LASER INTERFEROMETER GRAVITATION WAVE OBSERVATORY
-LIGO-
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Report LIGO-T010033-00-E March 13 2001

Data Compression study with the E2 Data

S.Klimenko, B. Mours, P. Shawhan, A. Sazonov

Distribution of this draft:

This is an internal working note
of the LIGO Project

LIGO Hanford Observatory
P.O. Box 1970;Mail Stop S9-02
Richland, WA 99352
Phone (509) 372-2325
Fax (509) 372-2178
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
P.O. Box 1970;Mail Stop S9-02
Livingston, LA 70754
Phone (225)686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

California Institute of Technology
LIGO Project – MS 51-33
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project – MS 20B-145
Cambridge, MA 01239
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

WWW: http://www.ligo.caltech.edu/

Contents

1 Introduction

In this document we report the results of the data compression study performed for the E2 run. The goal was to study the compression rate which could be achieved on realistic data sets using standard compression methods or new ones. One of the key points of this study was to compare lossy and lossless techniques.

Before starting to describe the results, let us recall that in the frame format (Frame Format Specification LIGO-T970130-D-E and VIRGO-SPE-LAP-5400-102), the compression is applied only at the vector level (the data for the ADC channel), not for the header information. This has two important consequences. First, different compression methods can be used within the same frame for different vectors. This allows us to optimize the compression. Second, since the header information is not compressed there is no need to uncompress all the channels when reading a frame. Only the needed channels need to be uncompressed which saves time during frame handling.

2 Data sets used

The data compression which can be achieved depends on the type of data. For this study we use the E2 data (http://blue.ligo-wa.caltech.edu/engrun/E2). Two kind of data set corresponding to different channel lists were used. The reduced data set (RDS) was a limited set of channels selected to be the most useful ones. The full frames (FF) is a larger data set containing all the channels which may be of some use. Table 1 gives a description of the data set contents. The RDS is in fact not very small because it contains a large number of fast channels stored as floating point. This shows that floating point channels are important and explains why a large fraction of this study was devoted to this type of data.

Table 1. Description of the frame file content

		Reduce Data Set	Full Data Set
Frame size		1 511 840 bytes	3 193 896 bytes
Data fraction taken by floating point channels	16 Hz channels	0.3% (80 channels)	2.3% (1139 channels)
	2 kHz channels	9.8% (18 channels)	6.2% (24 channels)
	16 kHz channels	52.0% (12 channels)	22.6% (11 channels)
	All channels	62.1%	31.1%
Data fraction taken by short integer channels	256 Hz channels	3.0 % (89 channels)	1.9%(119 channels)
	2 kHz channels	14.1% (52 channels)	23.1% (180 channels)
	16 kHz channels	17.3% (8 channels)	35.9% (35 channels)
	All channels	34.4%	60.9%
Data fraction of header, names, ...		3.5%	8.0%
Total number of channels		259	1508
Mean overhead per channel		201 bytes	171 bytes

Remarks:

- Data fraction taken by one channel does not include the overhead of each channel which is quoted separately (line: Data fraction of header, names, ...).
- Since there is some header information valid for a full file (the dictionary or the table of contents) the frame size depends slightly on the number of frames per file. The quoted size has been computed for files of 30 frames.

But data compression depends also on the data values. A quiet channel compresses much better than a channel with large noise. In order to study this effect we define three data stretches corresponding to three different conditions of the interferometer state:

- Stretch A: From H-657904501.F to H-657904530.F. This stretch corresponds to a full lock section.
- Stretch B: From H-657904221.F to H-657904250.F. During this stretch, the IFO lost lock at time = 657904227.
- Stretch C: From H-657904281.F to H-657904310.F. During this stretch, the IFO acquired lock at time = 657904304.

3 Performance of methods available as part of the frame standard

Only lossless compression methods are part of the current frame standard.

3.1 Compression performance for floating point

Only one method (gzip) is available for floating point. The result is presented in the first two lines of table 2 for the three data sets used. The compression is presented as the number of bits needed to store one data point. If compression is not used, this number of bits is 32 for floating point and 16 for short integer. There is no 32-bit integer in LIGO data right now.

The table 2 shows the following results for the floating points:

- The compression factor does not change so much from one data stretch to an other.
- The floating points in the full frames need 2 bits less than the RDS mainly because most of the slow channels (7.4% of the floating point data) are in fact 1 Hz channels with the same value repeated 16 times. This kind of pattern compresses well.
- The gzip compression speed is very slow, slower than the LIGO data rate production. This is a very strong limitation for the use of this technique.

Table 2: Data compression for different data set using standard compression methods

	Number of bits per word			Compression Speed (Mbytes/s)	Uncompression speed (Mbytes/s)
	Data stretch A	Data stretch B	Data stretch C		
Gzip on float (FF)	21.4	21.1	20.3	0.9	3.3
Gzip on float (RDS)	23.7	23.9	23.2	1.8	10.8
Gzip on short (FF)	6.8	6.6	6.7	2.5	9.5
Gzip on short (RDS)	9.2	9.2	9.2	1.7	7.3
Zero suppress on short (FF)	6.1	6.1	6.1	13.2	22.3
Zero suppress on short (RDS)	7.6	7.5	7.5	12.4	19.9

Remarks:

- The number of bits is the average on 30 frames weighted by the number of data points. It does not include the channel overhead.
- The compression speed does not include the frame IO.
- 1 Mbytes = 1024 Kbytes = 1024*1024 bytes.
- All speeds quoted in this report have been measured on the same computer (a Sun ultra 10 running at 450 MHz)

3.2 Compression performance for short integer

Two methods are available for short (16 bits) integers:

- the gzip compression. In this case the data are first differentiated to improve the compression rate.
- the zero suppress method. In this case differentiated date are stored with the minimal number of bits needed. This number of bits is computed for a bloc of data with a typical size of 6 data values. This method is available only for integers.

From the results given in the last four lines of table 2 we can make the following comments:

- Like for floating point, the compression factor does not change so much from one data stretch to an other.
- The zero suppress method gives slightly better results for the compression rate than gzip and is much faster. Therefore the zero suppress method should be used as the reference method. There is no reason to use gzip for short integer.

3.3 Optimization of the bloc size for zero suppress algorithm.

In the zero suppress method, the number of bits used to store the data word is defined for a given number of words (bloc size). If the bloc size is small, we can easily adjust the number of bits to the need of each data point but we have to pay some overhead to store the number of bits (4 bits per bloc). If the bloc size is too long, a spike in one data word will increase the whole bloc size. We used the full frame and RDS data set (stretch A) to optimize this bloc size. Table 3 presents the results. One can see that the optimum value for both kind of data set is a bloc size of 12 instead of 6 as currently used. This optimization will reduce the data size by nearly 1% and increase the compression/uncompression speed by about 6%.

Table 3. Compression rate and speed as function of the bloc size for the zero suppress method

Bloc size (# of words)	Number of bit per word (RDS)	Number of bit per word (FF)	Compression speed Mbytes/s	Uncompression speed Mbytes/s
4	7.763	6.292	12.0	19.8
6	7.633	6.172	12.5	20.7
8	7.587	6.127	12.9	21.7
10	7.573	6.109	13.2	21.6
12	7.569	6.104	13.2	22.0
14	7.574	6.106	13.5	22.5
16	7.577	6.110	13.4	22.7
18	7.591	6.118	13.6	22.3
20	7.600	6.125	13.6	22.4

3.4 Effective compression factor or Summary

Table 4 summarizes the overall performances of the existing compression methods. The compression factor is the effective one which takes into account the frame overhead. The frame writing/reading speed is also an effective one (including the frame overhead) but without the disk IO (the test was done writing/reading in the computer memory).

For applications where speed is an issue, the best method to use right now is only the zero suppress for integer. If more CPU resources are available the combination of the zero suppress and gzip gives the best results.

Table 4. Overall compression factor and speed (Stretch A)

Method and data set	Compression factor (compress./original size)	Writing Speed (MB/s)	Reading Speed (MB/s)
Gzip for all channels (FF)	1.83	1.9	7.4
Gzip for all channels (RDS)	1.44	1.9	8.6
Zero suppress + gzip for float (FF)	1.91	2.8	8.3
Zero suppress + gzip for float (RDS)	1.52	2.6	11.8
Zero suppress only (FF)	1.60	16.0	20.4
Zero suppress only (RDS)	1.22	27.6	35.5

4 A new lossless compression method for floating point numbers

Given the complexity of a floating point number which is in fact composed of two numbers (the exponent and the mantissa) it seems difficult to increase the compression factor. However it is possible to increase the compression speed using the technique described below.

4.1 The Zero suppress method on floating point

Most of the LIGO data are time series. This means that the values change slowly from one value to the next value. Therefore the sign and the exponent may stay constant for several values. Since these quantities are stored in bits 31 to 23 (the most significant bits for an integer) it is possible to deal with floating points like if they were integers and use the simple zero suppress method. The first line of table 5 gives the result of this method with the gzip method put as reference (second line of the table). It turns out that this method give the same compression factor as gzip but is much faster. It is therefore an excellent candidate to replace gzip and to provide a reasonable solution for lossless compression.

Table 5: Lossless data compression result on floating point for the RDS

	Number of bits per word (averaged over 30 frames)			Compression Speed (MB/s)	Uncompression speed (MB/s)
	Sample A	Sample B	Sample C		
Zero suppress on float	23.7	22.9	22.6	29.4	60.2
Gzip on float	23.7	23.9	23.2	1.8	10.8

Table 6 compares channel by channel the compression achieved on floating points for channels sampled at least at 2kHz. Both methods give different results. For a few channels like the H2:ASC-WFS1 channels, gzip gives a much better result because these are not real floating points numbers but just integer cast to floating point. If we remove this kind of ‘false’ floating point channels, then the zero suppress method provides a slightly better compression factor. One can also try to optimize the compression factor regardless of the CPU time spent and take for each channel the best compression method. In this case the mean number of bits for this data set drops from 23.8 (zero suppress) or 23.9 (gzip) to 21.3.

Table 6. Comparison of the lossless compression methods for floating points

Channel name	Frequency	Number of bit		First vector values
		Zero sup.	gzip	
H2:LSC-PRC_CTRL	16384	16.9	24.9	1.23507e-16 1.23310e-16 1.23386e-16
H2:ASC-ITMY_OPLEV_YERROR	2048	17.2	17.7	-.529022 -.528788 -.528020
H2:ASC-QPDX_DC	2048	17.3	11.6	-14518.0 -14463.0 -14491.0
H2:ASC-ITMY_OPLEV_PERROR	2048	17.4	19.4	-.647098 -.646149 -.646955
H2:ASC-QPDX_P	2048	17.9	19.6	-.526519 -.528867 -.528397
H2:ASC-ITMX_OPLEV_YERROR	2048	18.0	17.6	-.574053 -.577454 -.575421
H2:ASC-QPDX_Y	2048	18.4	20.2	-.155807 -.156053 -.155476
H2:ASC-ITMX_OPLEV_PERROR	2048	18.9	19.1	-.476847 -.478528 -.476646
H2:LSC-AS_I	16384	19.6	27.2	1309.83 1321.26 1311.17
H2:LSC-LA_NPTRR	16384	20.2	25.6	.854375 .854651 .845615
H2:LSC-POB_I	16384	20.6	15.9	400.787 398.977 401.051
H2:LSC-LA_NPTRT	16384	20.7	26.5	.917682 .926559 .920355
H2:LSC-POB_Q	16384	20.7	15.9	472.866 453.895 465.121
H2:ASC-ITMY_OPLEV_PITCH	2048	21.5	29.6	-12.1990 -12.2639 -12.2881
H2:ASC-WFS1_IY	2048	21.6	13.1	-1140.00 -1170.00 -1100.00
H2:ASC-ITMX_OPLEV_PITCH	2048	21.8	29.6	-1.62026 -1.78887 -1.91720
H2:ASC-ITMY_OPLEV_YAW	2048	22.3	29.7	1.30952 1.51880 1.76044
H2:ASC-ITMX_OPLEV_YAW	2048	22.7	29.8	.841868 .678134 .529719
H2:ASC-WFS1_QP	2048	22.9	14.3	1600.00 1600.00 1555.00
H2:ASC-WFS1_QY	2048	23.0	14.2	-985.000 -945.000 -930.000
H2:ASC-WFS1_IP	2048	23.4	11.1	-365.000 -335.000 -380.000
H2:ASC-QPDY_DC	2048	24.1	12.7	115.000 152.000 131.000
H2:LSC-AS_Q	16384	24.1	29.4	135.413 130.482 139.485
H2:ASC-QPDY_P	2048	24.5	20.5	-.391304 -.223684 -.404580
H2:LSC-REFL_I	16384	24.8	20.9	24.0333 19.1395 22.8253
H2:ASC-QPDY_Y	2048	26.7	20.9	-.286957 -.131579 -.145038
H2:LSC-REFL_Q	16384	27.4	21.2	7.78312 7.22269 -1.00833
H2:LSC-CARM_CTRL	16384	32.0	29.8	243.576 -486.139 -769.151
H2:LSC-MICH_CTRL	16384	32.0	29.8	-1416.71 -770.770 -612.202
H2:LSC-DARM_CTRL	16384	32.0	29.7	23.0314 99.0709 160.043

4.2 Optimization of the bloc size for zero suppress applied to floating point.

Like for the zero suppress method applied on short integer we can optimize the bloc size for floating point. Table 7 presents the results. The optimal bloc size is 8 words.

Table 7. Compression rate as function of the bloc size
for zero suppress method apply to float.

Bloc size (# of words)	Number of bit per word (RDS)
4	23.804
6	23.744
8	23.741
10	23.761
12	23.794
14	23.837
16	23.871

5 New Lossy Compression methods

Lossless compression methods have limited performances on floating points due to the complexity of the numbers. If we want to improve the compression factor we have to apply lossy compression methods which drop a 'small' fraction of the information. In many cases it is reasonable to do it since the data come from measurement and have some intrinsic instrumental noise. Therefore the typical constraint would be that the numerical noise introduced by the compression algorithm is smaller than the instrumental noise. However this numerical noise may have a complex structure and the effect of the compression noise may not be the same for different kind of analysis. Therefore it is almost impossible to give a general answer.

We will present the two methods which have been studied, a simple one (the float to integer conversion) and a more powerful one (the wavelet).

5.1 The floating point conversion to integer method

Since integers compress much better than floating points, the principle of this method is to convert floating points to integer numbers after storing a scaling factor. To improve the compression, the data are first differentiated before being converted to integers, which requires also to save the first data value in addition to the scaling factor. The only parameter of the method is the number of bits used for the integer.

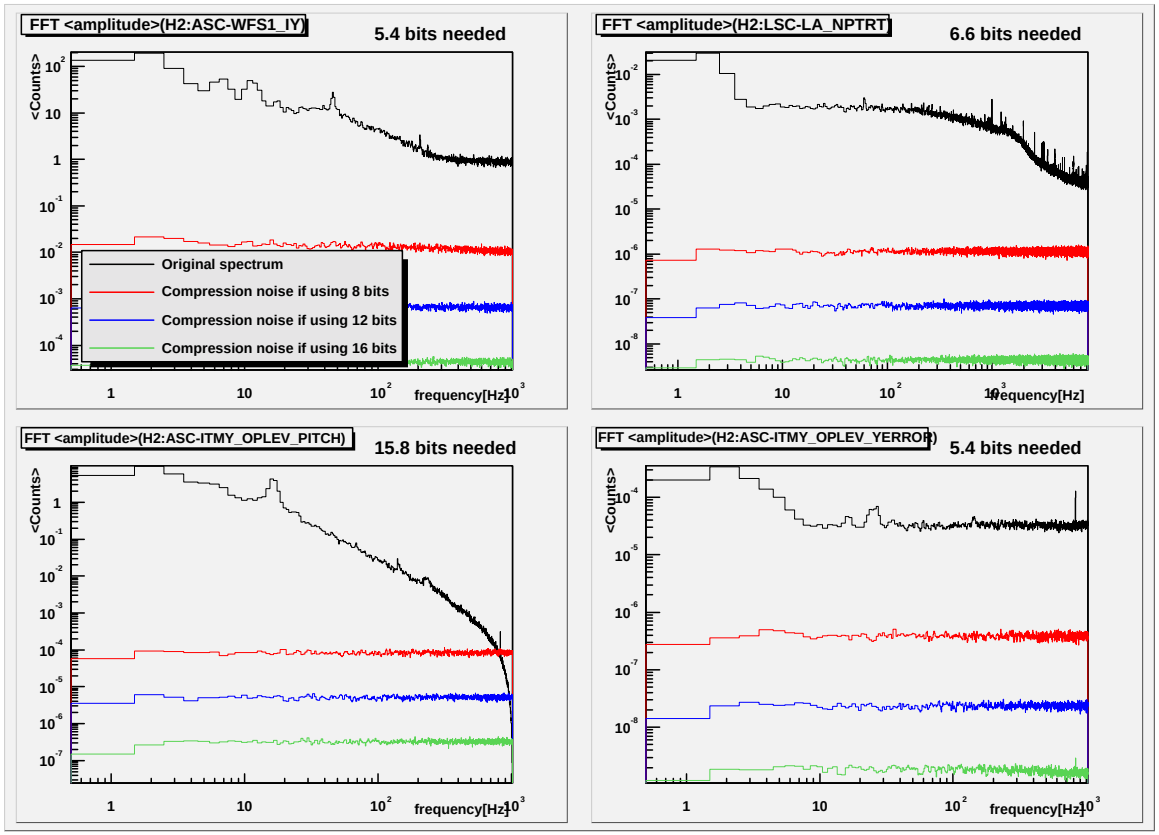

Figure 1. Typical amplitude spectrum and noise level introduced by the float to integer compression method

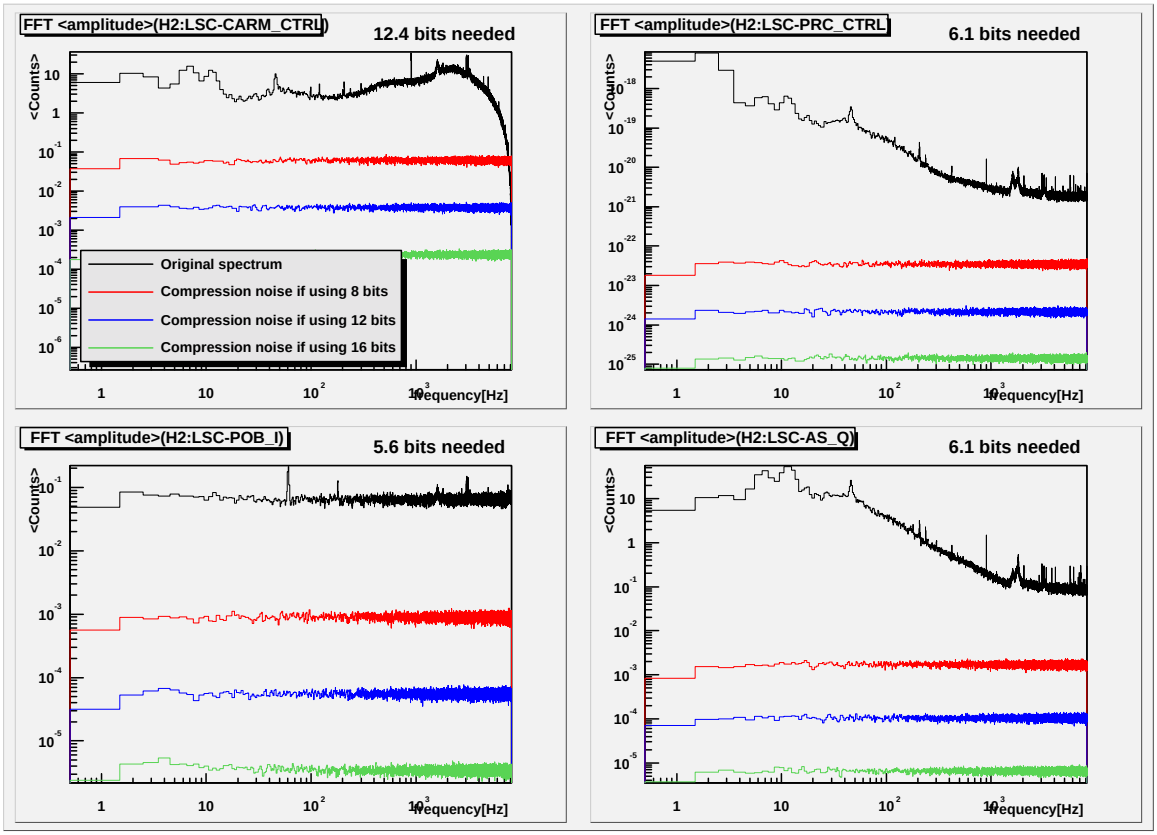

Figure 2. Typical amplitude spectrum and noise level introduced by the float to integer compression method

This is a very simple method which introduces white noise like for a real analog to digital convertor. However to really get white noise the round off of the integer conversion should not be done just on the differentiated data (which would introduce a 1/f noise) but should be done after checking that the rebuilt data do not diverge from the original data set.

Figures 1 and 2 present the typical data spectrum observed and the noise level introduced by an 8 bits, 12 bits and 16 bits storage. For most of the cases, an 8 bits storage seems adequate for this data set.

To quantify the number of bits needed we use the rule that the numerical noise introduce should be one order of magnitude lower than the signal for a given fraction of the frequency band. The reason why we are not using the full frequency band is because some of the channels have anti-aliasing filter killing the spectrum at high frequency. This introduces a dip in the right hand side of the spectrum which requires great numerical precision to be properly reproduced. This is especially true for control signals like H2:ASC-ITMY_OPLEV_PITCH or H2:LSC-CARM_CTRL which do not come directly from an ADC but which have been passed through digital filters. But this ‘feature’ is

present because the data have little interest in this corner. Therefore it seems reasonable to relax the noise constraint in this area.

Table 8 gives channel by channel the numbers of bits needed. The numbers of bits are given for two definitions of the band: 90 and 95% of the frequency band. Since many channels have very similar behavior, table 8 gives also the figure number which represent best the spectrum.

Table 8. Compression achieved for each channel using the float to integer conversion.

Channel name	Spectrum type	Number of bit			
		Float to int (B=90 %)	Float to int (B=95%)	zero suppress	gzip
H2:ASC-WFS1_IY	1a	5.4	5.4	21.6	13.1
H2:ASC-WFS1_QP	1a	5.7	5.7	22.9	14.3
H2:ASC-WFS1_QY	1a	5.2	5.2	23.0	14.2
H2:ASC-WFS1_IP	1a	5.2	5.2	23.4	11.1
H2:LSC-LA_NPTRR	1b	6.7	6.7	20.2	25.6
H2:LSC-LA_NPTRT	1b	6.6	6.6	20.7	26.5
H2:ASC-ITMY_OPLEV_PITCH	1c	13.8	15.8	21.5	29.6
H2:ASC-ITMX_OPLEV_PITCH	1c	13.3	15.1	21.8	29.6
H2:ASC-ITMY_OPLEV_YAW	1c	13.3	15.0	22.3	29.7
H2:ASC-ITMX_OPLEV_YAW	1c	13.1	15.0	22.7	29.8
H2:ASC-ITMY_OPLEV_YERROR	1d	5.4	5.4	17.2	17.7
H2:ASC-ITMY_OPLEV_PERROR	1d	5.4	5.4	17.4	19.4
H2:ASC-ITMX_OPLEV_YERROR	1d	5.3	5.3	18.0	17.6
H2:ASC-ITMX_OPLEV_PERROR	1d	5.6	5.6	18.9	19.1
H2:ASC-QPDX_DC	1d	5.4	5.4	17.3	11.6
H2:ASC-QPDX_P	1d	5.4	5.4	17.9	19.6
H2:ASC-QPDX_Y	1d	5.4	5.4	18.4	20.2
H2:ASC-QPDY_DC	1d	5.3	5.3	24.1	12.7
H2:ASC-QPDY_P	1d	5.7	5.7	24.5	20.5
H2:ASC-QPDY_Y	1d	5.5	5.5	26.7	20.9
H2:LSC-CARM_CTRL	2a	10.4	12.4	32.0	29.8
H2:LSC-MICH_CTRL	2a	13.7	16.7	32.0	29.8
H2:LSC-DARM_CTRL	2a	10.8	12.9	32.0	29.7
H2:LSC-REFL_Q	2b	6.2	6.2	27.4	21.2
H2:LSC-PRC_CTRL	2b	6.1	6.1	16.9	24.9
H2:LSC-AS_I	2c	6.3	6.3	19.6	27.2
H2:LSC-AS_Q	2c	6.1	6.1	24.1	29.4
H2:LSC-REFL_I	2c	5.8	5.8	24.8	20.9
H2:LSC-POB_I	2d	5.6	5.6	20.6	15.9
H2:LSC-POB_Q	2d	5.9	5.9	20.7	15.9

Remark: the gray lines correspond to the channel for which the number of bits needed to store the data changes when going from a 90% band coverage to a 95% coverage.

Finally, table 9 gives the mean number of bits obtained and the processing speeds. If we want to reduce (or increase) by a factor 2 the factor 10 of margin between the spectrum and the noise introduced, then the number of bit is decreased (increased) by one. The compression speed is given without including the FFT speed which may be needed to determine the number of required bits. A priori, such a mechanism could divide the compression speed by a factor 2 given the usual speed of the FFT. But this was not checked.

This method gives a factor 3 improvement in the compression factor. It is simple to apply especially for the uncompression but it requires some care in the definition of the number of bits used. It is worth noticing that once the data set has been compressed/ uncompressed one time, the next time the compression does not introduce additional noise if we use the same number of bits to encode the data.

Table 9. Overall compression performances for floating point (data stretch A)

	Number of bits per word	Compression Speed (MB/s)	Uncompression speed (MB/s)
Float to integer conversion (B=90%)	7.5	11.1*	24.*
Float to integer conversion (B=95%)	8.0	11.1*	24.*
Zero suppress on float	23.7	29.4	60.2
Gzip on float (RDS)	23.7	1.8	10.8

* The compression/uncompression speeds have been computed by summing the time needed for a float to 16 bits conversion plus the zero suppress algorithm applied on 16 bits integers. A direct conversion to the right number of bits should give the same or a slightly faster speed.

5.2 The Wavelet compression method

Data compression with wavelets has been studied in LIGO for a while (for details see http://www.phys.ufl.edu/ligo/wavelet/compress.html). It is a powerful method which could give a very high compression factors. The basic idea of the method is to reduce the dynamic range of the data in the wavelet domain. If scale factors are the same for all wavelet layers, it is equivalent to the method described above (data dunamic range reduction in time domain with float to integer conversion). If scale factors are proportional to the data rms in the layers, it introduces a colored noise, which follows the noise curve of the signal. For example, one can decide to have a good representation of

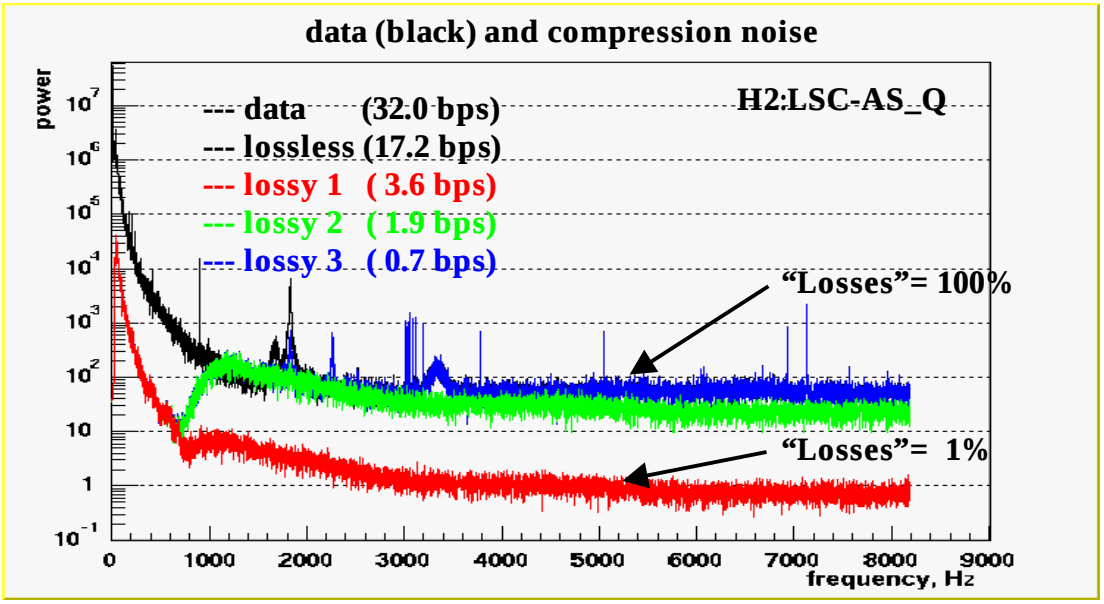

Figure 3 Various compression types and noise levels introduced by the wavelet compression

the low frequency part with only a crude representation of the high frequency part in order to be only sensitive to glitches. This is well illustrated by figure 3 for the asymmetric port channel compressed with different choices of losses. In this case it is possible to go as low as 0.7 bits per sample if we only want to keep the detail information below 1 kHz.

Table 10 presents the performances of the wavelet compression, compared to the previously studied methods. The data sample used was the usual RDS stretch A. The parameters used for the wavelet were the typical ones (see table 10).

Table 10. Wavelet compression performances compared to other methods (using the RDS stretch A data set)

Type of data	Method	Number of bits per word (sample A)	Compression Speed (Mbytes/s)	Uncompression speed (Mbytes/s)
Float	Wavelet compression	4.4	2.3	2.3
	Float to int conversion (B=90%)	7.5	11.1	24.
	Zero suppress	23.7	29.	60.
	Gzip	23.7	1.8	11.
Short	Wavelet compression	4.9	0.9	0.9
	Zero suppress	7.6	12.	20.
	Gzip	9.2	1.7	7.3

Remarks

- Wavelet parameters used:
  - number of Wavelet layers = 3
  - number of Wavelet binary tree layers = 4
  - wavelet tree loss value 1 and 2 = 1
  - lifting wavelet order = 10
  - Daubechies wavelet order = 10
- The wavelet compression factor is given only for the 2 kHz and 16 kHz channels because time series with lower rate are too short to performed the wavelet transform. The slower channels, data header have not been compressed (about 7% of the data) in this case.

For most of the LIGO channels (~90%) the wavelet compression introduces noise, which is at least one order of magnitude below the signal amplitude, and preserves the low frequency part of the spectrum (see Figure 4). By following more closely the high frequency part of the spectrum it gives a better result (4.2 bits per words) than the float to integer conversion (6 bits).

However aggressive wavelet compression can't be used blindly for all data channels. It may introduce artifacts for signal with strong narrow lines (like power monitoring channels). Figure 5 illustrates the problem. In this case, the wavelet compression introduces significant noise between the lines (it is also true for other lossy compression methods with comparable compression factor). There are several ways to solve this problem. One of them is to remove strong lines before applying wavelet compression, the residual signal will be well behaved, so wavelet compression can be safely applied.

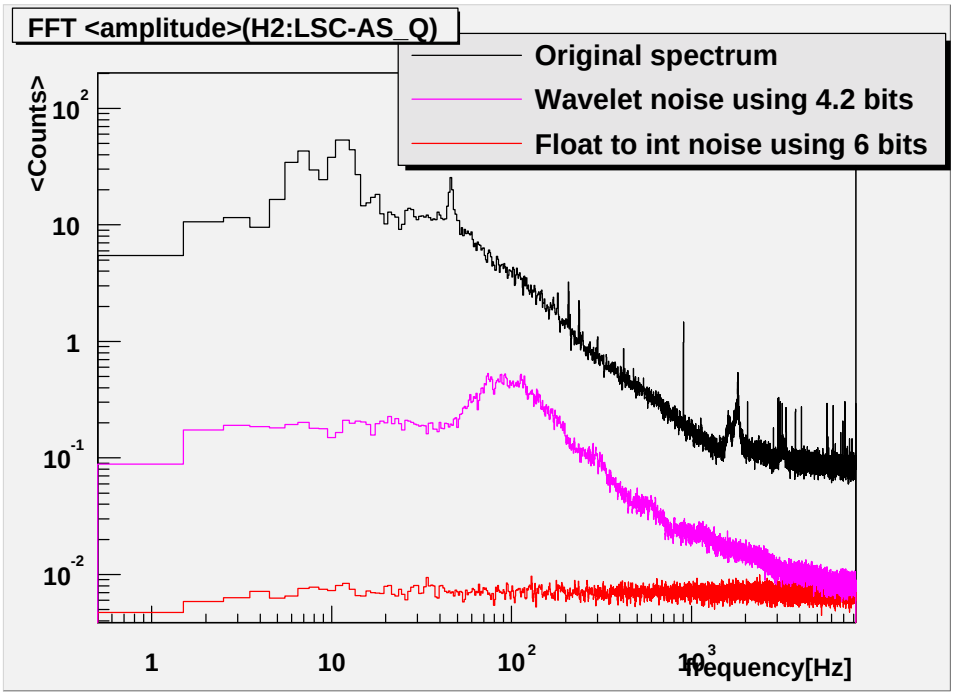

Figure 4. Noise introduced by the wavelet compression on a typical channel.

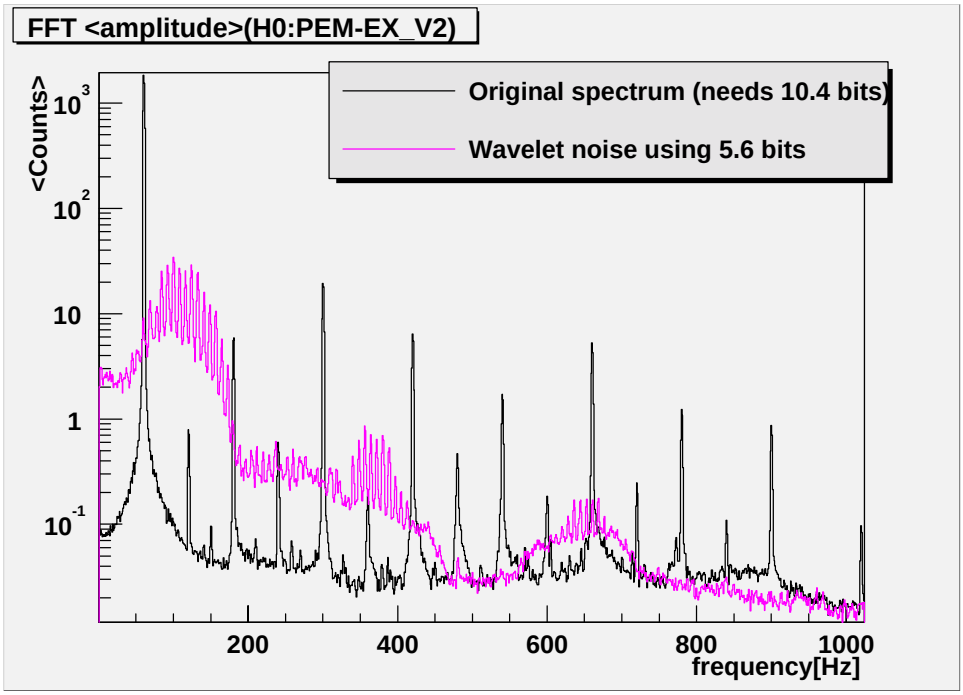

Figure 5. Noise introduced by the wavelet compression on a channel with strong lines

6 Compression with MPEG3

The figure 6 presents the result obtained (S. Klimenko, LSC meeting Aug 16, 2000) using the MPEG3 compression method. The compression factor achieved is not better than the one obtained using the wavelet method but it may introduce more bias. This is because the MPEG3 compression algorithm has been adapted to the human ear

perception model which is not the best to represent LIGO sensors. The MPEG3 is also quite slow and difficult to adapt to LIGO needs. Therefore we do not recommend to use this method as a compression technique.

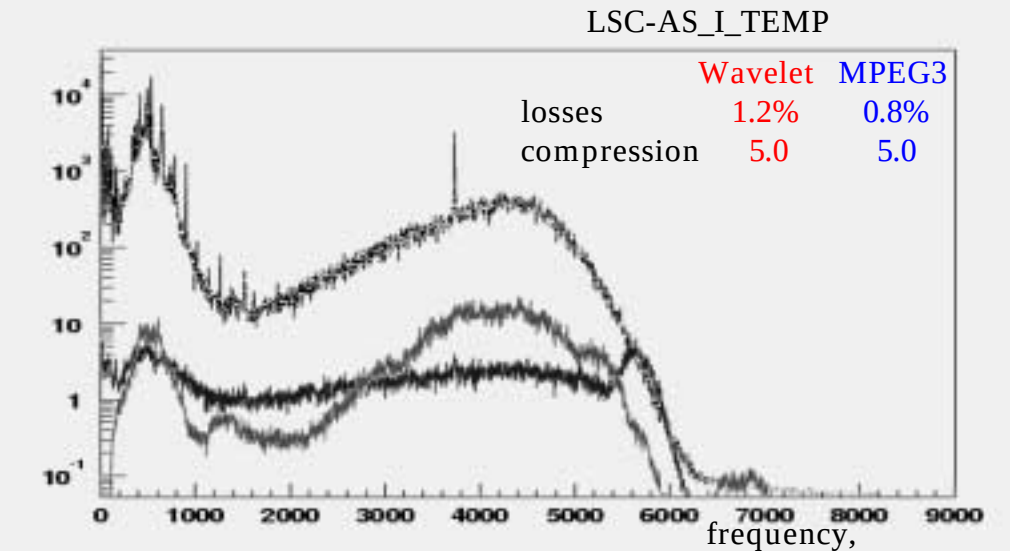

Figure 6 Comparison of the noise introduced by the wavelet compression and the MPEG3 compression

7 Summary

The compression methods currently included in the frame standard work well for short integer but are too slow and not very efficient on floating points. By adopting a new simple compression scheme for floating point (zero suppress method) it is possible to fix the speed problem and to provide a fast (~20MB/s) lossless compression scheme. The overall compression factor is around 1.5 (RDS) to 1.9 (full frame), depending on the fraction of data stored as floating point.

Further increase of the compression factor requires the use of lossy methods. The float to integer conversion is a simple and fast method which can provide on overall compression factor of 2.9/2.6 (RDS/FF). This method can be used for quasi-lossless compression when losses are well bellow 1%. A more advanced technique would be to use wavelet compression which allows the user to control the trade-off between compression factor and losses as a function of frequency. But all these lossy compression techniques require a careful cross check, channel by channel of the introduced noise. They may be used on the environmental and some control channels where a detail structure of signals may not be important. The lossy methods also can be used to generate reduced data sets for data analysis and investigation tasks.

It should also be noticed that when the compression factor becomes large (4 or higher) the frame header information could be as large as 20% of the total amount of data. Increasing the frame length by a factor 4, for instance, would be therefore useful.