AN FPGA-BASED IMPLEMENTATION OF A HYPERSPECTRAL ANOMALY DETECTION ALGORITHM FOR REAL-TIME APPLICATIONS

1.María Díaz, 3. Raúl Guerra, 5.Sebastián López

Institute for Applied Microelectronics (IUMA)
University of Las Palmas de Gran Canaria
35001 Las Palmas de Gran Canaria, Spain

2. Julián Caba, 4. Jesús Barba

School of Computer Science
University of Castilla-La Mancha
13071 Ciudad Real, Spain

ABSTRACT

Remote sensing has gained relevance in the last years, mainly due to the emergence of UAVs carrying airborne imagery sensors. In this regard, the on-board data processing for on-the-fly making-decision applications is also gaining momentum. Nevertheless, these flight vehicles are still limited in terms of power budget and computational capacity, which hampers the handling of the hyperspectral data. Consequently, there is an emerging trend towards the development of more hardware-friendly algorithms suitable for an efficient implementation in parallel computing devices. In this sense, the LbL-FAD algorithm arose in response to the lack of causal anomaly detectors that could be easily integrated in push-broom-based acquisition systems. In this work, we have analysed the feasibility of the performance and power needs of the LbL-FAD algorithm in a mid-range re-configurable FPGA-SoC such as the XC7Z020 chip. Concretely, a highly optimized FPGA accelerator of the LbL-FAD method has been described for the line-by-line detection of anomalous spectra.

***Index Terms*—** FPGAs, hyperspectral imaging, anomaly detection, line-by-line performance, real-time, High-Level Synthesis, low-power.

1. INTRODUCTION

Although the hyperspectral technology has been around for quite some time, it has received increasing attention in the last years from the scientific and industry sectors. Its expansion and growing recognition have been largely propelled by the emergence of compact-size aerial platforms, such as Unmanned Aerial Vehicles (UAVs) and, new space-borne missions. Nevertheless, these aerial vehicles are still limited in terms of computational capacities, power budget and data storage.

Consequently, on-Earth processing has been the mainstream solution for the hyperspectral data handling. In this regard, images sensed by airborne imagery sensors are traditionally downloaded to the ground segment for being off-line processed on supercomputing systems. Nonetheless, the ever-growing acquisition data rates of the latest-generation sensors and the bottleneck that represents the high data volume transmission could jeopardize the real-time response of time-sensitive applications. Accordingly, the on-board processing has become established as a potential solution for these scenarios.

Against this backdrop, it is needed in the literature new algorithmic approaches that take into account the constraints imposed by devices with limitations, as to the available resources and power budget, from the earliest stage of development. Additionally, the causality imposed by real-time applications based on push-broom/whisk-broom scanners must be also met through the definition of non-global algorithms capable of independently processing blocks of image pixels. In turn, this prevents the storing and management of large data volumes, thereby reducing the computing resources and speeding up the execution process.

In the field of anomaly detection, the Line-by-Line Fast Anomaly Detector for Hyperspectral Imagery (LbL-FAD) [1] emerged to resolve the aforementioned requirements, paving the way for real-time detection performance. The LbL-FAD algorithm is a subspace-based anomaly detector that employs an orthogonal projection strategy for estimating the orthogonal subspace spanned by the background distribution where anomalous entities are better detectable. For doing so, blocks of image pixels are independently processed ruling out any spatial alignment restriction. In this work, we have further analyzed the performance of the FPGAs for the real-time discrimination of rare spectra employing the LbL-FAD algorithm in a high data-rate scenario characterized by a push-broom scanner mounted in a UAV. In particular, we have implemented the LbL-FAD in a Xilinx Zynq-7000 programmable System on Chip (SoC). This SoC can be found in low-cost, low-weight and compact-size development boards, which makes it an interesting low-cost alternative for UAVs at the expense of less computing resources than other costly commercial products. However, as it is shown in Section 4, these are not limiting factors to achieve high-performance results due to the exploitation

MINECO of the Spanish Government (PLATINO project, no. TEC2017-86722-C4-1-R, subprojects 1 and 4), the ACIISI of the "Gobierno de Canarias" and the European Social Fund (FSE) (POC 2014-2020, Eje 3 Tema Prioritario 74 (85%)) and the Regional Government of Castilla-La Mancha (SymbIoT project, no. SBPLY-17-180501-000334).

of parallelism at different levels.

2. THE LBL-FAD ALGORITHM

The LbL-FAD algorithm addresses the anomaly detection issue in four main stages:

1. Line-by-line extraction of the most representative background spectra:

The LbL-FAD algorithm firstly estimates the orthogonal subspace that better depicts the background distribution. For doing so, the p most representative pixels within the first n_f sensed hyperspectral frames, $\mathbf{E^*} = [\mathbf{E}_1, ..., \mathbf{E}_{\mathrm{n_f}}]$, are extracted using an unmixing-like strategy.

2. Estimation of the background distribution:

Nonetheless, **E*** comprises several alike spectra since the hyperspectral frames are processed independently with the methodology followed by the LbL-FAD algorithm. Consequently, the most dissimilar pixels within **E*** are selected in this stage to obtain the subspace spanned by the background.

3. Line-by-line anomaly detection:

Once the background is modelled, the anomaly detection is performed on the new received hyperspectral frames. To this end, the orthogonal projection to the subspace spanned by the background distribution is computed for each image pixel. This is because an anomalous pixel is supposed to be poorly represented by the background pattern. Therefore, an abnormal spectrum should have a high enough projection onto the subspace not covered by the background, which makes easily the detection of abnormal spectra.

4. Line-by-line binary map generation:

Unlike other state-of-the-art anomaly detectors, the LbL-FAD outputs a line-by-line binary map where anomalous pixels are segmented from the background.

For computing the whole detection process, the LbL-FAD algorithm is based on a set of core operations that performs the well-known Gram-Schmidt orthogonalization method. They are collected in the pseudo code displayed in Algorithm 1 and independently applied to a block of BS hyperspectral pixels, $\mathbf{M}_k$. These operations, in the way that they are described in Algorithm 1, are performed in the Stage 1 and Stage 2 for the estimation of the background distribution.

1- **Average pixel calculation**. Firstly, the average pixel, $\hat{\boldsymbol{\mu}}$, of the input image block, $\mathbf{M}_k$, is calculated (Line 1).

2- **Centralization**. Then, $\mathbf{M}_k$ is centralized subtracting $\hat{\boldsymbol{\mu}}$ from each image pixel, obtaining the centralized version of the input image, **C** (Line 2).

Afterwards, the p most representative pixels within $\mathbf{M}_k$ are extracted from Lines 3 to 18 of Algorithm 1. For doing so, the following operations are repeated until the stop condition shown in Line 8 of Algorithm 1 is fulfilled.

3- **Brightness Calculation**. The selected pixels are those with the highest dot product with itself in each iteration, also referred as to brightness of a pixel (Lines 4 to 6). In this sense, $\mathbf{e}_n$ represents the selected pixel within the original image, $\mathbf{M}_k$

Algorithm 1 Set of core operations

Inputs: $\mathbf{M}_k = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{BS}]$, α
Outputs: $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p]$ {Characteristic pixels}; $\hat{\boldsymbol{\mu}}$ {Average Pixel}; $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_p]$ {Orthogonal vectors}; $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_p]$ {Orthonormal vectors}; τ {Threshold}
Algorithm:
1: Centroid or average pixel: $\hat{\boldsymbol{\mu}}$;
2: Centralization: $\mathbf{C} = \mathbf{M}_k - \hat{\boldsymbol{\mu}}$;
3: **while** exit $= 0$ **do**
4: **for** $j = 1$ **to** BS **do**
5: Brightness: $\mathbf{b}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$;
6: **end for**
7: Maximum Brightness: $j_{max} = \mathrm{argmax}(\mathbf{b}_j)$;
8: **if** $\frac{b_{j\max}}{(\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})' \cdot (\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})} \cdot 100 < \alpha$ **then**
9: Stop Condition: exit = 1
10: **else**
11: Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{j_{max}}$;
12: $\mathbf{q}_n = \mathbf{c}_{j_{max}}$;
13: $\mathbf{u}_n = \mathbf{q}_n / b_{j_{max}}$;
14: Projection: $\mathbf{v}_n = \mathbf{u}'_n \cdot \mathbf{C}$;
15: Subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
16: $\tau = b_{j_{\max}}$
17: **end if**
18: **end while**

(Line 11), $\mathbf{q}_n$ is its counterpart in **C** (Line 12) and $\mathbf{u}_n$ is the normalized version of $\mathbf{q}_n$ (Line 13).

4- **Projection**. Then, all vectors within **C** are projected onto $\mathbf{u}_n$, obtaining the projection vector, $\mathbf{v}_n$ (Line 14).

5- **Subtraction**. To finish, the spectral information within **C** that cannot be represented by the selected pixel in the actual iteration, and that in consequence is orthogonal to it, is retained in **C** for the next iteration (Line 15). For this reason, pixels within **Q** and **U** are orthogonal with each other.

Algorithm 2 The LbL-FAD algorithm.

Inputs: $\mathbf{HI} = [\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_{\frac{nr \cdot nc}{BS}}]$, n_f, α
Outputs: $\mathbf{AD} = [\mathbf{x}_{11}, \mathbf{x}_{12}, ..., \mathbf{x}_{kj}]$
Algorithm:
Stage 1:
1: **for** k $= 1$ **to** n_f **do**
2: $\mathbf{E}_\mathrm{k} =$ **Algorithm 1**$(\mathbf{M}_\mathrm{k}, \alpha)$
3: $\mathbf{E^*} = [\mathbf{E^*}, \mathbf{E}_\mathrm{k}]$;
4: **end for**
Stage 2:
5: $[\hat{\boldsymbol{\mu}}_b, \mathbf{Q}, \mathbf{U}, \tau] =$ **Algorithm 1**$(\mathbf{E^*}, \alpha)$
Stage 3: Applied to each new received frame, $\mathbf{M}_k$, $k > n_f$
6: **for** j $= 1$ **to** BS **do**
7: Centralization: $\mathbf{c}_j = \mathbf{r}_j - \hat{\mu}_b$
8: **for** n $= 1$ **to** p **do**
9: Projection: $v = \mathbf{U}'_n \cdot \mathbf{c}_j$
10: Subtraction: $\mathbf{c}_j = \mathbf{c}_j - \mathbf{Q}_n \cdot v$
11: **end for**
12: Brightness AD: $\mathbf{d}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$
13: **Stage 4:**
14: **if** $\mathbf{d}_j \leq 1.5 \cdot \tau$ **then** $(\mathbf{x}_{kj} = 0)$
15: **else**$(\mathbf{x}_{kj} = 1)$
16: **end if**
17: **end for**

The aforementioned set of core operations is employed for extracting the most representative pixels within each image block, $\mathbf{M}_k$ and consequently, for generating the matrix **E*** in Stage 1, as it can be seen from Lines 1 to 4 in Algorithm 2. Then, **E*** feeds this set of core operations to obtain the background subspace comprised of orthogonal vectors, **Q** and **U**, in the Stage 2 (see Line 5 of Algorithm 2). The definition of

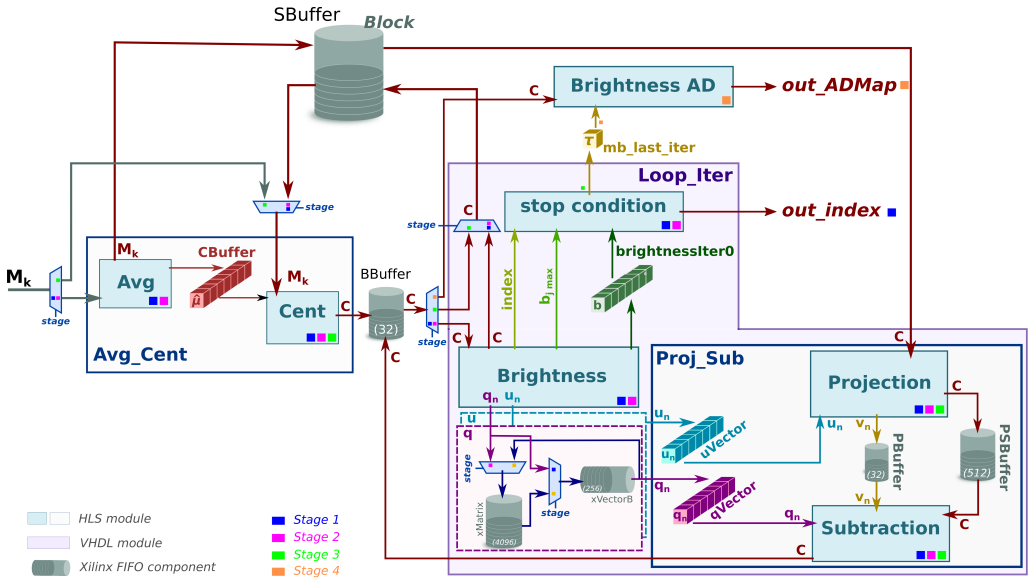

Fig. 1. Overview of the LbL-FAD hardware accelerator

these orthogonal subspace prevents to implicitly compute the orthogonal projection matrix, $\mathbf{P} = \mathbf{I} - \mathbf{W}(\mathbf{W}'\mathbf{W})^{-1}\mathbf{W}'$, for the subsequent anomaly detection, which is a very computationally demanding task. Additionally, $\mathbf{C}$ retains the spectral information lost if the background is reconstructed using the p reference vectors selected in Stage 2. On this basis, the remaining maximum brightness in $\mathbf{C}$, $\tau = b_{j_{\max}}$ (see Line 16 of Algorithm 1), could be potentially used as a benchmark to identify anomalous pixels in Stage 4.

For the detection and discrimination of the anomalous pixels performed in Stages 3 and 4, the aforementioned set of core operations is also used but executed in a different order. In this regard, the projection separation statistical, $\mathbf{d}_j$ (Line 12 of Algorithm 2), in overall calculates the brightness of the orthogonal component of each image pixel, $\mathbf{c}_j$, to the subspace spanned by the background distribution and represented by $\mathbf{Q}$ and $\mathbf{U}$. Accordingly, the Gram-Schmidt orthogonalization process (Projection and Subtraction operations) may be used for calculating the remaining spectral information of each image pixel, $\mathbf{d}_j$, which is in fact orthogonal to the space spanned by the background samples (Lines 8 to 12 of Algorithm 2). Finally, if $\mathbf{d}_j$ is higher than 1.5 times τ, this pixel is marked as an anomaly (Lines 14 to 16 of Algorithm 2).

3. FPGA IMPLEMENTATION OF THE LBL-FAD

The implementation of the FPGA-based hardware accelerator of the LbL-FAD algorithm has been carried out by using a combination of HLS (High-Level Synthesis) generated modules and custom glue logic in VHDL. HLS technology [2] has been used to synthesize the RTL (Register Transfer Language) code corresponding to the components that instantiate the functionality of the before mentioned core operators (see blocks and sub-blocks with white and light blue background in Figure 1). RTL models are the entry point to the implementation tools that are in charge of the generation of the *bitstream*, the programming file that configures the FPGA fabric to behave as it is described by the RTL. However, RTL models are low-level, time-consuming to write and verify, which leads to error-prone and lengthy development cycles. Thus, HLS tools are key to rise productivity of such a kind of developments since they are able to automatically generate RTL models out of a specification of the functionality using C programming language. Verification time is also sped up by a co-simulation process, where the RTL code is exercised with a set of stimuli generated by the test bench function written in C. HLS frameworks provide the developer with the necessary mechanisms to generate the bridging infrastructure (usually using a system-level modeling language such as SystemC) that fosters the reutilization of the higher-level testing environment, avoiding the need to rewrite it with the consequent saving in time and effort.

Once the set of core operators (plus the *Stop Condition*) modules are obtained, the orchestration of all of them is still pending. The proposed architecture of the accelerator aims to make the most of the inherent parallelism of the LbL-FAD algorithm. For such end, the different modules should be able to work concurrently in a perfectly synchronized dataflow. Despite the fact that the HLS tools are able to capture dataflow and other concurrent data processing semantics, there are limitations regarding the actual architectures they are able to generate. For example, it is not possible to model in HLS an iterative dataflow model where the outputs of one iteration feed the input of the next one. Instead, a classic pipeline architecture is inferred with inter-loop data dependencies that is more inefficient.

To overcome this issue, tailor-made VHDL modules (see

blocks with light purple background in Figure 1) have been used to implement an optimized dataflow. The VHDL logic is responsible for connecting the inputs and outputs of the HLS-synthesized blocks by means of a network of selectors and buffers (i.e. FIFO and BRAM components that are generated using vendor-specific tools) that is governed by a scheduler. The scheduler is implemented as a synchronous FSM (Finite State Machine) that selectively activates/deactivates HLS blocks and enables/disables data paths depending on the processing stage in which the algorithm is. In Figure 1, the different modules are marked with colored square/s indicating the stage or stages these resources are operative. The same color code is used to tag the data sources in the selectors (trapeziums in Figure 1) for the different steps of the algorithm. This strategy not only promotes the optimization of the available hardware resources for the targeted application (i.e. FIFOs are used as circular buffers to avoid instantiating larger memory modules) but it also permits that one memory buffer can be shared by more than one producer and consumer, such as in the case of the *SBuffer*, which is not possible to be modeled with current HLS tools. Additionally, each stage of the LbL-FAD algorithm depends on the previous one, so they must share information thorough internal buffers, i.e. Stage 2 extracts the orthogonal vectors **Q** and **U** that are stored in qMatrix and uMatrix, respectively (*xMatrix* in Figure 1), then Stage 3 uses such vectors to obtain the binary map of anomalies.

Also, it is worth mentioning that the architecture is configurable so it can be determined the degree of parallelism within the HLS modules, concerning the number of bands that can be processed in one clock cycle, namely PEs (Processing Elements). The higher the number of PEs , the higher the performance obtained, as it is disclosed in Section 4.

4. EXPERIMENTAL RESULTS

In order to evaluate the LbL-FAD hardware accelerator, the proposed architecture has been implemented using the Vivado Design suite from Xilinx. Specifically, the prototype has targeted the XC7Z020-CLG484 version of the Xilinx Zynq-7000 SoC. This FPGA has been selected because of its low-cost, low-weight and high flexibility, features that make it an interesting device to be integrated in aerial platforms, such as drones.

Figure 2 graphically shows the performance and power consumption obtained by the FPGA-based implementation for several versions of the accelerator, where both the number of PEs and the clock frequency have been varied. The input hyperspectral images are 825 lines height, each line comprising 1024 hyperspectral pixels with 12-bits depth and 160 bands each. The employed data set was collected by an aerial platform consisting of a Specim FX10 pushbroom hyperspectral camera mounted on a DJI Matrice 600 drone during some flight campaigns. The first n_f = 100 hyperspectral lines are used to calculate the background spectra and distribution in Stage 1 [1] (lines 1-4 of Algorithm 2) and the following lines

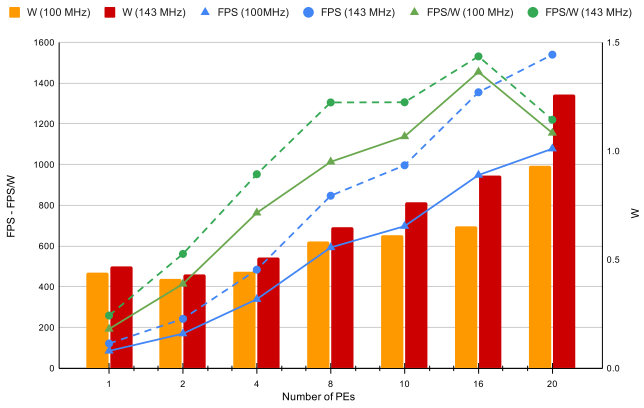

Fig. 2. Performance and power trade-off of different versions of the LbL-FAD accelerator

(725) are processed to search for anomalies.

Performance results (left vertical axis, blue series), measured as the number of frames composed of 1024 hyperspectral pixels each that can be processed per second (FPS), show a linear behaviour, indicating that the architecture scales accordingly with the increment of the number of PEs. On top of that, it is maintained the relation of $\approx$ +40% performance increase between the slowest (100MHz) and the fastest (143MHz) versions of the hardware.

The effects of the variation in the clock frequency of the circuit have been also studied in relation with the impact on power consumption. In general, the implementation is highly efficient from the point of view of the required energy budget (right vertical axis, orange and red bars), ranging from less than 0.5 watts (PE=1) to little more than 1 and 1.3 watts (PE=20, 100MHz and 143MHz versions respectively).

The power efficiency in terms of FPS per watt (left vertical axis, green series) is a figure of merit of great importance for applications that use remote sensing platforms powered by batteries. In this sense, it is critical to maximize the flight time of the drone. It can be observed that the implementation at the highest clock speed consumes a similar amount of energy than the slower one, whilst delivering a significant more performance. Therefore, the relation FPS/W is clearly favorable for the hardware running at 143MHz.

5. REFERENCES

[1] M. Díaz, R. Guerra, P. Horstrand, S. López, and R. Sarmiento, "A line-by-line fast anomaly detector for hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 8968–8982, 2019.

[2] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, April 2011.