Hyperspectral Image Compression Tecniques for Space Applications

Aday García, Roberto Sarmiento, José F. López

Integrated System Design Division, IUMA
Universidad de Las Palmas de Gran Canaria (ULPGC)
Las Palmas de Gran Canaria, Spain
E-mail: agarcia@iuma.ulpgc.es, roberto@iuma.ulpgc.es, lopez@iuma.ulpgc.es

***Abstract* — This paper starts describing the existing techniques used to compress hyperspectral images. Contributions are then addressed on the hardware implementation of the Lossy Compression for ExoMars (LCE) algorithm. Synthesis results of its implementation on a Virtex-5 FPGA family are also shown in this work. These results are further compared with the FPGA implementation of the ESA lossless compression algorithm.**

Keywords: hyperspectral image, lossy compression, low complexity, error-resilience, hardware implementation

I. INTRODUCCTION

Hyperspectral images are usually captured and stored onboard a satellite or aircraft to be transmitted afterwards to a ground station. Present and future high resolution instruments for space remote sensing missions make it necessary for the onboard payload to handle an extensive amount of image data. The existing limitations in the available bandwidths and onboard storage often require to apply image compression to reduce the data volume prior to transmission to the ground segment.

Typically, image compression techniques have been classified into two categories: lossless and lossy methods. Lossless compression has been traditionally desired to preserve all the information contained in the image. However, the compression ratios which can be achieved with such techniques are limited [1]. On the contrary, lossy compression yields higher compression ratios at the cost of introducing losses in the information [2]. Despite the loss of quality in the reconstructed image, these last techniques have become very popular, especially when the required compression is greater than what it can be achieved with lossless techniques.

When hyperspectral images are acquired by a satellite, the compression algorithms have to be implemented in a piece of hardware able to operate onboard. Although coding efficiency is certainly a key aspect for onboard compression, the encoder complexity, error-resilience or hardware friendliness are requirements to be taken into account.

In order to be able to operate in real time, the encoder complexity has to be limited as the computational power available onboard for compression is generally limited. The low complexity requirement generally rules out the application of transform coding methods. Therefore, the prediction plus entropy coding paradigm is amenable to low complexity.

Algorithms should be capable of containing the effect of bit-flipping or packet losses in the compressed file. These errors typically occur because of the noise on the communication channel. The data partition into units that are coded independently is an approximation to alleviate this problem.

Since onboard compression algorithms are usually implemented on FPGA or ASIC, the algorithm has to be designed using simple techniques. The algorithm should buffer a limited amount of data, e.g. not the whole hyperspectral cube, but a few spectral lines. Furthermore, in order to speed up the compression process the algorithm should be able to be parallelized.

The rest of this paper is organized as follows. Section II describes the LCE algorithm [3]. Section III exposes the proposed implementation approach. Section IV presents the most significant results obtained and, finally, Section V outlines the conclusions extracted from this work.

II. LOSSY COMPRESSION FOR EXOMARS (LCE) ALGORITHM

The LCE algorithm [3] is designed to operate onboard a satellite and to achieve high compression ratios for the ESA-ExoMars mission. The main objectives in its design are the low complexity, error-resilient and easy to implement on hardware. The algorithm consists of a predictor followed by a Golomb entropy coder. It compresses individual blocks of data independently, so that an error in one block does not affect the decompression of the rest of the blocks.

A. Prediction

The algorithm compresses independent non-overlapping spatial blocks of 16x16 pixels with all bands. Let $x_{m,n,i}$ be the pixel of a hyperspectral image in the m-th line, n-th pixel and i-th band. For the first band $(i = 0)$, 2D compression is performed using a predictor defined as $\tilde{x}_{m,n,0} = (\hat{x}_{m-1,n,0} + \hat{x}_{m.n-1,0}) \gg 1$, where $\tilde{x}$ denotes the predictor, $\tilde{x}$ the decoded value and $\gg$ stands for right shift. All predictor values, except for the first sample, are mapped to non-negative values. For all other bands, the samples are predicted from the decoded samples $x_{m,n,i-1}$ in the previous band. A least-square estimator is computed as $\propto = \propto_N / \propto_D$. Quantized versions of $\propto$, denoted $\hat{\propto}$, are generated using a scalar quantizer. Finally, the predicted values are computed for all samples in a block as

$\tilde{x}_{m,n,i} = \hat{m}_i + \hat{\alpha}\,(\hat{x}_{m,n,i-1} - m_{i-1})$ and the prediction error as $e_{m,n,i} = x_{m,n,i} - \tilde{x}_{m,n,i}$.

B. Rate-Distorsion Optimization

If the prediction is close to the actual pixel value that it makes sense to skip the encoding of the prediction error samples, a one-bit-flag is raised indicating that the current block contains all-zero prediction samples (this is denoted as *zero_block* condition).

C. Quantization and Mapping

Prediction error samples are quantized to integer values, $eq_{m,n,i}$, and dequantized to reconstructed values, $er_{m,n,i}$. It is possible to choose between a uniform scalar quantizer and a uniform-threshold quantizer. The reconstructed values are mapped to non-negative values.

D. Entropy Coding

The mapped prediction residuals of a block are encoded in raster order using a Golomb code which parameter is constrained to a power of two.

III. Hardware Implementation of the LCE Algorithm

The LCE algorithm was originally implemented in ANSI C programming language to be executed on a single threaded CPU. Based on the changes described in [4], the algorithm has been implemented by using the Catapult C Synthesis and ModelSim tools.

A. Split of the algorithm into functionality independent modules

The first step was to split the provided reference code into modules that being functionally independent perform the same global compression result. The outputs of this process produced eight modules implemented in ANSI C files independently. Special attention was made to the resulting interfaces because they will be latter the interface ports at hardware level.

B. Module implementation by using Catapult C

Once the modules were split, its implementation was performed by using the Catapult C Synthesis tool. At this step, the design methodology proposed by the tool was followed, setting global hardware constraints, such as, the clock frequency, to 100MHz, and the reset/enable behavior. The technology was set to the Xilinx family Virtex-5 model 5VFX100TFF1136, speed grade -1. Also, the handshake protocol was configured to use start and stop signals. The architectural constraints such as loop unrolling and initiation interval were studied and set for each module in particular. The output of this step was a synthesizable RTL code used for the implementation of the whole algorithm.

C. Hardware controller implementation

The final step carried out was the design, implementation and verification of a hardware controller module. This module manages each module and its data flow. Additionally, it controls the external interfaces. Fig. 1 shows the black-box model of the hardware controller module designed.

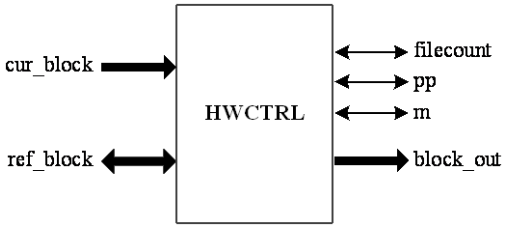

Figure 1. Hardware controller (hwctrl) black-box model

IV. Results

This section presents the synthesis results obtained. The synthesis tool employed was Precision RTL from Mentor Graphics. The target platform to perform the synthesis was the Virtex-5 FPGA model 5VFX100TFF1136 with speed grade -1. Table I depicts the synthesis results obtained after the place and route of the whole hardware controller module. Table II presents the comparison between the LCE algorithm implementation and the ESA lossless implementation [4].

TABLE I. Independent Module Synthesis Results

Module	LUT	Slices	RAM Blocks	DSP48	Max. Frequency
hwctrl	7746 (12,1%)	1937 (12,11%)	4 (0,88%)	25 (9,77%)	86,964MHz

TABLE II. LCE vs. ESA Lossless resource comparison

	LCE Algorithm		ESA Lossless [4]
	4VLX200 (-11)	*5VFX100 (-1)*	*4VLX200 (-11)*
LUT	9283	7746	10306
Slices	4642	1937	6312
RAM Blocks	4	4	9
DSP48	25	25	9
Max. Frequency	75,844 MHz	86,964 MHz	81 MHz

V. Conclusions

As it is seen from Table II, the implementation results obtained for the LCE algorithm are comparable to the ESA lossless implementation. The main difference is the number of DSP used, which is greater for the lossy algorithm because of the mathematical operations involved. The results also show that the number of RAM required is really low for the LCE algorithm.

The LCE algorithm has been implemented into an FPGA by using Catapult C tool. Design, implementation and verification of a hardware controller module which integrates the Catapult C implementation is also presented. The proposed implementation achieves synthesis results comparable to the ESA lossless implementation despite of its greater complexity.

References

[1] R.E. Roger and M.C. Cavenor, "Lossless compression of AVIRIS images," IEEE Transactions on Image Processing, vol. 5, no. 5, pp. 713–719, May 1996.

[2] B. Penna, T. Tillo, E. Magli, and G. Olmo, "Transform coding techniques for lossy hyperspectral data compression," IEEE Transactions on Geoscience and Remote Sensing, vol. 45, no. 5, pp. 1408–1421, May 2007.

[3] A. Abrardo, M. Barni, and E. Magli, "Low-complexity predictive lossy compression of hyperespectral and ultraspectral images", ICASSP, 2011 IEEE International Conference, pp. 797-800, May 2011.

[4] Lossless Data Compression, Green Book, CCSDS 120.0-G-1, May 1997

[5] L. Santos, José F. López and R.Sarmiento, LCE GPU Implementation Report. ESA Standard Document.