Losless compression of LIGO data.

S.Klimenko, G.Mitselmakher, A.Sazonov

This is an internal working note of the LIGO Project.
Abstract

A method for lossless compression of LIGO data based on integer wavelet transforms is described. It works in combination with fast and simple data encoder optimized for compression of Gaussian random signals. The results of data compression for the engineering run data are presented for different compression algorithms.

Contents

1 Introduction 2
2 Random Data Encoder. 2
3 Integer wavelet transform. 3
   3.1 Lifting Wavelet Transform ................................. 4
   3.2 Wavelet Transform Binary Tree ........................... 6
4 Compression of LIGO data. 6
5 Conclusion 7
6 Acknowledgments 7
1 Introduction

LIGO is one of the most data-intensive projects. The expected total bit-rate is 15MB/s and the full 2 year LIGO data stream will yield about 1 Petabyte of data. The design of the data reduction procedures, that produce scientific data sets, is one of the important tasks of the LIGO data analysis.

There are four levels of the data, ranging from the full interferometer data (Level 0) to whitened GW strain data (Level 3, 1/1000 of the full data stream) [1]. The full data stream will be available for about 16 hours after acquisition, but will not be archived. The data will be processed to form the Archived Reduced Data Set (Level 1) that will be about 1/10 of the full data stream. At this two stages of analysis it’s important to save all useful data with minimal losses. Thus, fast and efficient lossless data compression can be essential for generation of the Archived Reduced Data Set.

In this paper we discuss lossless compression of the LIGO data with wavelets. The LIGO data is mainly Gaussian noise with admixture of non-Gaussian signals. The idea of using wavelets is to decompose data into components that can be fairly well described as a white Gaussian noise. In other words, wavelets are used to decorrelate the data, that means the representation of data in wavelet domain is more compact then the original representation.

We present a method for lossless data compression based on the lifting wavelet transform that maps integers to integers. The wavelet transform works in combination with the rdc (stands for random data compression) encoder that is optimized for compression of random Gaussian signals.

For this analysis the engineering run data (Hanford, April, 2000) was used. The results of compression for different data channels are presented in comparison with other compression techniques.

2 Random Data Encoder.

The output of LIGO data channels is digitized with 16 bit ADCs and sampled at the rate between 1Hz - 16kHz. The typical data set is a time series where each sample is represented with a 2 bytes word, so the total length of the data set with \(N\) samples is \(2 \times N\) bytes.

First, we consider the compression of white Gaussian noise (WGN) signal with rms \(\sigma_n\). The average number of bits needed to encode such a signal is proportional to \(log_2(\sigma_n)\). Figure 2 shows how many bits \((k)\) is needed to encode data samples of the WGN signal with \(\sigma_n = 1000\). Since there is no correlation between the samples, we could estimate the minimal length of the encoded signal, \(L_{\text{min}} = \sum_i k_i\) (assuming that we need 0 bits to encode zero). It is not achievable in practice, because of the overhead due to service records that define structure of the encoded data. However, different algorithms are possible, when the encoded data length \(L\) is quite close to the \(L_{\text{min}}\). The simplest one is to divide the data on short and long samples, which are encoded using words of \(k_S\) and \(k_L\) bits long respectively. The \(k_L\) is just a maximum number of bits needed to encode samples for a given data set. The \(k_S\) can be found from minimizing of the following equation:
\[ L(k_S) = N_S(k_S) \cdot k_S + N_L(k_S) \cdot (k_L + k_o), \tag{1} \]

where, \( N_S \) and \( N_L \) is the number of short and long samples respectively \( (N = N_S + N_L) \), and \( k_o \) is the overhead constant. The contiguous sequence of short samples ended with a long sample forms a data block, so the \( k_o \), actually, is the length of the block service word, that has the information about the number of short and long samples in the block. Using this algorithm we developed a simple data encoder \( (rdc) \), that produces data with the structure shown in Figure 2. We tested the \( rdc \) encoder along with the standard one \( (gzip) \) on the WGN signals. The results are shown in Figure 2. For the WGN signals the \( rdc \) encoder gives better compression factor then \( gzip - 9 \) (best compression) encoder. Though we didn’t investigate the time efficiency of the \( rdc \) encoder accurately (currently we are running the \( rdc \) in the ROOT CINT interpreter), it is at least 5 times faster then the \( gzip - 1 \) (fastest) encoder.

3 Integer wavelet transform.

For lossless compression an invertible wavelet transform that maps integers to integers is needed. Another requirement is to find the wavelet representation of the data quickly. For this reason we use the biorthogonal lifting wavelet transform \[3, 4\]. It can map integers and it allows to switch between the original data and its wavelet representation in a time proportional to the size of the data. Below we describe the lifting wavelet transform that we have implemented for the purpose of lossless compression of the LIGO data.
3.1 Lifting Wavelet Transform

The basic idea behind lifting wavelets is that the transform can be done in three stages: *split*, *predict* and *update*. At the first stage the data $x(t)$ is split into two subsets (or layers) $x_o$ and $x_e$. We simply put odd samples in the $x_o$ layer and even samples in the $x_e$ layer. In the second stage, based on the correlation present in the original data, the $x_e$ layer is used to predict the $x_o$ layer. In practice it might not be possible to predict the odd layer exactly, however, the prediction $P(x_e)$ is likely to be close to the $x_o$. Subtracting the prediction from the $x_o$ we can get the *detail* wavelet coefficients

$$d = x_o - P(x_e),$$

that retain much less information if the prediction is reasonable. The *update* stage is used to generate the *approximation* wavelet coefficients. It’s necessary to preserve the mean of the data for the approximation layer that represents the coarse structure of the signal:

$$a = x_e + U(d), \quad \bar{a} = \bar{x}.$$

For example, for the Haar wavelet the predict and update stages are

$$P(x_e) = x_e, \quad U(d) = \frac{1}{2}d.$$  

Figure 3.1(a) shows all three stages of the lifting wavelet transform. Since the initial data set is integer, we can easily get the integer wavelet transform by modifying the predict and
update operators:

\[ P_I = \text{int}(P), \quad U_I = \text{int}(U). \]  

(5)

The transform can be done in place (using the same space allocated for the data array \(x(t)\)) and it’s easily reversible (Figure 3.1(b)). Currently we use a family of symmetric, biorthogonal wavelet transforms. It’s predict and update operators can be built using the Lagrange interpolation formula:

\[ P = \sum_i x_{ei} \cdot h_i, \quad U = \frac{1}{2} \sum_i d_i \cdot h_i, \quad h_i = \frac{\prod_j (t - t_j)}{\prod_j (t_i - t_j)}, \quad i = 1, 2, ... NP, \quad i \neq j. \]  

(6)

The coefficients \( h_i \) for the first two wavelet transforms \( NP = 2, 4 \) are respectively \((1/2, 1/2), (-1/16, 9/16, 9/16, -1/16)\).

One lifting step produces the detail and approximation coefficients and then the next step can be applied to the data in the approximation layer. Because of the specific select procedure, each next lifting step yields a detail layer which is twice shorter in length than the previous one. Thus the length of the initial data set \(x\) should be divisible by \(2^m\), where \(m\) is the number of steps.

Figure 3: Compression of white Gaussian noise with different rms. The gzip compression flag is -9.
3.2 Wavelet Transform Binary Tree

The conventional wavelet transform described above can be illustrated graphically with the wavelet decomposition tree (Fig. 3.2(a)). We can consider the detail coefficients as time series and apply the wavelet transform to decompose the detail layers further. In this case the result of decomposition will be a binary tree (Fig. 3.2(b)). If there is a complete decomposition of depth $m$, the lower layer of the tree has $2^m$ nodes, each $N/2^m$ samples long, where $N$ is the number of samples in the original data. Since each transformation step doubles the frequency resolution, the nodes represent data in equidistant frequency bands, that is different compare to the traditional transform where the wavelet layers have different frequency resolution (dyadic basis).

4 Compression of LIGO data.

We used the engineering run data collected in April 2000 to test different compression methods. We compared three encoders: gzip (the compression flag is -9), eri [2] and the rdc encoder described above. The encoders were applied to the original data in time domain (TD) and the decorrelated data. To decorrelate data we used differentiation, wavelet transform (NP=6) and wavelet binary tree transform (NP=6). The results of compression are shown in Figure 4. The combinations wavelet + rdc shows better compression ratio then the other methods. Compare
to the traditional differentiation + gzip method, the average compression ratio for 16kHz channels is better by $\sim 20\%$ and for the 2kHz channels the improvement is $\sim 30\%$. Note, that the combination of the rsd and gzip encoders (last column) doesn’t show any improvement in the compression factor.

5 Conclusion

The lossless compression of LIGO data can be a useful tool for generation of the Archived Data Set. The expected compression factor is around 2, that is essential for saving of the data storage space. We suggest to use wavelet transforms and the rdc encoder to decorrelate and compress the LIGO data. This method is fast, simple and offers better compression factor then the other methods.

6 Acknowledgments

The authors would like to thank A.Studnik for help with the data processing. This work is supported by the NSF grant PHY-9722114.
Compression ratios for 16kH data channels and different compression methods

<table>
<thead>
<tr>
<th>Channel name</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>RDC</th>
<th>RDC+</th>
</tr>
</thead>
<tbody>
<tr>
<td>H2:IOO-MC_F</td>
<td>1.48</td>
<td>2.30</td>
<td>1.55</td>
<td>1.88</td>
<td>2.06</td>
<td>2.33</td>
<td>1.87</td>
<td>1.98</td>
<td>2.40</td>
<td>2.35</td>
<td>2.36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:IOO-MC_I</td>
<td>1.41</td>
<td>1.68</td>
<td>1.75</td>
<td>1.38</td>
<td>1.63</td>
<td>1.65</td>
<td>1.39</td>
<td>1.61</td>
<td>1.75</td>
<td>1.78</td>
<td>1.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:PSL-FSS_FAST_F</td>
<td>3.34</td>
<td>6.30</td>
<td>1.92</td>
<td>5.06</td>
<td>6.35</td>
<td>5.98</td>
<td>4.66</td>
<td>5.58</td>
<td>5.27</td>
<td>4.64</td>
<td>4.69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:PSL-FSS_MIXERM_F</td>
<td>1.22</td>
<td>1.35</td>
<td>1.34</td>
<td>1.24</td>
<td>1.35</td>
<td>1.40</td>
<td>1.24</td>
<td>1.35</td>
<td>1.42</td>
<td>1.44</td>
<td>1.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:PSL-FSS_PCDrive_F</td>
<td>1.20</td>
<td>1.36</td>
<td>1.33</td>
<td>1.21</td>
<td>1.36</td>
<td>1.35</td>
<td>1.22</td>
<td>1.29</td>
<td>1.40</td>
<td>1.42</td>
<td>1.43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:PSL-ISS_ISERR_F</td>
<td>3.05</td>
<td>4.04</td>
<td>3.08</td>
<td>3.17</td>
<td>4.04</td>
<td>3.86</td>
<td>3.10</td>
<td>3.98</td>
<td>3.91</td>
<td>3.72</td>
<td>3.76</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:LSC-AS_Q_TEMP</td>
<td>2.33</td>
<td>4.35</td>
<td>2.08</td>
<td>3.34</td>
<td>3.57</td>
<td>3.82</td>
<td>3.41</td>
<td>4.25</td>
<td>4.31</td>
<td>4.05</td>
<td>4.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:LSC-AS_I_TEMP</td>
<td>1.13</td>
<td>1.24</td>
<td>1.22</td>
<td>1.16</td>
<td>1.26</td>
<td>1.27</td>
<td>1.17</td>
<td>1.29</td>
<td>1.30</td>
<td>1.43</td>
<td>1.44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:LSC-AS_DC_TEMP</td>
<td>3.64</td>
<td>6.01</td>
<td>2.46</td>
<td>4.96</td>
<td>6.23</td>
<td>6.00</td>
<td>4.61</td>
<td>5.68</td>
<td>5.48</td>
<td>4.81</td>
<td>4.86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average ratio</td>
<td>1.72</td>
<td>2.19</td>
<td>1.71</td>
<td>1.89</td>
<td>2.13</td>
<td>2.18</td>
<td>1.88</td>
<td>2.11</td>
<td>2.23</td>
<td>2.24</td>
<td>2.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compression ratios for 2kH data channels and different compression methods

<table>
<thead>
<tr>
<th>Channel name</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>gzip</th>
<th>ERI</th>
<th>RDC</th>
<th>RDC</th>
<th>RDC+</th>
</tr>
</thead>
<tbody>
<tr>
<td>H0:PEM-BSC7_ACCX</td>
<td>1.60</td>
<td>1.90</td>
<td>2.00</td>
<td>1.47</td>
<td>1.70</td>
<td>1.83</td>
<td>1.52</td>
<td>1.71</td>
<td>2.05</td>
<td>2.12</td>
<td>2.14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H0:PEM-BSC5_ACCZ</td>
<td>1.84</td>
<td>2.34</td>
<td>2.28</td>
<td>1.71</td>
<td>1.97</td>
<td>2.20</td>
<td>1.77</td>
<td>1.88</td>
<td>2.41</td>
<td>2.46</td>
<td>2.48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H0:PEM-BSC7_ACCZ</td>
<td>1.73</td>
<td>2.21</td>
<td>2.21</td>
<td>1.66</td>
<td>1.87</td>
<td>2.18</td>
<td>1.71</td>
<td>1.86</td>
<td>2.27</td>
<td>2.27</td>
<td>2.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:ASC-ETMX_P</td>
<td>1.12</td>
<td>1.17</td>
<td>1.21</td>
<td>1.08</td>
<td>1.09</td>
<td>1.13</td>
<td>1.11</td>
<td>1.12</td>
<td>1.22</td>
<td>1.23</td>
<td>1.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:ASC-BS_P</td>
<td>1.17</td>
<td>1.27</td>
<td>1.30</td>
<td>1.12</td>
<td>1.17</td>
<td>1.22</td>
<td>1.15</td>
<td>1.20</td>
<td>1.31</td>
<td>1.29</td>
<td>1.30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H0:PEM-HAM7_ACCX</td>
<td>1.64</td>
<td>2.20</td>
<td>2.06</td>
<td>1.48</td>
<td>1.86</td>
<td>1.92</td>
<td>1.56</td>
<td>1.77</td>
<td>2.14</td>
<td>2.24</td>
<td>2.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H0:PEM-BSC5_ACCY</td>
<td>1.72</td>
<td>2.15</td>
<td>2.20</td>
<td>1.54</td>
<td>1.74</td>
<td>1.98</td>
<td>1.61</td>
<td>1.77</td>
<td>2.17</td>
<td>2.27</td>
<td>2.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H2:SUS-EMTX_SENSOR_UL</td>
<td>1.87</td>
<td>2.09</td>
<td>2.40</td>
<td>1.67</td>
<td>1.94</td>
<td>2.17</td>
<td>1.78</td>
<td>1.88</td>
<td>2.47</td>
<td>2.50</td>
<td>2.52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average ratio</td>
<td>1.53</td>
<td>1.80</td>
<td>1.84</td>
<td>1.43</td>
<td>1.59</td>
<td>1.72</td>
<td>1.48</td>
<td>1.59</td>
<td>1.88</td>
<td>1.91</td>
<td>1.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: Compression ratio for different LIGO channels.
References


