The final publication is available at ACM via http://dx.doi.org/10.1145/3577949.3577969

Space Compression Algorithms Acceleration on Embedded Multi-Core and GPU Platforms

Alvaro Jover-Alvarez*, Ivan Rodriguez*, Leonidas Kosmidis

Barcelona Supercomputing Center (BSC) and Universitat Politecnica de Catalunya (UPC); email: {ajover, irodrig, lkosmidi}@bsc.es

David Steenari

European Space Agency (ESA); email: David.Steenari@esa.int

Abstract

Future space missions will require increased on-board computing power to process and compress massive amounts of data. Consequently, embedded multi-core and GPU platforms are considered, which have been shown beneficial for data processing. However, the acceleration of data compression - an inherently sequential task - has not been explored. In this on-going research paper, we parallelize two space compression standards on both CPUs and GPUs using two candidate embedded GPU platforms for space showing that despite the challenging nature of CCSDS algorithms, their parallelization is possible and can provide significant performance benefits.

Keywords: embedded GPUs, multi-core, space compression.

**Both first authors contributed equally to the paper.*

1 Introduction

The on-board processing requirements of future space missions are constantly increasing, requiring new hardware to satisfy this need.Embedded COTS platforms featuring multi-core CPUs and GPUs are promising candidates, combining high-performance and low power consumption. The GPU4S (GPU for Space) ESA-funded project [1] studies whether on-board processing algorithms are amenable to GPU parallelization as well as whether embedded GPUs can satisfy the performance requirements of future space missions, effectively paving the way for their adoption.

However space compression algorithms are among the most challenging space processing algorithms in order to be parallelized, due to their inherent sequential nature, created by data dependencies.

Due to the importance of data compression, current spacecraft include specific ASIC or FPGA implementations of the various space compression CCSDS standards for supporting these tasks. However, an efficient parallel software implementation of these standards targeting embedded multi-core CPUs and GPUs can allow a series of benefits for future space missions.

There are 3 main families of space compression standards defined by the Consultative Committee for Space Data Systems (CCSDS) which consists of representatives from several space agencies and private corporations: CCSDS 121 covers general purpose data compression in a lossless way, CCSDS 122 covers both lossless and lossy image compression and CCSDS 123 focuses on hyper-spectral lossless and near-lossless compression. Early project results in GPU4S with commonly used processing algorithms [2] indicate that embedded GPUs can provide significant processing improvements of several orders of magnitude compared to existing space processors such as LEON/SPARC. Compared to FPGAs, which are commonly used in on-board processing applications, GPUs offer the capability to reconfigure the on-board processing using software in a fast manner.

We present our work on the acceleration of two of the most widely used space compression standards nowadays, CCSDS 121.0-B-3 [3] and CCSDS 122.0-B-2 [4] using parallel embedded COTS platforms which are considered good candidates for on board processing in the future.

Our results on two embedded platforms with multicore CPUs and GPUs, the NVIDIA Xavier and the AMD Embedded Ryzen V1605B show that the parallelization of space compression algorithms for on-board processing is possible and comparable with existing space solutions. Our implementations are available as open source, as part of ESA's OBPMark (On-Board Processing Benchmark) benchmarking suite [5], focusing on the evaluation of general purpose devices for upcoming space missions, using complex applications relevant to the space domain.

2 Parallelisation approaches

2.1 CCSDS 121.0-B-3

The CCSDS 121.0-B-3 [3] implements a lossless compression of 1D data. The algorithm architecture consists of two blocks, a *preprocessor* and an *adaptive entropy encoder*.

The preprocessor step is optional and can be omitted. It applies a reversible function to the input data to remove the correlation between its values and to convert them in a probability distribution. The predictors work with a block size J parameter which needs to be provided for the decompression.

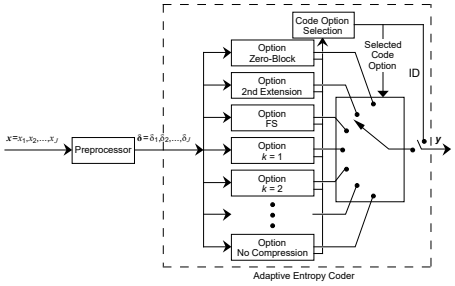

Figure 1: Structure of the CCSDS 121.0-B-3 [3] space compression algorithm. Image courtesy of [3].

The adaptive entropy encoder selects a different encoding of the preprocessed input based on the input data distribution. In fact, the input data are encoded using all the encoders implemented in the adaptive encoder, and the best one is selected i.e. the one which results to a higher compression ratio for each input data block.

The standard defines a series of encoders which can be used: Zero-Block, Second-Extension, Fundamental Sequence, Sample Splitting and No-compression (Figure 1). The Sample Splitting encoder is parametric based on a value k, with $1 \leq k \leq 29$, resulting to a total of 33 possible encoders. Note that similar to the preprocessor, not all encoders are required to be provided by a compliant implementation.

For the parallelization we follow a coarse-grain approach for both the CPU and GPU. In the CPU implementation, which is based on OpenMP, we distribute input data blocks of size J to each of the CPUs in the system. Each of the CPUs applies the entire pipeline shown in Figure 1 on its provided data, and outputs its selected compression encoder and its compressed data.

Our GPU parallelization uses a similar approach implemented in CUDA for the NVIDIA and in OpenCL for the AMD platform. However, instead of distributing the image blocks to the CPUs, we are distributing them in separate *Streams*/*Command Queues*. This version requires more frequent synchronisations compared to the CPU version, in order to synchronise between different kernel invocations, which are more costly.

2.2 CCSDS 122.0-B-2

The CCSDS 122 [4] standard provides lossless or lossy 2D data compression based on the Discrete Wavelet Transform (DWT). It consists of two main functional blocks, the Discrete Wavelet Transform and a Bit-Plane encoder.

The purpose of the Discrete Wavelet Transform is to decompose the input to a high and low frequency components to decorrelate the input data before the encoding.

The standard uses 3 levels of 2D DWTs, each of which we compute in parallel on both the multicore and GPU implementations by applying the one dimensional DWT first in rows and then in columns. For each level, the process is repeated for the top left part of the image in a pyramidal fashion. The remaining values on the top left corner are the ones containing the highest quantity of information and are called DC coefficients. The rest of the values which only add extra information are called AC coefficients. Both lossless and lossy compression can be achieved with this method, by using an integer approximation or floating point version of the transform with higher compression achieved by the latter. We obtained similar performance for both lossless and lossy compression for each parallel implementation.

The bit planar encoder encodes the coefficients of the decomposed image in blocks consisting of coefficients which correspond roughly to a region of the input image. When the integer transform is used, the encoder exploits information about least significant bits of certain frequency components being 0 as a result of their scaling. DC and AC components follow a different encoding scheme but despite their different encoding algorithms, in both cases their characteristics are taken into account in order to increase the compression ratio, such as the dynamic range they represent. The selected encoding method is specified in the output to enable its reconstruction later. Like CCSDS 121, it is possible that values remain uncoded, especially if this minimizes the number of required bits. If different component encodings require the same number of bits with the uncoded option, the standard mandates the use of the uncoded one.

3 Experimental Results

3.1 Experimental Setup

We execute our implementations on two embedded SoCs featuring multiple CPUs and GPUs, the NVIDIA Xavier and the AMD Embedded Ryzen V1605B. These two embedded platforms are the latest embedded GPUs of these vendors and have been identified as good candidates in terms of theoretical performance and power consumption, by multiple independent studies of using GPUs in space [6] [7] [8] [9] [10] and they are considered for further evaluation of their properties.

Both boards have similar characteristics, and we are using them with 4 enabled CPUs since in the case of the NVIDIA Xavier the manufacturer ensures that the board maximum power consumption is capped at 15W, which has been identified as a limit for on-board processing hardware [7]. For the AMD board such information is not provided by the manufacturer, but it is configured with the same properties for fair comparison. Both boards use Ubuntu 18.04 LTS. However, it is worth to note that OpenCL is not currently supported by AMD out of the box, so we are using a custom driver provided by Bruhnpace AB [11], which might not be as optimized as a driver provided by the GPU vendor.

3.2 Results

For the performance evaluation of our algorithms we report both execution times as well as MSamples/s and MPixels/s which are the standard metrics used in the literature. During the execution we measure the voltage and the current and report the maximum power consumption of the boards for each experiment.

As we have mentioned, our GPU implementations are parametric, so the number of streams and thread block sizes are configurable. We tune these parameters in order to select the values that provide the best performance. For the CPU version, OpenMP automatically uses the number of available cores, which is 4 in both boards.

3.2.1 *CCSDS 121*

For the evaluation of our CCSDS 121 implementations, we use the standard methodology followed by other works in the literature, using randomly generated data. In particular, we use 16 MB of randomly generated data which is divided in 1024 Steps, each of which consists of 256 blocks of 64 bytes. Moreover, we ensure that all the compared implementations use the same input data and produce identical output.

Figure 2 shows the results between the sequential implementation and our parallel versions for both platforms for various Block Sizes (J values). We notice that the sequential CPU version is faster in the AMD, which means that the Embedded Ryzen x86 CPU has higher performance than the NVIDIA designed "Carmel" ARM v8.2 CPU. Similarly, the OpenMP version is faster in AMD than in the NVIDIA platform. In both cases, we see a speedup of the parallel version compared to the sequential one, 81% in the Xavier and an impressive 2.2× in the Emdedded Ryzen.

Regarding the GPU performance, the NVIDIA GPU provides a speedup of up to 2.1× over the sequential version but equivalent or lower performance than the parallel CPU version on the same platform. However, in the AMD platform, the GPU version provides similar performance with the sequential version and it is 2× slower than the parallel CPU version on the same platform.

We have identified that the reason of low GPU performance comes from the use of atomic operations. We are using the atomics operations for the implementation of the ZeroBlock encoder. Due to the high overhead of atomics on the AMD platform, ZeroBlock becomes the bottleneck of the GPU implementation. In NVIDIA GPUs, each hardware generation reduces the cost of atomics operations. However, in AMD GPUs such information is not available.

The AMD platform contains a more powerful CPU than the Xavier, as it can be seen in terms of absolute performance (Mpixels/s) for the sequential version. This in addition to the fact that the overhead of atomics is smaller in the CPU, due to smaller number of threads translates to exceptional multicore performance, which is faster than the GPU implementations of both platforms.

Moreover, we observe that the maximum performance is obtained for block size 16. Our multi-core performance on the AMD V1605B is close to the requirement of 60 MSamples/s which is usually a target for space applications, as specified by a recent ESA funded FPGA development project which resulted to the best state-of-the-art CCSDS 121 hardware implementation [12]. Table 1 shows the power consumption for the same experiments. However, there are no reported power consumption data for state-of-the-art CCSDS 121 implementations for comparison.

3.2.2 *CCSDS 122*

For the evaluation of our parallel implementations in this compression algorithm, again we have followed the standard practice used in the literature for the performance evaluation of this algorithm implementations, using uniform images such as completely black, random generated images and real

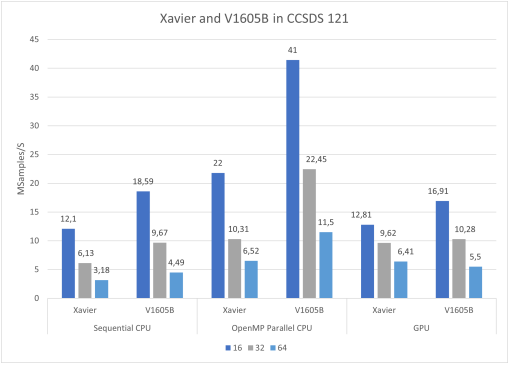

Figure 2: CSDS121 performance in MSamples/s for the NVIDIA Xavier and V1605V fixed at 1024 steps with 256 sample intervals for different Block Sizes J.

J Size	CUDA	Sequential	OpenMP
16	9.16 W	9.35 W	9.67 W
32	9.28 W	9.03 W	9.83 W
64	9.03 W	9.40 W	9.89 W

Table 1: Measured power consumption when running CCSDS 121 with different J size on the NVIDIA Xavier platform.

images from space missions. In particular, we have employed several synthetic and real images of different sizes, such as one from NASA's Mars Pathfinder Mission from the area surrounding Yogi [13] (a rock named by Geoffrey A. Landis) and one from NOAA, taken by the Metop C satellite on 2019-12-21 during its ascending orbit direction.

Table 2 shows the comparison of our implementations using the floating point implementation of the DWT. Although we implemented both integer and floating point version of the DWT, the results are very similar, so we only show one of them. The same trends observed with the CCSDS 121 implementations are visible in this version, too, however on a different scale. In this algorithm, the OpenMP implementations are only slightly faster than the sequential versions, and the AMD CPU is faster than the ARM CPU. On the other hand, the NVIDIA GPU provides a 10× performance benefit compared to the sequential version. However, the AMD GPU is significantly slower (4×) than the sequential CPU version.

According to our analysis, the bottleneck in the AMD GPU implementation comes from additional memory copies and the conversion operations between floating point and integers, which are very costly in the AMD platform. These operations are performed just before and after the DWT stage. As an indication, in the integer version of DWT for the 122, the copy operation takes 3s, while in the Xavier 20ms. Similarly, in the floating point version, in the AMD platform the memory copy and conversion takes 9.4s, while in the Xavier it takes 300ms. Again, we don't know if this is a hardware issue e.g. if the memory bandwidth of the AMD platform is lower than the one of the Xavier, or an issue of the unofficial driver e.g. the driver does not use DMA for the memory transfers, or does not overlap kernels and memory copies. Of course, the sequential version of the algorithm does not require these very

Image	Secuential (Xavier)	Sequential (V1605B)	OpenMP (Xavier)	OpenMP (V1605B)	CUDA (Xavier)	OpenCL (V1605B)
NOAA Image	5.48 Mp/s 9.07 W	7.194 Mp/s ∼15 W	5.96 Mp/s 9.23 W	7.918 Mp/s ∼15 W	6.466 Mp/s 11.48 W	0.906 Mp/s ∼15 W
Mars marspath	6.16 Mp/s 9.02 W	7.434 Mp/s ∼15 W	5.01 Mp/s 8.96 W	7.204 Mp/s ∼15 W	5.636 Mp/s 9.35 W	0.878 Mp/s ∼15W
Random (2048 x 2048)	3.844 Mp/s 9.50 W	5.478 Mp/s ∼15 W	4.32 Mp/s 9.76 W	7.302 Mp/s ∼15 W	17.047 Mp/s 11.52 W	0.876 Mp/s ∼15 W
Black (2048 x2048)	3.75 Mp/s 9.35 W	6.108 Mp/s ∼15 W	4.66 Mp/s 9.62 W	7.866 Mp/s ∼15 W	16.78 Mp/s 10.28 W	0.86 Mp/s ∼15 W
Random (4096 x 4096)	3.308 Mp/s 9.43 W	5.398 Mp/s ∼15 W	4.14 Mp/s 9.70W	6.014 Mp/s ∼15 W	30.642 Mp/s 13.08 W	0.88 Mp/s ∼15 W
Black (4096 x4096)	3.24 Mp/s 9.24 W	5.912 Mp/s ∼15 W	4.21 Mp/s 9.88 W	6.482 Mp/s ∼15 W	32.43 Mp/s 12.54 W	0.882 Mp/s ∼15 W

Table 2: Performance in MPixels/s and average maximum power execution for CCSDS122 with various images and sizes

expensive transfers, so the CPU sequential version is faster than the GPU one in the AMD platform.

Compared to a state-of-the-art space qualified ASIC implementation [14], which matches the space requirement of 60 MPixels/s compression rate, we obtain half of its performance with double power consumption on the Xavier GPU implementation. However, [14] only supports pixel ranges up to 16 bit, while our evaluation has been performed using 32 bit arithmetic and it is configurable to use up to 64 bits per pixel. This illustrates a significant difference between software implementations and ASIC hardware implementations, that can be easily extended for the requirements of future missions, which will use higher resolutions and larger dynamic ranges per pixel. Moreover, note that the highest performance in our implementations is achieved with larger images, regardless of whether they are random or uniform. This means that our approach will benefit from the larger image sizes which will be used in future missions.

4 Conclusions and Future Work

In this on-going research paper we presented our work on the CPU and GPU parallelization for CCSDS 121 and 122.We have shown that although the parallelization of compression algorithms is challenging, it is possible to obtain significant speedups with their CPU and GPU parallelization, as our results on two embedded GPU platforms show. In fact, our obtained results are very close to the requirements of existing space missions both in terms of performance as well as in power consumption. Moreover, they are competitive with existing space processors.

As a future work we intend to finish our parallel implementations on the CCSDS 123.0-B-1, since it shares several common blocks with the CCSDS 121.0-B-3. Moreover, we would like to further investigate the issue of the low performance of the AMD GPU despite the fact that its characteristics are similar to the NVIDIA one. A possible reason is the custom GPU driver we are using, so we will explore other possibilities to confirm this fact or rule it out.

5 Acknowledgments

This work was funded by the Ministerio de Ciencia e Innovacion - Agencia Estatal de Investigacion (PID2019-107255GB-C21/AEI/10.13039/501100011033 and IJC-2020-045931-I) and partially supported by the European Space Agency (ESA) through the GPU4S (GPU for Space) activity and the HiPEAC Network of Excellence.

References

[1] L. Kosmidis, I. Rodriguez-Ferrandez, A. Jover-Alvarez, S. Alcaide, J. Lachaize, A. C. O. Notebaert, and D. Steenari, “GPU4S: Major Project Outcomes, Lessons Learnt and Way Forward,” in *Design, Automation and Test in Europe Conference and Exhibition, (DATE)*, 2021.

[2] L. Kosmidis, I. Rodriguez, A. Jover, S. Alcaide, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in Space - Latest Project Updates,” *Elsevier Microprocessors and Microsystems*, vol. 77, Sept 2020.

[3] CCSDS The Consultative Committee for Space Data Systems, *CCSDS 121.0-B-3, Lossless Data Compression*. CCSDS Blue Book, 2020. https://public.ccsds.org/Pubs/121x0b3.pdf.

[4] CCSDS The Consultative Committee for Space Data Systems, *CCSDS 122.0-B-2, Image Data Compression*. CCSDS Blue Book, 2017. https://public.ccsds.org/Pubs/122x0b2.pdf.

[5] ESA, “OBPMark (On-Board Processing Benchmarks),” 2021. http://www.obpmark.org.

[6] Powell, Wesley and Campola, Michael and Sheets, Teresa and Davidson, Abigail and Welsh, Sebastian, “Commercial Off-The-Shelf GPU Qualification for Space Applications,” tech. rep., NASA, 2018.

[7] L. Kosmidis, J. Lachaize, J. Abella, O. Notebaert, F. J. Cazorla, and D. Steenari, “GPU4S: Embedded GPUs in Space,” in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, pp. 399–405, Aug 2019.

[8] Mr. Nan Li, Mr. Aimin Xiao, Mr. Mengxi Yu, Dr. Jianquan Zhang, Dr. Wenbo Dong, “Application of GPU on-orbit and Self-adaptive Scheduling by its Internal Thermal Sensor,” in *International Astronautical Congress (IAC)*, 2018.

[9] F. C. Bruhn, N. Tsog, F. Kunkel, O. Flordal, and I. Troxel, “Enabling Radiation Tolerant Heterogeneous GPU-based Onboard Data Processing in Space ,” *CEAS Space Journal*, vol. 12, pp. 551–564, June 2020.

[10] D. Luchena, V. Schiattarella, D. Spiller, M. Moriani, and F. Curti, “A new complementary multi-core data processor for space applications,” 10 2018.

[11] Unibap AB and Mälardalen University, “"Bruhnspace ROCm project for AMD APUs",” 2020. https://bruhnspace.com/en/bruhnspace-rocm-for-amd-apus/.

[12] U. de Las Palmas de Gran Canaria, “Expro+ esa ao/1-8032/14/nl/ak ccsds lossless compression ip-core space applications,” tech. rep., Universidad de Las Palmas de Gran Canaria, 2017.

[13] N. P. Project, “False color image of the area surrounding yogi, nasa mars pa,” jun 1998.

[14] J.-L. Poupat, “Cwicom & coreci: Towards a highly integrated & innovative image compression unit,” *ESASP*, vol. 694, p. 35, 2011.