1.5GBIT/S 4.9W HYPERSPECTRAL IMAGE ENCODERS ON A LOW-POWER PARALLEL HETEROGENEOUS PROCESSING PLATFORM

Oscar Ferraz, Vitor Silva, and Gabriel Falcao

Instituto de Telecomunicações
Dept. of Electrical and Computer Engineering, University of Coimbra, Portugal
(E-mail: {oscar.ferraz, vitor, gff}@co.it.pt)

ABSTRACT

This work explores the utilization of low-power heterogeneous devices for parallelizing the compute-intensive hyperspectral and multispectral image compression CCSDS-123 entropy encoders. Multithread processing allows for the near-optimal system's bandwidth to be exploited increasing the system overall performance. The experimental platform consists of a low-power Jetson TX2 GPU equipped with an ARM Cortex-A57 and Denver 2 host processors, reporting more than 1552 Mb/s and, more importantly, 315 Mb/s/W, all running under a global 5 W power budget, which makes it a good candidate for onboard image compression.

Index Terms— Low Power Graphics Processing Units, Parallel Programming, Multispectral Image Compression, Hyperspectral Image Compression, Lossless Compression

1. INTRODUCTION

The consultative committee for space data systems (CCSDS)-123 [1] is a hyperspectral and multispectral lossless image compression standard composed of a 3D predictor and an entropy encoder. Usually, the systems that generate this type of images (satellites, drones, etc...) impose severe energy restrictions. Hence, field-programmable gate arrays (FPGAs) are suitable candidates to implement the CCSDS-123 due to low-power requirements and energy consumption. The smartphone market has turned CPUs and graphics processing units (GPUs) into energy efficient devices, making them potential competitors against FPGAs.

In this work we exploit a low-power GPU board (Jetson TX2) to parallelize the CCSDS-123 sample-adaptive entropy encoder and block-adaptive entropy encoder. In the encoding procedure, which contains data dependencies, hybrid parallelizations (CPU + GPU) are proposed, thus producing a solution based on heterogeneous computing. The entropy encoder represents approximately 90% of the processing time using only intra-band prediction ($P = 0$).

This work was supported by Instituto de Telecomunicações and Fundação para a Ciência e a Tecnologia, under Projects UIDB/EEA/50008/2020 and PTDC/EEI-HAC/30485/2017.

The implementations are subject to tests that compare the parallel execution times with the serial counterparts in order to identify the best solutions. The energetic analysis is also performed, measuring the power used by the board over the algorithm's execution time. In the end, the throughput rate and energy efficiency are compared with the current state-of-the-art [2, 3, 4, 5, 6, 7].

2. MULTISPECTRAL AND HYPERSPECTRAL IMAGE COMPRESSION

Multispectral images typically incorporate dozens of bands while hyperspectral images usually consist of several hundred [8]. Multispectral and hyperspectral images (MHIs) have 3 dimensions [9, 10] with a size of $N_x \times N_y \times N_z$, being N_x the number of rows, N_y the number of columns and N_z designating the number of bands.

The CCSDS proposed a solution for the lossless compression of MHIs [1, 11]. The CCSDS 123 is composed of two main parts: a predictor and an entropy encoder. The first part uses an adaptive 3D prediction model which calculates the difference between the observed value and the predicted value, outputting mapped prediction residuals (MPRs) ($\delta_z(t)$) with low entropy. The predicted values are calculated by its neighboring samples and by P neighboring bands. Those MPRs, can be encoded by 2 types of entropy coders. The sample-adaptive entropy coder uses Golomb-Power-of 2 coding to encode the MPRs [11]. This entropy encoder encodes MPRs independently for each spectral band in the t domain ($t = y \cdot N_x + x$). Each MPR is encoded using a variable-length binary codeword, which is adaptively selected based on statistics that are updated after each sample is encoded [11].

The block-adaptive entropy encoder follows the CCSDS 121, a 1D universal lossless encoder that uses Rice codes [12]. The residuals are divided in blocks containing 8, 16, 32 or 64 residuals. Four algorithms are executed concurrently for each block resulting in 4 different codewords: i) The sample splitting option splits off the k least significant bits from each sample and the most significant bits (MSBs) are encoded as alternative unary coding. This method iterates k from 0 to

13 in order to find the value which yields the best compression. ii) The second extension option encodes pairs of residuals, encoding γ ($\gamma = (\delta_i + \delta_{i+1})(\delta_i + \delta_{i+1} + 1)/2 + \delta_{i+1}$) using alternative unary coding. iii) The zero-block option encodes the number of consecutive all-zero blocks. iv) The no-compression option is selected when none of the above option provides any compression to the block. From these four methods, the one which yields a better compression, is selected and the compressed block is inserted in the bitstream.

Table 1 shows the compression ratios for both encoders in various images with different numbers of bands used in the prediction (P). As shown in [5], increasing P, decreases throughput performance due to significant increment of the predictor's execution time. In order to reduce that time increment and as a result of, in most images, increasing P does not bring any significant compression benefits, the solutions developed will be made with $P = 0$.

Table 1. Compression rates for the sample adaptive entropy encoder and the block adaptive entropy encoder. For $P > 3$, the variation on the compression ratios are less than 0.2. Therefore, they are not represented in the following table. The compression parameters are described in annex C from [11]

	Bands used in prediciton (P)	AVIRIS Hawaii	AVIRIS Yellowstone	CRISM frt00010f86	CRISM frt00009326	CASI
Sample adaptive entropy encoder	0	3.27	1.37	1.45	1.55	1.64
	1	5.26	1.61	1.52	1.66	1.88
	2	5.27	1.61	1.52	1.66	1.90
	3	5.56	1.61	1.52	1.66	1.90
	⋮	⋮	⋮	⋮	⋮	⋮
	15	5.77	1.60	1.53	1.66	1.90
Block adaptive entropy encoder	0	3.24	1.37	1.45	1.54	N/A
	1	5.18	1.60	1.51	1.66	N/A
	2	5.38	1.61	1.52	1.66	N/A
	3	5.46	1.61	1.52	1.66	N/A
	⋮	⋮	⋮	⋮	⋮	⋮
	15	5.66	1.59	1.52	1.66	N/A

3. ENCODERS PARALLELIZATION

Jetson TX2 has 2 different central processing units (CPUs): the Denver 2 CPU developed by Nvidia with 2 cores and an ARM Cortex-A57 CPU with 4 cores, enabling to run dedicated threads simultaneously in six different cores.

The encoders output variable-length codewords, making the task of parallelizing the encoder difficult. However, it is possible to extract some parallelization from the bitstream generation. The rule of thumb is to distribute the bitstream generation between the available compute units and, ultimately, concatenate the various bitstreams into one. The main goal is that every core should take approximately the same time to process a variable amount of data and achieve the best throughput performance.

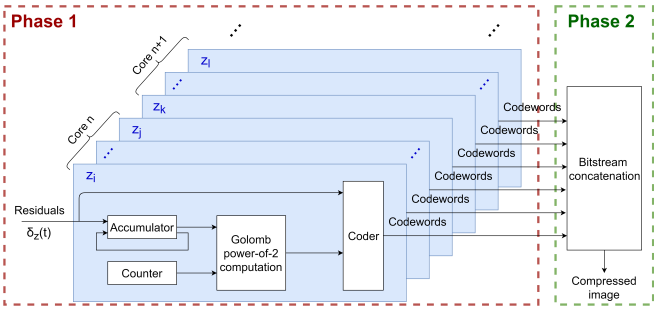

Fig. 1. Sample adaptive encoder parallelized in the CPU. Each core encodes a determined number of bands. Phase 1 is comprised of the codeword generation in each core. Phase 2 is the codewords concatenation and starts after Phase 1 finishes.

The sample adaptive encoder encodes each band separately, achieving band-level parallelism. To parallelize this encoder, each core encodes a specified number of bands as shown in Fig. 1 and lastly, one core merges the resulting bitstreams into one.

In the block adaptive encoder, different algorithms are applied to the blocks of MPRs. The sample splitting is executed in the GPU since it takes longer to execute. The remaining methods are executed in the Denver 2 CPU to ensure the lowest processing time. Once each method outputs the option selected for each block, the workload is distributed over the available 6 cores in order to generate the bitstream. As in the sample adaptive encoder, when all the cores finish generating the correspondent bitstream, the bitstreams are merged by one of the cores. Fig. 2 shows the scheme for the parallel block adaptive encoder.

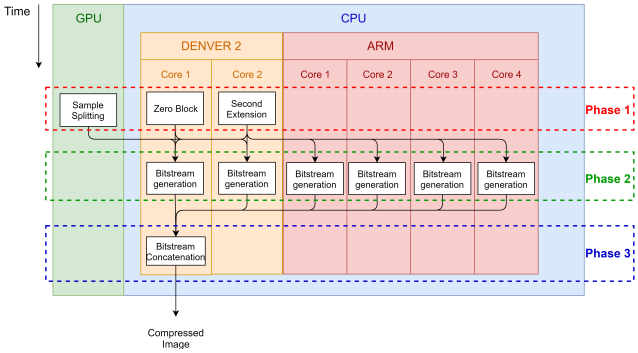

Fig. 2. Diagram of the parallelized CCSDS 123 block adaptive encoder. Phase 1 is composed by the zero block and second extension methods, which runs on the denver 2 CPU, and by the sample splitting, executing in the GPU. After Phase 1 finishes, Phase 2 generates bitstreams in each core of the 2 available CPUs. When Phase 2 terminates, in Phase 3, the bitstreams are concatenated into one.

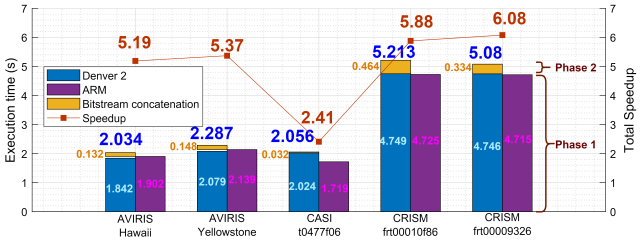

Fig. 3. Detailed execution times for the CCSDS 123 sample adaptive encoder running on the CPU for hyperspectral images. Phase 1 is composed by the ARM and Denver 2 bars. Phase 2 is the bitstream concatenation. The bands executed in each core are: AVIRIS Hawaii - D=48, ARM=32; AVIRIS Yellowstone - D=48, ARM=32; CASI - D=4, ARM=16; CRISM frt00010f86 - D=120, ARM=76; CRISM frt00009326 - D=120, ARM=76;

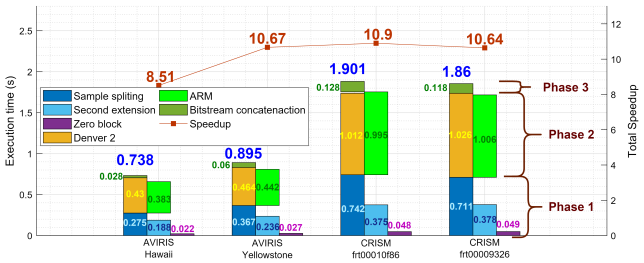

Fig. 4. Detailed execution times for the CCSDS 123 block adaptive encoder for hyperspectral images. Phase 1 comprises the sample splitting, second extension and zero block. Phase 2, represented by the ARM and Denver 2 bar. Phase 3 concatenates the bitstream into one.

4. EXPERIMENTAL RESULTS

This work is based on a serial implementation of CCSDS 123 developed by European space agency (ESA) [13]. All the speedups and the parallelizations are calculated from this code. The serial version is compiled by gcc for the native architecture (-march=native) and with level 2 optimizations (-O2). The tests are executed using the command *taskset –cpu-list* and the command *nice -n*. The execution times were measured using *clock_getime* from time.h C library. All the images tested were downloaded from [14]. The serial times were attained by executing the functions in one Denver 2 CPU core. The tests were executed with the recommended compression parameters from annex C from [11] with $P = 0$ and $B = 1$. Fig. 3 depicts the results for the parallel sample adaptive encoder. This figure is divided in 2 phases: the ARM and Denver 2 in phase one, running concurrently, and the second phase composed of the bitstream concatenation. All speedups are higher than 5, except for the CASI image. The loads in this image are harder to balance between the CPU cores due to the low number of bands, reducing parallelism. Processing multispectral images with less than 6 bands, results in an inefficient implementation that suffers from vacated cores. If the multispectral image has more than 6 bands, the loads are extremely difficult to balance.

The parallel version of the block adaptive encode follows the diagram of Fig. 2. In the first phase, the blocks are pre-processed to determine the shortest codeword using the 3 methods. From the 3 algorithms executed concurrently, the most expensive is sample splitting. This method represents more than 50% of the total processing of the serial encoder. In order to exploit the GPU architecture to obtain the best performance, this method is executed in the GPU with one thread per block, achieving faster execution times compared to executing in the CPU and achieving speedups superior to the sample adaptive entropy encoder. Fortunately, this algorithm is easily parallelizable and runs on the GPU.

Faster times are achieved when compared to executing in the CPU. For these methods, the Denver 2 CPU achieves better performances compared to the ARM CPU and, for that reason, the remaining methods are executed, each one, on a Denver 2 core.

Then, in the second phase, the bitstream generation is divided between the cores using threads. Each core encodes the block that has the shortest codeword calculated in the previous phase. Empirical results show that, in order to balance the loads, the MPR blocks are divided roughly by 30% for each Denver 2 CPU core and 10% for each ARM CPU core. After the bitstream generation, one Denver 2 core concatenates the bitstreams from the previous phase into a single one.

Fig. 4 shows the results for the block adaptive entropy encoder. A good balance was obtained in the CPU cores with the difference being approximately 100 ms (10%). Overall speedups obtained range from 8.5 to 10.9. In multispectral images a good balance is harder to achieve between the CPU cores.

Table 2. Results for implementations of the CCSDS 123 encoders in various platforms, adapted from [3]. * Higher is better. Our solution uses the block-adaptive entropy encoder, encoding the AVIRIS Hawaii image, for $P = 0$.

Platform	Language	D (bit)	Throughput (Mb/s)	Power (W)	Efficiency* (Mb/s/W)
V-5 SX50T[2]	VHDL	13	577.72	**0.70**	825.31
V-7 XC7VX690T[3]	VHDL	16	**3701.44**	5.30	698.38
V-4 XC2VFX60 [3]	VHDL	16	2062.24	0.95[2]	**2170.78**
GT 610[3]	OpenCL	16	1091.2	<29.00[1]	37.63
2x GTX 560M[4]	CUDA	12	4707.48	<150.00[1]	31.38
GTX 750ti[5]	CUDA	12	**5299.80**	<60.00[1]	88.33
Jetson TX1[6]	CUDA	8	1548.89	<10.00[1]	154.89
i7-2760QM[4]	OpenMP	12	1688.16	<45.00[1]	37.51
2x Xeon X5690[7]	OpenMP	12	240.60	<260.00[1]	0.93
Our solution	CUDA	16	1552.32	**4.93**	**314.87**

[1]This denotes the TDP given by the manufacturer

[2]This value was obtained by the Designer SmartPower tool from Microsemi in [15]

5. RELATION TO PRIOR WORK

Prior works [15, 3, 16, 4, 5, 6, 7] do not present individual results for the predictor and encoder alone. From our experimental results, the predictor takes approximately 10% of the processing time for $P = 0$ on GPU. In order to compare our results with the state-of-the-art, we apply the same logic, assuming that 90% of the processing time is for the encoding process, showing the CCSDS 123 results in Table 2.

Space grade FPGAs provide flexible and high performance solutions at low-power with tolerance to space radiation (Virtex 5QV [3, 15, 17]). In [15], it is proposed a low complexity architecture with low hardware occupancy, which was achieved by identifying parameters that affect the performance. Although low energy consumptions were obtained, compression time increased compared to [18, 19].

Real-time compression was achieved in [3, 2, 17, 16, 20] but in [3] older FPGA models suffer from lack of memory. In [17], the authors accomplished high throughput performance without using external memory but incurring in higher energy consumption.

In [21], the authors propose a heterogeneous architecture using an FPGA and a GPU in the same system. The GPU takes advantage of the high throughput performance to compute the calculations, while the FPGA uses the flexible logic for formatting and interfacing the data between the system storage and the GPU.

Using Nvidia's GPU in [5], the authors concluded that of all parameters, 3 had the most impact on execution time and compression ability, the number of prediction bands, the prediction neighborhood (column-oriented or neighbor-oriented) and prediction mode (full mode or reduced mode). It was found that as the number of bands used in the prediction increases, throughput performance decreases, but compression ratio increases. Consequently, there is a trade-off between compression ratio and throughput performance.

In [4], by using a general purpose graphics processing unit (GPGPU) approach to explore spectral and spatial parallelism of the algorithm, the execution times were superior to the ones developed using open multi-processing (OpenMP). The authors exploited the high data reuse and the high speed of GPU memory to obtain better performance compared to the OpenMP version.

Writing open computing language (OpenCL) code on GPUs is 6 times faster than developing very high speed integrated circuits hardware description language (VHDL) on FPGAs [3].

6. CONCLUSIONS

The use of low-power heterogeneous processors equipped with energy-efficient GPUs brings a new paradigm to the field of onboard multispectral and hyperspectral compression, even though, not as the efficiency as FPGAs. By exploring all available compute units, it is possible to attain speedups of 10, taking less than 1 second to encode hundreds of MBs of data. Overall, for the best of our knowledge, the proposed system surpasses the state-of-the-art literature, in terms of energy-efficiency and power requirements. It should be kept in mind that the adopted heterogeneous architecture is supported by embedded ARM-based cores and a low-power GPU with only 256 CUDA cores (where a desktop GPU such as the ones used in [4, 5] has more cores) and is capable of achieving 314.87 Mb/s/W under a 5 Watt power budget.

However, FPGAs still dominate the state-of-the-art. In fact, FPGAs require hardware design knowledge, while the current approach is more suitable for the signal processing community.

7. REFERENCES

[1] Consultative Committee for Space Data Systems, "CCSDS 123.0-B-1 Lossless Multispectral & Hyperspectral Image Compression ," Internet: https://public.ccsds.org/Pubs/123x0b1ec1s.pdf, May, 2012, [Apr. 11, 2019].

[2] D. Keymeulen, N. Aranki, A. Bakhshi, Huy Luong, C. Sarture, and D. Dolman, "Airborne demonstration of fpga implementation of fast lossless hyperspectral data compression system," in *2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, July 2014, pp. 278–284.

[3] D. Báscones, C. González, and D. Mozos, "Parallel Implementation of the CCSDS 1.2.3 Standard for Hyperspectral Lossless Compression," *Journal of Remote Sensing*, vol. 7, 09 2017.

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

[5] R. L. Davidson and C. P. Bridges, "GPU accelerated multispectral EO imagery optimised CCSDS-123 lossless compression implementation," in *2017 IEEE Aerospace Conference*, March 2017, pp. 1–12.

[6] R. L. Davidson and C. P. Bridges, "Error Resilient GPU Accelerated Image Processing for Space Applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 1990–2003, Sep. 2018.

[7] D. Keymeulen, N. Aranki, B. Hopson, A. Kiely, M. Klimesh, and K. Benkrid, "Gpu lossless hyperspectral data compression system for space applications," in *2012 IEEE Aerospace Conference*, March 2012, pp. 1–9.

[8] P. Ghamisi, J. Plaza, Y. Chen, J. Li, and A. J. Plaza, "Advanced spectral classifiers for hyperspectral images: A review," *IEEE Geoscience and Remote Sensing Magazine*, vol. 5, no. 1, pp. 8–32, March 2017.

[9] E. A. Bernal and Q. Li, "Hybrid vectorial and tensorial compressive sensing for hyperspectral imaging," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2015, pp. 2454–2458.

[10] A. W. Bitar, L. Cheong, and J. Ovarlez, "Target and background separation in hyperspectral imagery for automatic target detection," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, April 2018, pp. 1598–1602.

[11] Consultative Committee for Space Data Systems, "CCSDS 123.2-G-1 Lossless Multispectral & Hyperspectral Image Compression ," Internet: https://public.ccsds.org/Pubs/120x2g1.pdf, Dec, 2015, [Apr. 24, 2019].

[12] Consultative Committee for Space Data Systems, "Ccsds 121.0-b-2 lossless data compression," Internet: https://public.ccsds.org/Pubs/121x0b2ec1.pdf, May, 2012, [May. 7, 2019].

[13] European Space Agency, "European Space Agency Public License – v2.0," Internet: https://amstel.estec.esa.int/tecedm/misc/ESA_OSS_ license.html, [Aug. 21, 2019].

[14] Consultative Committee for Space Data Systems, "123.0-B-Info TestData," Internet: https://cwe.ccsds.org/sls/docs/Forms/Allitems, [Aug. 21, 2019].

[15] L. Santos, L. Berrojo, J. Moreno, J. F. López, and R. Sarmiento, "Multispectral and Hyperspectral Lossless Compressor for Space Applications (HyLoC): A Low-Complexity FPGA Implementation of the CCSDS 123 Standard," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 9, no. 2, pp. 757–770, Feb 2016.

[16] J. Fjeldtvedt, M. Orlandić, and T. A. Johansen, "An efficient real-time fpga implementation of the ccsds-123 compression standard for hyperspectral images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 10, pp. 3841–3852, Oct 2018.

[17] A. Tsigkanos, N. Kranitis, G. A. Theodorou, and A. Paschalis, "A 3.3 gbps ccsds 123.0-b-1 multispectral & hyperspectral image compression hardware accelerator on a space-grade sram fpga," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.

[18] M Klimesh, "Low-Complexity Lossless Compression of Hyperspectral Imagery via Adaptive Filtering," *Jet Propulsion Laboratory (JPL), California, USA, The Interplanetary Network Progress Report*, vol. 42-163, pp. 1–10, 2015.

[19] A. Abrardo, M. Barni, and E. Magli, "Low-complexity predictive lossy compression of hyperspectral and ultraspectral images," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2011, pp. 797–800.

[20] L. M. V. Pereira, D. A. Santos, C. A. Zeferino, and D. R. Melo, "A low-cost hardware accelerator for ccsds 123 predictor in fpga," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2019, pp. 1–5.

[21] R. L. Davidson and C. P. Bridges, "Adaptive multispectral gpu accelerated architecture for earth observation satellites," in *2016 IEEE International Conference on Imaging Systems and Techniques (IST)*, Oct 2016, pp. 117–122.