Compression techniques with a focus on astronomical data: An overview

Heinrich Strauss
Department of Computer Science
University of Cape Town
hstrauss@cs.uct.ac.za

ABSTRACT

Radio-Astronomical Observation data in its raw format is comprised of noisy data tiles which are cleaned before being subjected to manual and/or automated analysis. We consider in this synthesis the methods employed which provide good compression rates while maintaining data integrity for the observed data and noise and their applicability in the "General-Purpose on GPU" programming paradigm.

1. INTRODUCTION

The amount of astronomical data captured has increased quite dramatically in the last 30 years: early sensors would capture images at 512×512 with a 16-bit value per observed pixel. More recently, it is not uncommon to find image plates which have been captured at 16384×16384 with 16-bit, or even 32-bit, values per image plate per radio source.

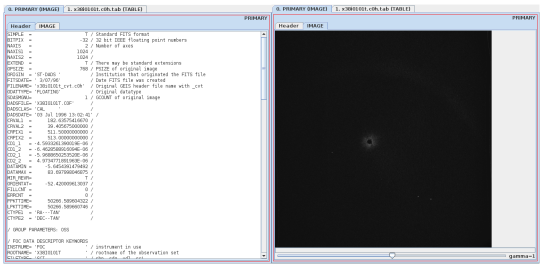
Much of the captured data is noise from various sources: atmospheric noise, noise from the sensing Charged-Coupling Devices (CCDs), radio interference from other transmitters in observed spectra, interference on the transmitting channel to the central storage, among others. While much of this is discarded during the data-cleanup phase prior to processing, data custodians are often uncomfortable with completely removing these data-points, since it is time-dependent observational data which is not obtainable again and may be employed to explain researched phenomena[4].

No compression algorithm can work equally well on every data set[16]. We investigate possible methods of compressing the raw data from the receivers, with a strong focus on lossless compression methods, so that data can be archived as raw data in the event they are needed again. If this can be transformed into a massively parallelisable form, General-Purpose Computing on a GPU (GPGPU) methods may be of value in speeding up the handling of these data. This has already implemented for signal convolution, as shown by Harris, et al.[6] and for detection by Resnick, et al.[15]

1.1 Flexible Image Transport System (FITS)

Flexible Image Transport System (FITS) is an open standard format for interchange of scientific data[18]. Since 1981, it has evolved to accomodate the varying types of data in the scientific fields and is still used in the astronomical data field as the de-facto standard for information interchange.

The data format depends on a number of header "cards" interleaved in the data blocks which describe the format of the data. These are limited to 80 7-bit ASCII characters for each card and there are 36 cards per data block. The individual FITS files are described by the (implementation-based) metadata encoded in these and contain one or more image plates. This allows for arbitrary data to be encoded into the FITS files and compresses well under standard data-file compression implementations. This is possible since the cards can be used to determine the data-type stored and therefore make predictions on the location of lossless compressible data.



```
SIMPLE  =                    T / Standard FITS format
BITPIX  =                  -32 / 32 bit IEEE floating point numbers
NAXIS   =                    2 / Number of axes
NAXIS1  =                 1024 /
NAXIS2  =                 1024 /
EXTEND  =                    T / There may be standard extensions
OPSIZE  =                  768 / PSIZE of original image
ORIGIN  = 'ST-DADS '           / Institution that originated the FITS file
FITSDATE= ' 3/07/96'           / Date FITS file was created
FILENAME= 'x38i0101t_cvt.c0h'  / Original GEIS header file name with _cvt
ODATTYPE= 'FLOATING'           / Original datatype
SDASMGNU=                    1 / GCOUNT of original image
DADSFILE= 'X38I0101T.C0F'      /
DADSCLAS= 'CAL     '           /
DADSDATE= '03 Jul 1996 13:02:41' /
CRVAL1  =      182.63575416670 /
CRVAL2  =      39.405675000000 /
CRPIX1  =      511.50000000000 /
CRPIX2  =      513.00000000000 /
CD1_1   = -4.5933261390019E-06 /
CD1_2   = -6.4628588916094E-06 /
CD2_1   = -5.9688650253520E-06 /
CD2_2   =  4.9734771891963E-06 /
DATAMIN =     -5.6454391479492 /
DATAMAX =      83.697998046875 /
MIR_REVR=                    T /
ORIENTAT=     -52.420009613037 /
FILLCNT =                    0 /
ERRCNT  =                    0 /
FPKTTIME=      50266.589604322 /
LPKTTIME=      50266.589660746 /
CTYPE1  = 'RA---TAN'           /
CTYPE2  = 'DEC--TAN'           /

/ GROUP PARAMETERS: OSS

/ FOC DATA DESCRIPTOR KEYWORDS
INSTRUME= 'FOC               ' / instrument in use
ROOTNAME= 'X38I0101T         ' / rootname of the observation set
```

Figure 1: A FITS header file and the output from the PRIMARY table.

By experimentation over the data-set provided by the Sloane Digital Sky Survey (SDSS), the ratio is about 1:2 by experimentation on the SDSS dataset of 2009.

The compressibility stems from the adjacent points, which are often close to each other (and to the background mean) in value, with noise or reference objects (e.g. galaxies, stars) sparsely-interleaved after noise-suppression within the original data. By decreasing the entropy in the data from the raw data we obtain much smaller file sizes after compression.

2. NOISE

The noise sources listed in the introduction produce varying types of Gaussian and Poisson-noise, among others, which are distinguishable by the distribution of the noise. Gaussian Noise is considered random enough to preclude compression[17]. It is therefore prudent to exclude as much Radio-Frequency Interference (RFI) and "stable noise" (such as that from the sensors) to improve the compressibility of the data.

Most commonly, 16-bit integers or single-precision floating point numbers are used to describe pixels on each 2D image plate captured[12]. If floating point numbers are used, the accuracy cannot be ensured, so lossy algorithms (and their implicitly higher compression ratios) become quite attractive.

3. COMPRESSION METHODS OVERVIEW

3.1 Wavelet Transforms

For at least the decade past, Wavelet Transforms have been commonly used to encode the astronomical data in such a way that compression is feasible[17]. The most commonly observed transform in the literature is the 2-D Haar-Transform (H-Transform). This standard method of compression (Hcompress) was suggested by White and Percival, which could be used in either lossless or lossy mode, depending on whether integer arithmetic is used[19], is as follows:

1. Manipulate the image to obtain roughly equal noise per pixel
2. Apply the H-Transform
3. Encode the output using quadtress and optionally compress the result (which should be quite amenable to standard compression techniques).

This results in simple arithmetic calculations, provided the size of the image being viewed is square with size an integral power of 2. White and Percival estimate that the H-Transform will require about $16N^2/3$ operations for an NxN square[19]. In the case where the image is not a square, it can be trivially extended along the non-conforming axes to yield a square image for the purposes of applying the transform. All stored data can, therefore, considered to be stored in a square matrix format, without loss of generality.

The output of the H-Transform is strongly biased towards zeroes (wherever the background is almost consistent, such as noise-free astronomy images), so RLE should efficiently compress the data.

3.2 Haar Transform

The Haar Transform, or H-Transform[19], is given by the wave function:

$$\psi(t) = \begin{cases} 1 & 0 \le t < 1/2, \\ -1 & 1/2 \le t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

This is run recursively on a dataset and yields a form of the data which is completely invertible, given integer arithmetic. An easy interpretation of this transform-space is whether or not a given square contains values which are brighter than their background. This is used to automatically find points of interest (such as stars, galaxies and clouds) to further investigate. The transformed data can then be run through a quad-tree encoding so that the image may be retrieved progressively. This allows a quick overview of the individual image plates to ascertain whether interesting points are present. In the event that nothing of consequence is identified on the image plate, the data retrieval can be aborted in favour of another image where less transmission bandwidth will be used. [19].

It should not be inferred that the H-transform is the best transform for any particular image. It is chosen only to exemplify a Wavelet Transform and the importance of selecting a good transform should not be underestimated for any particular implementation.

3.3 Quad-Tree Coding

Quad-Tree-based implementations are often suggested as the de facto way of encoding astronomical data[12][9].

For a $2^n \times 2^n$ square, we can view and encode the internal data recursively as 4 equally-sized squares of size $2^{n-1} \times 2^{n-1}$. The data can then be traversed as follows:

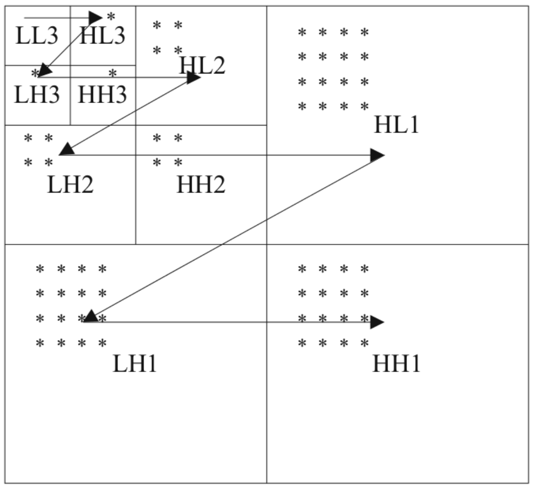


Figure 2: Traversing a Quad-Tree[4]

If we have a region that is uniform in colour (or value), we see that the Quad-Tree traversal will encode them as successive bits of equal value. This allows for easier Run-Length Encoding (RLE). The base-case for a single pixel is stored as the value of the relevant pixel.

In analogue to progressive JPEG encoding[3], this allows individual detail levels to be displayed and the image will become clearer at the scale of each successively smaller Quad-Tree.

4. DISCUSSION

4.1 Du and Ye's Approach

Du and Ye[4] showed that with two transforms, Integer Wavelet Transform (IWT) and Embedded Zero-tree Wavelet (EZW), and a Quad-Tree representation, we can achieve a more compressible data-set with their algorithm IWT+EZW+TS+A. This yields an increased compression factor of 1.46 over

GZIP and 1.20 over the 5/3 transform[3], which was ratified in the JPEG2000 lossless standard, over the LAMOST dataset.

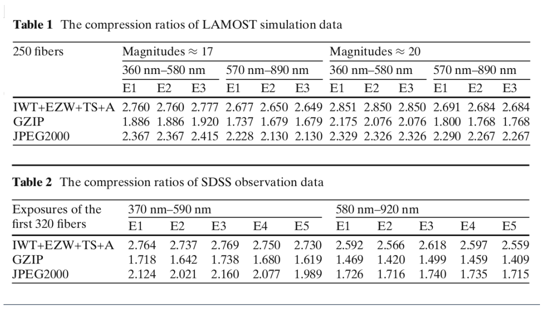


Table 1 The compression ratios of LAMOST simulation data

250 fibers	Magnitudes ≈ 17						Magnitudes ≈ 20					
	360 nm–580 nm			570 nm–890 nm			360 nm–580 nm			570 nm–890 nm		
	E1	E2	E3	E1	E2	E3	E1	E2	E3	E1	E2	E3
IWT+EZW+TS+A	2.760	2.760	2.777	2.677	2.650	2.649	2.851	2.850	2.850	2.691	2.684	2.684
GZIP	1.886	1.886	1.920	1.737	1.679	1.679	2.175	2.076	2.076	1.800	1.768	1.768
JPEG2000	2.367	2.367	2.415	2.228	2.130	2.130	2.329	2.326	2.326	2.290	2.267	2.267

Table 2 The compression ratios of SDSS observation data

Exposures of the first 320 fibers	370 nm–590 nm					580 nm–920 nm				
	E1	E2	E3	E4	E5	E1	E2	E3	E4	E5
IWT+EZW+TS+A	2.764	2.737	2.769	2.750	2.730	2.592	2.566	2.618	2.597	2.559
GZIP	1.718	1.642	1.738	1.680	1.619	1.469	1.420	1.499	1.459	1.409
JPEG2000	2.124	2.021	2.160	2.077	1.989	1.726	1.716	1.740	1.735	1.715

Figure 3: IWT+EZW+TS+A Performance[4]

Since two Wavelet Transforms are performed, it is reasonable to expect that the processing time complexity will increase over the H-transform.

4.2 Pence, Seaman and White

An analysis of the compression methods commonly employed is undertaken by Pence, Seaman and White in [12]. They show that the Rice compression algorithm, implemented from the CFITSIO library[13], yields better compression than GZIP in all tested cases against real-world data gathered from the NOAO Mozaic CCD (16-bit) and the NEWFIRM camera (32-bit). Hcompress yields smaller file sizes than Rice at the expense of much greater processing time.

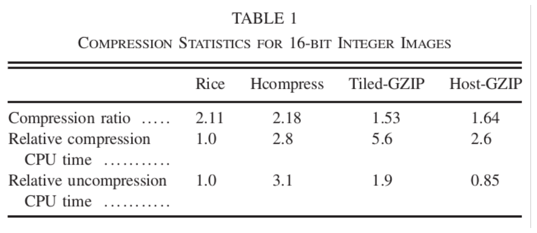


TABLE 1
Compression Statistics for 16-bit Integer Images

	Rice	Hcompress	Tiled-GZIP	Host-GZIP
Compression ratio	2.11	2.18	1.53	1.64
Relative compression CPU time	1.0	2.8	5.6	2.6
Relative uncompression CPU time	1.0	3.1	1.9	0.85

Figure 4: Compression of 16-bit integer images[12]

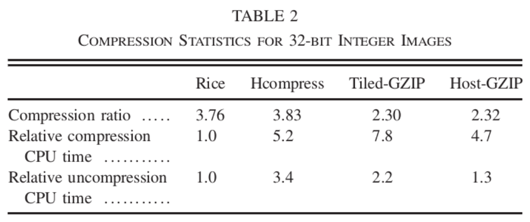


TABLE 2
Compression Statistics for 32-bit Integer Images

	Rice	Hcompress	Tiled-GZIP	Host-GZIP
Compression ratio	3.76	3.83	2.30	2.32
Relative compression CPU time	1.0	5.2	7.8	4.7
Relative uncompression CPU time	1.0	3.4	2.2	1.3

Figure 5: Compression of 32-bit integer images[12]

If 32-bit values become more common in image plates, the compression ratios presented become higher, as there are more redundant bits in the data captured near zero, which make up a large proportion of the captured points. The Rice algorithm is the most efficient algorithm by processing time and average compression ratio. The increase in value representation size implicitly doubles the required transmission and storage bandwidth. At sufficiently large-scale, network bandwidth and storage speed will become an issue. Other than aligning the data for improved processing or compressibility, the changes required to current implementations are minimal.

4.3 General Discussion

The methods described above hinge strongly on Wavelet transforms, Quad-Trees and classic compression techniques. The need for lossless compression is also a recurring theme in the literature. We have seen a natural progression of compression algorithms with Richard L. White contributing to two of the three common implementations [19][2][12]. As one would expect, the newer algorithms are customised for the data they are compressing, and so yield better results.

The suggested pattern for compression of the astronomical data is therefore

- Transform the data using one or more Wavelet Transforms
- Encode the data into a Quad-Tree to decrease entropy and allow progressive decomposition
- Apply standard compression techniques to reduce the amount of bandwidth being consumed.

This has the obvious drawback of being computationally intensive to store and retrieve image plates.

The Wavelet transforms and Quad-Tree implementations seem to lend themselves well to parallelising. Compression requires a dictionary to map expanded data to compressed data, which is complicated if the non-entropic data are divided between computational nodes. The simplest strategy is to define large enough blocks of data and have these manipulated independently. This would introduce delay into the data stream, which precludes real-time observation and manipulation of the data source if interesting points are observed during the capture phase, which is an intended use-case in future.

The common input format for astronomy data compression is the FITS format, which allows the data to be stored in a patterned data structure. With the optimisation of the header fields, the compressibility factor can be tweaked, since the generally 16-bit integer or floating-point fields can be transformed for compressibility.

The tendency of the data to approximate the background levels plays a large role in the predictability and repeatability of the data and the improved performance of Rice over other, more commonly implemented algorithms shows that this can be exploited more aggressively by choosing an evolved compression algorithm[12]. Since the observed values are often close to zero, we should take care to ensure that denormalisation of any IEEE-794 floating point values does not affect the accuracy of the data stored. It is also important that denormalisation be consistently used throughout any compression implementation to preserve the lossless nature of the transform.

Finding an optimal Wavelet function is also of critical importance, as can be seen from the work of Du and Ye[4].

By choosing the IWT and EZW transforms, a significant enhancement in the overall compression rate over the H-Transform was noted. Even the JPEG lossless 5/3 transform improved over GZIP by a factor of 1.22. As long as the wavelet is invertible, we are able to perform lossless compression, however the Quad-Tree conversion and compressibility factor would remain as potential hazards to to ability to parallelise. These problems have already been implemented in a parallelisable method, as we now show.

The encoding and alignment of the data allow parallelisable decomposition of the Quad-Tree implementation. This is of importance, as Single-Instruction, Multiple Data methods can be used for the construction of the Quad-Tree. This in turn opens the possibility of a GPGPU implementation for this, a concept which was explored in [8]. Although no concrete examples are shown by Ibarra and Kim, Kelly and Breslow[9] describe this in some detail.

The importance of classic data compression techniques should not be underestimated, as even a modest improvement over uncompressed data allows more data to be transmitted and stored within the same hardware confines. GZIP does not perform well compressing this type of image plates compared to newer methods, such as Rice. This is quite understandable, given that the dictionary would have to account for pixel differences of even 1-bit and GZIP has no means of compressing the differences between values of nearby points[12]. The requirement by the data custodians that the compressed data is accurate precludes lossy algorithms. As long as integer arithmetic is used, the chance that the Wavelet transform is invertible is excellent[1]. Alternately, specifying clear accuracy requirements for the values captured would allow for lossy compression methods to be explored. Adaptive Filtering, which identifies points of interest and weights the distribution of the compressed data towards these points, would be a natural path to investigate.

Franaszek, Robinson and Thomas tested parallel data compression with a shared dictionary in [5]. Execution times were quite comparable to the common serial implementations. If this can be parallelised using GPGPU methods, we hope to see improved performance, though large memory buffer copies may make the task difficult.

Percival and White showed that the data convergence period over a trasmission medium is greatly reduced when sending the transformed data instead of the raw image data[14]. If this can be used to reconstruct the data where they are to be manipulated, it would extend the lifetime of the infrastructure, both for storage and transmission.

With regard to the stored data, the FITS format allows flexibility of data at the cost of customised implementations. Since the data format is likely to be static for a particular implementation, this can be stored in a more efficient transformed format and extracted into FITS data at request, or processed directly from the FITS data. This requires in-depth analysis of the data formats, which will be dealt with during further analysis. Hardware implementations already exist for Huffman-coded data, as described in [10]. Naturally, disk-based and network-based I/O are in the critical path for the data processing: these are orders of magnitude slower than the transfer of the data within a computing node. Even presently, transmission at over 40-gigabit per second speeds is prohibitively expensive and storage systems are similarly impeded, even in the advent of Solid-State Disks.

Many of the compression techniques discussed do not apply to streaming data, as they require data lookahead in order to build the compression algorithm's dictionary[12]. The work of Du and Ye[4], however, alludes to an approach to gain better compression: Hunt and Rodríguez[7] suggest fast piecewise linear predictors to compress the streaming data by heuristically selecting from a number of independent compression predictors. Oseret and Timset[11] suggest Object-Based Compression. Hunt and Rodríguez claim a lossless compression ratio of between 1:1.3 and 1:3.2 averaged over five data sets. Oseret and Timset's method is selectively lossless, but claims compression ratios of about 1:200.

What remains to be shown is how much of the noise in the data can be seperated from the data and if the static noise, such as sensor noise, can be eliminated before compression with no loss of information. Given the size of the datasets, we also need to ascertain whether GPGPU-based methods have enough memory bandwidth to be able to compress the data at realtime rates to enable interactive manipulation in future.

5. REFERENCES

[1] M. D. Adams and F. Kossentni. Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis. *Image Processing, IEEE Transactions on*, 9(6):1010–1024, 2000.

[2] L. P. Deutsch. Gzip file format specification version 4.3. 1996.

[3] G. Dillen, B. Georis, J.-D. Legat, and O. Cantineau. Combined line-based architecture for the 5-3 and 9-7 wavelet transform of jpeg2000. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(9):944–950, 2003.

[4] B. Du and Z. Ye. A novel method of lossless compression for 2-d astronomical spectra images. *Experimental Astronomy*, 27(1-2):19–26, 2009.

[5] P. Franaszek, J. Robinson, and J. Thomas. Parallel compression with cooperative dictionary construction. In *Data Compression Conference, 1996. DCC'96. Proceedings*, pages 200–209. IEEE, 1996.

[6] C. Harris, K. Haines, and L. Staveley-Smith. Gpu accelerated radio astronomy signal convolution. *Experimental Astronomy*, 22(1-2):129–141, 2008.

[7] S. Hunt and L. S. Rodriguez. Fast piecewise linear predictors for lossless compression of hyperspectral imagery. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*, volume 1. IEEE, 2004.

[8] O. H. Ibarra and M. H. Kim. Quadtree building algorithms on an simd hypercube. In *Parallel Processing Symposium, 1992. Proceedings., Sixth International*, pages 22–27. IEEE, 1992.

[9] M. Kelly and A. Breslow. Quadtree construction on

the gpu: A hybrid cpu-gpu approach. *Retrieved June13*, 2011.

[10] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge. Improving code density using compression techniques. In *Microarchitecture, 1997. Proceedings., Thirtieth Annual IEEE/ACM International Symposium on*, pages 194–203. IEEE, 1997.

[11] E. Oseret and C. Timsit. Optimization of a lossless object-based compression embedded on gaia, a next-generation space telescope. In *Optical Engineering+ Applications*, pages 670003–670003. International Society for Optics and Photonics, 2007.

[12] W. Pence, R. Seaman, and R. White. Lossless astronomical image compression and the effects of noise. *Publications of the Astronomical Society of the Pacific*, 121(878):414–427, 2009.

[13] W. D. Pence. New image compression capabilities in cfitsio. In *Astronomical Telescopes and Instrumentation*, pages 444–447. International Society for Optics and Photonics, 2002.

[14] J. Percival and R. White. Efficient transfer of images over networks. In *Astronomical Data Analysis Software and Systems II*, volume 52, page 321, 1993.

[15] G. Resnick, M. Kuttel, and P. Marais. Gpu accelerated source extraction in radio astronomy: A cuda implementation. *University of Cape Town, South Africa*, 2010.

[16] C. E. Shannon and W. Weaver. A mathematical theory of communication, 1948.

[17] J.-L. Starck and J. Bobin. Astronomical data analysis and sparsity: from wavelets to compressed sensing. *Proceedings of the IEEE*, 98(6):1021–1030, 2010.

[18] D. Wells, E. Greisen, and R. Harten. Fits-a flexible image transport system. *Astronomy and Astrophysics Supplement Series*, 44:363, 1981.

[19] R. L. White and J. W. Percival. Compression and progressive transmission of astronomical images. In *1994 Symposium on Astronomical Telescopes & Instrumentation for the 21st Century*, pages 703–713. International Society for Optics and Photonics, 1994.