for $\pi = [t_{11}, t_{12}, t_{13}]$. Firstly, if $D_{result} = (i,k,j) - (i,k,j-1) = (0,0,1)$, and taking into account that $\pi \cdot d_i > 0$, then $t_{13} > 0$. The same reasoning can be applied to $D_{sample} = (i,k,j) - (i-1,k,j) = (1,0,0)$, thus obtaining $t_{11} > 0$, and to $D_{skewer} = (i,k,j) - (i,k-1,j) = (0,1,0)$, resulting in $t_{12} > 0$. Therefore, we can conclude that $\pi = [1,1,1]$. In order to maintain the biyective condition of T, we set the values of S using combinations of 1's and 0's. At this point, we also need to decide which calculations will be stored in each processing element, and what would be the total volume of information that will be communicated to neighboring processing elements which are directly interconnected throughout the network. We must also take into account which kind of hardware architecture will be our target, along with its advantages and limitations, in particular, in terms of resources available and communication speed between the host and the FPGA board. Therefore, our design should be aimed at maximizing computational power of the hardware and minimizing the cost of communications. This is particularly relevant in our specific application, where more than 200 data values will be handled for each intermediate result, a fact that may introduce problems related with limited resource availability and inefficiencies in hardware replication and reusability. Having the above issues in mind, we have opted for a solution given by $S = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, which results in the following data shifts:

$$S \times D_{result} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \text{ i.e., local results remain static at each processing element,} \quad (2)$$

$$S \times D_{pixels} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \text{ i.e., pixels will be shifted from top to bottom,} \quad (3)$$

$$S \times D_{skewers} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \text{ i.e., skewers will be shifted from left to right,} \quad (4)$$

Before addressing the implementation of the proposed systolic-based implementation in Handel-C, Fig. 1 provides a block diagram of the proposed architecture for the implementation of FPPI, where "s" stands for "skewer" and "p" stands for "pixel." Finally, "VD" denotes the virtual dimensionality concept, which defines the number of elements in the processing matrix as a function of the number of endmembers, *p*. As will be shown in section V, our experimental results reveal that the best performance is achieved when the size of partitions is set to *p*. We have also experimentally tested that an increase in the partition size may lead to under-utilization of processing elements and significantly lower speedups.

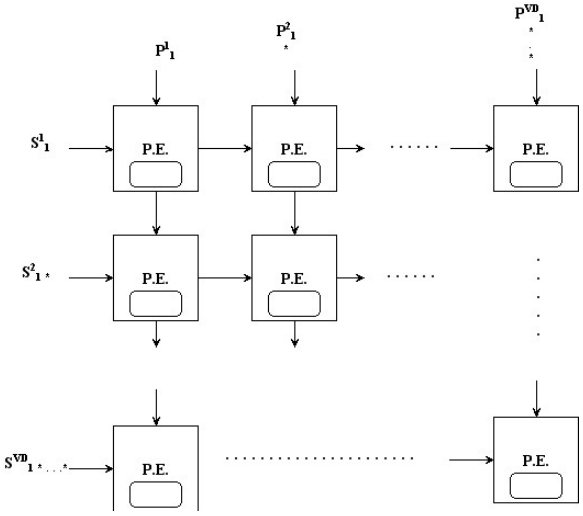


Figure 1. Pseudo-code diagram of the proposed systolic array design for implementation of the FPPI algorithm.

4. HANDEL-C IMPLEMENTATION

The final decision on implementing our design using Handel-C instead of other well-known hardware description languages such as VHDL or Verilog was taken on the account that it is possible to generate hardware versions of hyperspectral analysis algorithms using a high-level-based approach and in relatively short time. Several projects have been developed to bring new methods for describing hardware implementations, most notably, Ocapi-xl[13] or SpecC[14]. However, our experience shows that, for this kind of reconfigurable hardware, it may be better to follow the guidelines imposed by the vendors. With the above remarks in mind, the DK3.1 software package emerges as one of the most interesting alternatives, in particular, taking into account its portability to UNIX-type operating systems. In this regard, the main advantage of Handel-C design language is its simplicity, where the design can be defined and evaluated using a pseudo-C programming style. For illustrative purposes, the source code in Handel-C of the proposed FPPI implementation is shown in Table 4. This code is based on 8 skewers and 8 sample pixel vectors for each partition. For a full understanding of the piece of code shown in Table 4, we point to reference material[7]. The implementation provided in Table 4 was compiled and transformed to an EDIF specification automatically by using the DK3.1 software package. With this specification, and using other tools such as Xilinx ISE[9] to simplify the final steps of the hardware implementation, we also incorporated hardware-specific limitations and constraints to the mapping process into a Virtex-II FPGA. These tools allowed us to evaluate the total amount of resources needed by the whole implementation, along with sub-totals related to different functional units available in the FPGA that will be discussed in the following section, along with our results in terms of algorithm performance and execution time.

5. EXPERIMENTAL RESULTS

In this section, we provide an experimental assessment of the proposed FPGA-based implementation, not only in terms of parallel performance and hardware implementation efficiency, but also in terms of endmember extraction accuracy in the context of a real application based on a well-known hyperspectral data set. Resultingly, our proposed Virtex-II hardware implementation will be compared to the original FPPI algorithm implemented in Matlab, and to the PPI available from Research Systems ENVI software using well-known hyperspectral data.

```
void main(void){
unsigned int 16 max[VD];
unsigned int 16 end[VD];
unsigned int 3 i; //The size is defined by the VD
unsigned int 3 k; //The size is defined by the VD
unsigned int 7 j; //Number of bands

par(i=0;i<8;i++){
       max[i]=0;
}
par(k=0;k<VD;k++){
       par(i=0;i<VD;i++){
              for(j=0;j<NB;j++){
                     Processing_Element[i][k](pixels[i][j],skewers[k][j],0@i,0@k);
              }
       }
}
for(i=0;i<VD;i++){
       max[i]=Processing_Element[i][k](0@max[i],0,0@i,0@k);
}
phase_1_finished=1;
while (!phase2) { //Waiting to enter phase 2 }
for (i=0;i<VD;i++) end[i]=0;
for (i=0;i<VD;i++){
       for(k=0;k<VD;k++) {
              for(j=0;j<NB;j++){
                     end[i]=end[i] &&
                     Processing_Element[i][k](pixels[i][j],skewers[k][j],0,0);
              }
       }
}
phase_2_finished=1;
global_finished=0;
for(i=0;i<VD;i++){
       global_finished= global_finished && end[i];
}
```

Table 4. Source code of our Handel-C implementation of the FPPI algorithm.

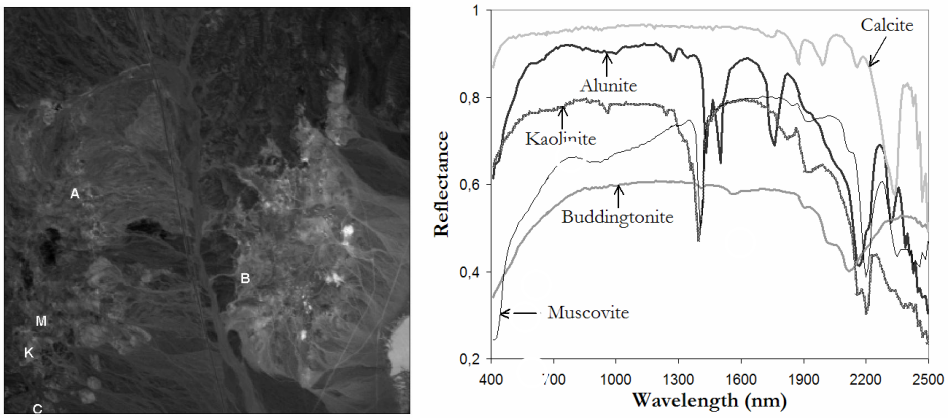
With the purpose of experimenting with a widely available hyperspectral data set, we have selected an image scene collected by the NASA Jet Propulsion Laboratory Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) over the Cuprite mining district in Nevada, USA. Fig. 2(a) shows the spectral band collected by the sensor above at 827 nm wavelength, where several pure pixels made up of five minerals of interest, namely, alunite, buddingtonite, calcite, kaolinite and muscovite are white-circled and labeled by letters A, B, C, K and M. This 224-band scene is well understood mineralogically, and has reliable ground truth in the form of a library of mineral spectra collected at the site by USGS. Fig. 2(b) shows the corresponding USGS mineral spectra. The availability of ground-truth has made this scene a standard test site for comparison of endmember extraction algorithms, as shown in several previous studies[3,6].

To address the issue of endmember extraction accuracy by the different methods, Fig. 3(a) shows the endmember pixels extracted by ENVI's PPI and Fig. 3(b) shows the endmember pixels extracted by our Matlab implementation of FPPI. It should be noted that our hardware-based implementation of FPPI produced exactly the same results than those displayed in Fig. 3(b). After comparing Fig. 3(a) and Fig. 3(b), it is clear that both the PPI and the FPPI produced very similar results, with most extracted pixels overlapped. Regarding computational complexity, Table 5 tabulates the number of iterations and computing time for both FPPI (sofware- and hardware-based implementations) and ENVI's PPI. As we can see from Table 5, the number of iterations carried out by the FPPI was much smaller than that needed for the ENVI's PPI. Quite opposite, the ENVI's PPI required thousands of iterations to achieve results similar to those found by the FPPI in terms of signature purity (the execution times reported on Table 5 for the non-hardware versions were measured in a PC with AMD Athlon 2.6 GHz processor and 512 Mb of RAM). As shown by Table 5, the Matlab-based software

implementation of FPPI algorithm was more than 24 times faster than the ENVI's PPI algorithm in the same computing environment, while the FPGA-based implementation showed a significant increase in performance with regards to the two considered software versions due to the adopted low-level hardware implementation.

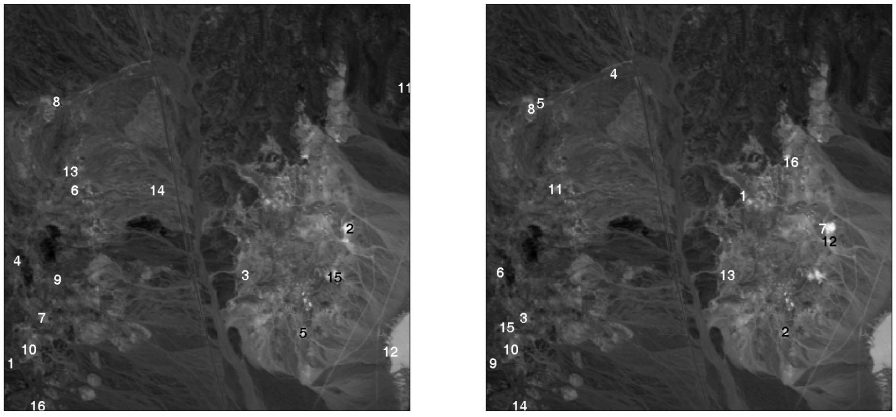
ENVI's PPI		FPPI-software		FPPI-hardware	
Iterations:	Computation time:	Iterations:	Computation time:	Iterations:	Computation time:
10^4	3068	237	124	237	62

Table 5. Number of iterations and computation time in seconds for ENVI's PPI, FPPI-software, and FPPI-hardware algorithms.



(a) Original band (b) USGS mineral spectra

Figure 2. (a) Spectral band at 827 nm of the Cuprite AVIRIS image scene; (b) Spatial positions of pure pixels made up of Alunite (A), Buddingtonite (B), Calcite (C), Kaolinite (K) and Muscovite (M); (c) USGS spectral signatures.



(c) ENVI's PPI (d) FPPI

Figure 3. Endmember pixels extracted by (a) ENVI's PPI algorithm; (b) FPPI algorithm.

It should be noted that the main bottleneck of the hardware version, as it is indeed the case with other typical parallel implementations[15], is mainly due to the cost of communications. In our particular solution, the need to rely on DMA accesses to provide the FPGA with the data and thus exploit its computational power results in the most significant fraction of the time consumed by the algorithm, i.e., more than the 96% of the execution time. Obviously, this issue should be improved in future developments of our method. It should also be noted that our experiments are based on a relatively small scene for demonstration purposes, but the use of aggressive communication protocols based on DMA from host to FPGA may slow down computational performance significantly for higher-volume hyperspectral data sets. In order to address this relevant issue, we need to fit as much information as possible into the FPGA, thus reducing the number of DMA operations and optimizing the use of the processing power available from specialized hardware, which is capable of performing multiple vector dot products in less than just a few nanoseconds.

Implementation	Number of gates	Number of slices	Percentage of total	Maximum operation frequency (MHz)
FPPI_2x2	97443	1185	3%	29.257
FPPI_4x4	212412	3587	10%	21.782
FPPI_8x8	526944	12418	36%	18.032

Table 6. Summary of resource utilization for different FPGA-based implementations of the FPPI algorithm.

Finally, Table 6 shows the resources used for hardware implementations of different sizes, tested on the Virtex-II XC2V6000-6 FPGA of the Celoxica's ADMXRC2 board[12]. This FPGA has a total of 33792 slices available. As we can see from Table 6, there are still opportunities to expand the design, although it should be noted that the slices dedicated to RAM are generally more resource-consuming than those dedicated to other components, so we expect to be able to find an inflexion point in the maximum size of RAM that can be used. This is also one of our main goals for future research. We also anticipate that, once we start increasing the storage needs above this size, the number of slices available in the FPGA would likely decrease drastically, and the speedup gains would also decrease. For illustrative purposes, Table 6 also shows the maximum operation frequency (in MHz) admitted by the different hardware implementations tested. This is an important parameter in order to measure the performance of a circuit because, generally, there is a correlation between the amount of frequency admitted by the circuit and its peak performance.

6. CONCLUSIONS AND FUTURE LINES

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. Current sensor design practices could greatly benefit from the inclusion of data-oriented pre- and post-processing modules, such as FPGAs, which can be easily mounted or embedded in the sensor due to its compact size. Although current airborne and satellite systems could greatly benefit from the integration of fast data processing modules, the true fact is that current systems are planned several years in advance, using information which is mostly based on achievable technical landmarks due to electronics (generally obsolete when the mission is launched), and do not incorporate any information feedback from data analysis and processing experience, which is also a major shortcoming. In this paper, we propose a first step towards the incorporation of hardware-based (FPGA) components for data processing (not only data acquisition) on board remote sensing platforms. The algorithm selected for demonstration has been the Pixel Purity Index (PPI), one of the most well-known approaches for hyperspectral data analysis in the remote sensing community. Our experimental results demonstrate that a hardware version of the PPI algorithm can significantly outperform both the original implementation of the algorithm, available in commercial software, and a recent algorithm optimization exclusively based on software considerations. The reconfigurability of FPGA systems[16] opens many innovative perspectives from an application point of view, ranging from the appealing possibility of being able to adaptively select the data processing algorithm to be applied on board, out of a pool of available algorithms, from a control station on Earth immediately after the data is collected by the sensor, to the possibility of providing a response in real-time in remote sensing applications that certainly demand so, such as wildland fire monitoring and tracking, oil spill detection, etc. Although the experimental results presented in this paper are very encouraging, further work is still needed to arrive to an optimal hardware design and implementation for the PPI and other hyperspectral analysis algorithms.

REFERENCES

1. R.A. Schwogengerdt, *Remote Sensing: Models and Methods for Image Processing*, 2nd. Ed., Academic Press, 1997, p. 447.
2. C.-I Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification*, Kluwer Academic/Plenum Publishers, 2003.
3. J. W. Boardman, F. A. Kruse and R. O. Green, "Mapping target signatures via partial unmixing of AVIRIS data," *Summaries of JPL Airborne Earth Science Workshop*, Pasadena, CA, 1995.
4. Research Systems, Inc., *ENVI User's Guide*. Boulder, CO: Research Systems, Inc., 2001.
5. 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.

6. C.-I Chang and A. Plaza, "A fast iterative algorithm for implementation of pixel purity index," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 4, Jan. 2006 (to appear).
7. Celoxica Ltd., *Handel-C Language Reference Manual*, 2003.
8. Celoxica Ltd., *DK Design Suite User Manual*, 2003.
9. Xilinx Inc., Available online: http://www.xilinx.com
10. A. Plaza, D. Valencia, J. Plaza and P. Martínez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, accepted for publication (to appear).
11. M. Valero-García, Juan J. Navarro, José M. Llabería, Mateo Valero and Tomás Lang, "A method for implementation of one-dimensional systolic algorithms with data contraflow using pipelined functional units," *Journal of VLSI Signal Processing*, vol. 4, pp. 7-25, 1992.
12. Celoxica Ltd., Available online: http://www.celoxica.com
13. Ocapi-xl project, Available online: http://www.imec.be/design/ocapi
14. SpecC project, Available online: http://www.specc.org
15. D. Valencia, A. Plaza, P. Martinez and J. Plaza, "On the Use of Cluster Computing Architectures for Implementation of Hyperspectral Analysis Algorithms," *Proceedings of the 10th IEEE Symposium on Computers and Communications*, pp. 995-1000, Cartagena, Spain, 2005.
16. M. A. Vega-Rodríguez, J. M. Sánchez and J. A. Gómez, "Guest Editor's Introduction – Special Issue on FPGAs: Applications and Designs," *Microprocessors and Microsystems*, vol. 28, pp. 193-196, 2004.