Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 2, February 2014, pg.302 – 307

RESEARCH ARTICLE

Analysis of Lossy Hyperspectral Image Compression Techniques

Dr. S.M.Ramesh (Assistant Professor), P.Bharat (P.G. Scholar), J.Anand (P.G.Scholar), J.Anbu Selvan (P.G.Scholar)
Department of Electronics and Communication Engineering,
Bannari Amman Institute of Technology, Sathyamangalam-638401, India
Email ID: bharat.p.9019@gmail.com, mass.anand14@gmail.com, anbuselvanjs@gmail.com

Abstract—Graphics Processing Units (GPU) are becoming a widespread tool for general-purpose scientific computing, and are attracting interest for future on board satellite image processing payloads due to their ability to perform massively parallel computations. This paper describes the GPU implementation of an algorithm for on board loss hyper spectral image compression and proposes an architecture that allows accelerating the compression task by parallelizing it on the GPU. The selected algorithm was amenable to parallel computation owing to its block-based operation, and has been optimized here to facilitate GPU implementation incurring a negligible overhead with respect to the original single-threaded version. In particular, a parallelization strategy has been designed for both the compressor, which is implemented on a GPU using MATLAB. Experimental results on several hyper spectral images with different spatial and spectral dimensions are presented, showing significant speed-ups with respect to a single-threaded CPU implementation. These results highlight the significant benefits of GPUs for on board image processing, and particularly image compression, demonstrating the potential of GPUs as a future hardware platform for very high data rate instruments.

Index Terms—Graphics processing unit (GPU), lossy hyper spectral, image compression, MATLAB.

I. INTRODUCTION

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

Typically, image compression techniques have been classified into two categories of 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 lossless techniques are limited. On the other hand, lossy compression yields higher compression ratios at the cost of introducing losses of information. Despite the loss of quality in the reconstructed image, these kinds of techniques have become very popular, especially when the required compression is greater than what can be achieved with lossless techniques. Moreover, studies like [1] have assessed the effect of the losses on specific applications of hyper spectral images, i.e., target detection or classification, showing that high compression ratios can be achieved with little impact in performance. The importance of lossy satellite image compression is witnessed by the fact that the Consultative Committee for Space Data System (CCSDS) has developed and approved relevant international standards, including a standard for lossless multi- and hyper spectral image compression [2], and is currently

working towards the definition of a lossy one. When hyper spectral images are acquired by a satellite, the compression algorithms have to be implemented in hardware able to operate on board the satellite. Although coding efficiency is a key aspect for on board compression, there are other very important requirements that should also be taken into account. The computational power available on a satellite is limited, not even close to that of a processor for workstation applications; hence there is a strong interest towards the development of low complexity approaches for hyper spectral image compression.

Several methods have been proposed for lossy hyper spectral image compression, some of them being generalizations of existing 2D image or video compression algorithms. A few algorithms were designed using JPEG 2000 [3] with a spectral decor relator. For example, in [4], [5] a Karhunen-Loeve transform was used to compress hyper spectral cubes. Discrete wavelet transform and Tucker decomposition were applied in [6], while a pairwise orthogonal spectral transform was developed in [7]. In [8] the H.264/AVC standard for video compression was applied to hyper spectral cubes. All these techniques achieve very satisfactory performance in rate-distortion sense, and many of them were proven to have little or no negative impact on image analysis, e.g., classification and target detection, even at relatively low bit-rates. On the other hand, these methods are rather complex for on board application, typically requiring significant computational and memory resources. Recently, a new low-complexity paradigm was proposed, which we refer to as "predictive lossy compression"[9]. This technique is based on a prediction stage, followed by quantization, rate-distortion optimization and entropy coding. It leverages the simplicity and high-performance of prediction-based compression, requiring very few operations and memory, while advanced quantization and rate-distortion optimization ensure state-of-the-art compression performance. Unlike near-lossless compression, which is known to have poor performance at low bit-rates, predictive lossy compression has competitive performance at all bit-rates, and is particularly good at high bit-rates, which are of great interest in multi- and Hyper spectral image compression.

Besides the development of hyper spectral image compression algorithms with suitable complexity and memory requirements, it is also necessary to consider the hardware where the algorithm is going to be implemented and executed. Normally, the on board compression algorithm is implemented on a field programmable gate array (FPGA), an application-specific integrated circuit (ASIC) or a digital signal processor (DSP).Among the possible different hardware platforms, Graphics Processing Units (GPUs) represent a very attractive opportunity. Although they became popular due to the video games industry, GPUs have undergone a tremendous development in recent years, evolving to allow their usage for general purpose computation. They offer the possibility to dramatically increase the computation speed in applications that are data-parallel, which is the case of certain types of image processing and compression tasks.

Commercial GPUs are not currently suited to operate on board a satellite, mainly because they are not radiation hardened and due to their high power consumption. However, they have recently attracted a lot of interest in the remote sensing community because of their ability to significantly speed up several processing tasks. Low-power GPUs are already powering graphical tasks in several smart phones. In the future, if there is evidence of their usefulness for on board image processing, it is possible that space-qualified GPUs will be manufactured, bringing the potential of multi-core computation to the satellite imaging scenario. For these reasons, several authors have already started to consider GPU implementations of on board processing tasks, investigating the performance improvements obtained when using GPUs as opposed to single-threaded computation. In [10] it was shown that significant speed-up can be obtained a parallel software architecture is designed, with limitations regarding the size of the dedicated memory of the GPU. GPUs were also utilized for target detection and classification, yielding quick and reliable implementations and showing the necessity of solving memory storage issues to achieve a larger speed-up factor [11],[12]. Significant speed-up factors are also obtained for compression and decompression schemes for satellite data, as shown in [13] and [14]. Regarding on board compression, a GPU implementation of a lossless compression algorithm for hyper spectral images was developed in [15], and [16] showing promising results. In this paper we consider the algorithm in [9], which was proposed for the ESA-Exomars mission and is a hence referred to as "lossy compression for Exomars" (LCE). The motivation behind using this algorithm is twofold. First, LCE is a predictive lossy compression algorithm, and hence it is very simple and capable of achieving high throughput even in a single-threaded implementation. Second, it is inherently a data- and task-parallel algorithm, and hence amenable to a very efficient GPU implementation with very large potential speed-up. In particular, this paper proposes a GPU architecture and implementation of LCE using the NVidia CUDA parallel computing environment [17], and evaluates the performance improvement with respect to a CPU-only implementation. The GPU implementation allows assessing the performance gain, demonstrating the benefits of GPUs for remote sensing applications. Moreover, it identifies the potential difficulties and bottlenecks encountered when trying to accelerate algorithms for hyper spectral image processing, if space-qualified GPUs become a reality in the future. The advantage of using a GPU at the ground station lies in the reduced size and power consumption with respect to a cluster of workstations.

The remainder of this paper is organized as follows. Section II presents the LCE algorithm and Section III introduces GPU programming with MATLAB. Sections III explain the parallelization strategy that was followed to implement the compressor on a GPU. The experimental results for the compressor presented in Section IV. Finally, Section V draws some conclusions.

II. LOSSY COMPRESSION FOR EXOMARS (LCE) ALGORITHM

The LCE algorithm [9] was designed to meet several requirements so that it could operate on board a satellite and to achieve high compression ratios. The main objectives were to make it low complexity, error-resilient and easy to implement on hardware. The algorithm consists of a predictor followed by a Golomb entropy coder with power-of-two parameter.

In order to make the LCE algorithm error resilient, it was designed in such a way that it compresses individual blocks of data independently and, as a consequence, an error in one block does not affect the decompression of other blocks. Besides, it was designed so that that its hardware implementation is simple and easy to parallelize in order to speed up the process for high data-rate sensors.

The different stages of the algorithm are illustrated in Fig. 1and explained next.

A. Prediction

The algorithm compresses independent non-overlapping spatial blocks of size 16×16 with all bands in the hyper spectral cube, which are processed separately. Let $x_{m,n,i}$be the pixel ofa hyperspectral image sample in the *m*-th line,*n*-th column and *i*-th band. For the first band, 2D compression is performed without any information of other bands (INTRA-mode) using a predictor defined as

$$\bar{x}_{m,n,0} = (\hat{x}_{m-1,n,0} + \hat{x}_{m,n-1,0}) \gg 1 \qquad (1)$$

Where $\bar{x}$ denotes the predictor, $\hat{x}$ the decoded value and≫stands for right shift. All predictor values, except for the first sample are mapped to nonnegative values. For all other bands, the samples $x_{m,n,i}$ are predicted from the decoded samples $\hat{x}_{m,n,i-1}$ in the co-located block in the previous band.

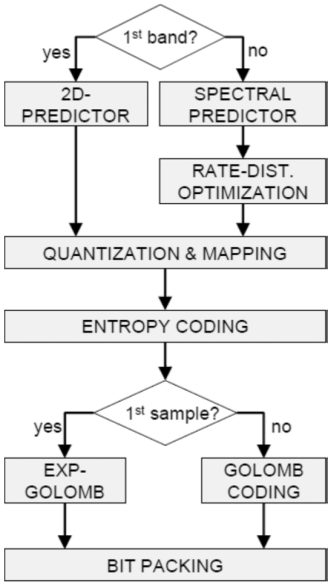


Fig. 1 Flowchart of the LCE algorithm

A least-square estimator (LMS) is computed over the block as:

$$\alpha = \alpha_N / \alpha_D \qquad (2)$$

With $\alpha_N = \sum_{m,n}[(\hat{x}_{m,n,i-1} - m_{i-1})(x_{m,n,i} - m_i)]$ and $\alpha_D = \sum_{m,n}[(\hat{x}_{m,n,i-1} - m_{i-1})(x_{m,n,i-1} - m_{i-1})]$, where m_i and m_{i-1} and *i-1*. Quantized versions of α and m_i , denoted $\hat{\alpha}$and$\hat{m}_i$, are generated using a scalar quantizer. Finally, the predicted values are computed for all samples in a block as:

$$\bar{x}_{m,n,i} = \hat{m}_i + \hat{\alpha}\left(\hat{x}_{m,n,i-1} - m_{i-1}\right) \quad (3)$$

and the prediction error is calculated as:

$$e_{m,n,i} = x_{m,n,i} - \hat{x}_{m,n,i} \quad (4)$$

B. Rate-Distortion Optimization

Before proceeding with the quantization of the prediction error samples, the algorithm checks if the prediction is so close to the actual pixel values that it makes sense to skip the encoding of the prediction error samples, but rather rise a one-bit-flag indicating that the current block contains all-zero prediction error samples (this is denoted as *zero block* condition). To make this decision, the energy of the predictor error is computed:

$$D_0 = \frac{1}{256}\sum_{m,n} e_{m,n,i}^2 \quad (5)$$

If $D_0 < D_T$, with D_T a chosen threshold, then the *zero block* condition is triggered.

C. Quantization and Mapping

Prediction error samples are quantized to integer values $eq_{m,n,i}$, and dequantized to reconstructed values $er_{m,n,i}$. Forthe first band, this process is performed pixel by pixel usinga scalar uniform quantizer. For the other bands, it is possible to choose between a scalar uniform quantize and the uniform threshold quantize (UTQ) presented in [18]. Finally, the reconstructed values are mapped to non-negative values.

D. Entropy Coding

The 16 ×16 residuals of a block are encoded in raster order using a Golomb [19] code which parameter is constrained to a power of two, except for the first sample of each block, which is encoded using an exponential Golomb code of order zero. The quantized first sample of the first band is not encoded, and saved to the compressed file with 16 bits.

E. File Format

The compressed file is a concatenation of coded blocks which are read spatially in raster order, and each block is coded with all bands. The following information is written in the compressed file for all blocks: 1) for bands other than the first one, parameters α and m; 2) one bit to signal the *zero block* condition; 3) for the non-skipped blocks, the encoded quantized prediction error samples of the block are written in raster order.

III. PARALLELIZATION OF THE LCE COMPRESSORWITH MATLAB

The LCE algorithm software was implemented in C language to be executed on a single threaded CPU. The encoder operates in such a way that the hyper spectral image is divided into non-overlapping16 ×16 spatial blocks. The main function consists of a nested loop, which processes the blocks in raster order, and every block is processed with all spectral channels before the compression of the next block starts.

Each block in the image is identified by two spatial coordinatesb_v and b_0 , which indicate the location of the block in the vertical and horizontal spatial directions respectively. In the following, denotes a 16×16 pixels block in band. The innermost loop of the main nested loop covers all bands in the image. In every iteration of this loop, the stages of the LCE compressor, namely prediction, rate-distortion optimization, quantization and mapping are performed on every sample of block. Therefore, multiple loops are needed inside the inner loop which cover all bands so that all the pixels in a16×16 blocks can be processed.

The MATLAB implementation of the compressor consists of different kernels, instead of a single one which performs all stages, since the parallelization that can be achieved for the different stages is different. The basic idea is to design three different kernels, one to perform the prediction, rate-distortion optimization, quantization and mapping; a different one for the entropy coding stage and finally another one to perform the bit packing.

Our goal when designing the MATLAB implementation of the compressor was to make it general for any hyper spectral image, regardless its spatial or spectral dimension, achieving high performance at the same time. However, in the future and depending on the final application, the parallelization strategy presented can be easily adapted to be optimal for the compression of a hyperspectral cube with a specific size, which is known beforehand.

1. ***Entropy Coding***

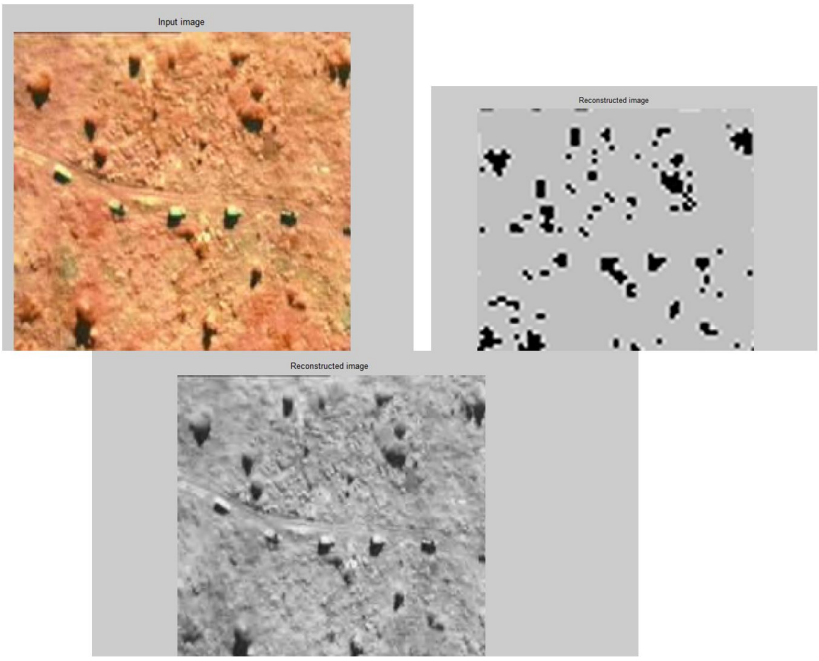
The entropy coding of a 16×16 spatial block in a specific band can be performed without any information of other bands. This makes it possible to process more data in parallel than the previous stages.

To optimize the performance of the MATLAB implementation of the entropy coder, it is decided that the kernel launches the maximum possible number of threads, which in the case of the TESLA C2075 is 1024. This means, that every MATLAB block does the entropy coding of four (1024/256) blocks of the hyper spectral image, each block containing 16×16 samples. The number of MATLAB blocks which are launched by the entropy coder can be calculated for the hyper spectral image under compression as:

Mat lab blocks = (lines×columns×bands)/(1024)

For some hyper spectral images, the previous equation can give a result which is greater than 65535, which is the maximum allowed number of MATLAB blocks. In these cases, the kernel has to be called more than once, what can have a negative impact in the performance of the MATLAB application. Making each MATLAB block have the maximum possible number of threads is also a way to minimize the chances of having to call the entropy coding kernel repeatedly.

The strategy followed for parallelizing the entropy coding is based on pre-processing the Golomb parameter of all prediction error samples in a 16×16 blocks and afterwards computing the code words of every prediction error sample.



Conclusion

A GPU implementation of a lossy compressor and decompress or for hyper spectral images is presented as well as the parallelization strategy followed, utilizing MATLAB parallel architecture. Experimental results are obtained for multispectral, hyper spectral and ultra-spectral images, showing high performance and speedup of up to 15.41 for the GPU compressor when compared to its CPU counterpart.

The performance of the GPU compressor shows better results than that of the GPU decompress or, mainly because the parallelization of the decompress or is limited by the format of the compressed file, which makes it impossible to identify the independent elements that can be processed in parallel. In this exploration, this issue is solved by introducing additional information in the compressed file, specifically in the form of a header.

However, other options should be studied in the future, in order to optimize the trade-off between the amount of parallelization obtained when adding information to the compressed file and the resulting decrease of the compression ratio. It is also shown that the spatial and spectral dimensions of the images must be taken into account when designing a parallelization strategy.

Nowadays it is not possible to employ GPUs for onboard satellite image compression, due to their lack of tolerance to sun radiation and high power consumption. However, this exploration demonstrates that high performance and very flexible implementations can be obtained when GPUs are used for satellite data compression, and help to justify further research in this field, making efforts towards the future development of space qualified GPUs.

REFERENCES

[1] M. Harris and M. Garland, "Optimizing parallel prefix operations for the Fermi architecture," in *GPU Computing Gems Jade Edition*. Boston, MA, USA: Morgan Kaufmann, 2012, pp. 29–38.

[2] A. Plaza, M. Ciznicki, and K. Kurowski, "GPU implementation of jpeg2000 for hyperspectral image compression," in *SPIE 8183*, 2011.

[3] D. Keymeulen, N. Aranki, B. Hopson, A. Kiely, M. Klimesh, and K.Benkrid, "GPU lossless hyperspectral data compression system for space applications," in *Proc. 2012 IEEE Aerospace Conf.*, Mar. 2012, pp. 1–9.

[4] B. Hopson, K. Benkrid, D. Keymeulen, and N. Aranki, "Real-time CCSDS lossless adaptive hyper spectral image compression on parallel GPGPU & multicore processor systems," in *Proc. 2012 NASA/ESA Conf. Adaptive Hardware and Systems (AHS)*, Jun. 2012, pp. 107–114.

[5] S.-C. Wei and B. Huang, "GPU acceleration of predictive partitioned vector quantization for ultra spectral sounder data compression," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp.677–682, Sep. 2011.

[6] C. Song, Y. Li, and B. Huang, "A GPU-accelerated wavelet decompression system with spiht and Reed-Solomon decoding for satellite images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 683–690, Sep. 2011.

[7] D. B. Heras, F. Arguello, J. L. Gomez, J. A. Becerra, and R. J.Duro, "Towards real-time hyper spectral image processing, a GP-GPU implementation of target identification," in *Proc. 2011 IEEE 6th Int.Conf. Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, Sep. 2011, vol. 1, pp. 316–321.

[8] L. Santos, S. Lopez, G. M. Callico, J. F. Lopez, and R. Sarmiento, "Performance evaluation of the h.264/avc video coding standard for lossy hyper spectral image compression," *IEEE J. Sel. Topics Appl. Earth Observ.Remote Sens.*, vol. 5, no. 2, pp. 451–461, Apr. 2012.

[9] I. Blanes and J. Serra-Sagrista, "Pairwise orthogonal transform for spectral image coding," *IEEE Trans. Geosci. Remote Sens.* vol. 49, no. 3, pp. 961–972, 2011.

[10] B. Penna, T. Tillo , E. Magli, and G. Olmo, "Transform coding techniques for lossy hyperspectral data compression," *IEEE Trans. Geosci. Remote Sens.*, vol. 45, no. 5, pp. 1408–1421, May 2007.

