Lossless Data Compression for ALICE HLT

Authors:

J. Wagner, V. Lindenstruth, M. Richter, T. Steinbeck, J. Thäder

Abstract

This report introduces the High Level Trigger (HLT) component for lossless ALICE time projection chamber read out (ALTRO) data compression which is mainly based on the algorithm by D.A. Huffman [Huf52]. It is demonstrated that the direct input of the 10 bit ADC values from the ALTRO data blocks reaches comparable compression ratios to converting the outcome of the ALTRO chip into an 8 bit format before compressing. In order to guarantee a code for every ADC value, the principle of the original algorithm was slightly modified. Furthermore, the code is run time optimised for online processing, having already proved its HLT suitability in a real configuration with 2 Hz running speed in the TPC commissioning in September 2007.
Lossless Data Compression for ALICE HLT

J. Wagner, V. Lindenstruth, M. Richter, T. Steinbeck, J. Thäder
Kirchhoff Institute of Physics, University of Heidelberg, Germany

August 6, 2008

Abstract

This report introduces the High Level Trigger (HLT) component for lossless ALICE time projection chamber read out (ALTRO) data compression which is mainly based on the algorithm by D.A. Huffman [Huf52]. It is demonstrated that the direct input of the 10 bit ADC values from the ALTRO data blocks reaches comparable compression ratios to converting the outcome of the ALTRO chip into an 8 bit format before compressing. In order to guarantee a code for every ADC value, the principle of the original algorithm was slightly modified. Furthermore, the code is run time optimised for online processing, having already proved its HLT suitability in a real configuration with 2 Hz running speed in the TPC commissioning in September 2007.

1 Introduction

Since the huge amount of data coming from the ALICE detector is stored on tapes for later detailed analysis, some kind of data compression is useful to reduce the needed amount of disk space. Among all possible data compression techniques, lossless methods are preferred because they guarantee the precise reconstruction of the compressed data while using less space to store this information. Taking into account that each year 1 PB of new data is collected on tapes that cost about 1 million CHF, the capability of data compression is decisively of financial interest. For example, writing data to mass storage at a rate of 1.25 GB per second, a 1 PB tape is filled in less than 10 days. Requiring compression algorithms which provide codes with minimal redundancy and never exceed the original data size, one settles for entropy encoders. The Huffman coding algorithm finally excels, since it is the fastest, stable one with the least fragility to errors. It can also be implemented in hardware directly, bringing another velocity gain.

To motivate that the Huffman encoder indeed is an optimal technique for lossless data compression in ALICE, section 2 gives a short overview of other possibilities of lossless data compression and their results. As this report presumes the Huffman algorithm known, section 3 only focuses on the main differences and implementational adaptations to meet the requirements to use Huffman’s algorithm for ALICE
HLT. The ALTRO data [ALT02] is entered in units of 10 bits, i.e. no quasi-lossless conversion is implemented as it used to be applied in [Ber02]. Furthermore, the way a Huffman code table is created and the changes on the compression and decompression procedures are explained. Section 3 then closes with the discussion of the programming details which provide the run time acceleration. The subsequent section 4 covers the topic of experimental results encompassing compression ratios of cosmic data, simulated proton-proton data, and recent background noise measurements. Afterward, a short summary and planned future tests form the epilogue.

2 Motivation

In the paper written by H. Beker and M. Schindler [Bek96], the fundamental principles of lossless compression were discussed, which were finally implemented and tested in [Ber02]. The data compression is based on the ADC values coming out of the read out cards of the detector and being stored in the ALTRO data format. To adjust these ADC values to the pad response function of the detector, a logarithmic transformation from 10 to 8 bit values was performed. By means of this, the same relative uncertainty for all ADC values could be obtained. Thus, the implemented Huffman compressor worked on 8 bit data, resulting in best compression ratios of approximately 0.496 when the compressed output size of an event is considered with respect to the original uncompressed event size. Contrary to this, the Huffman compressor presented here directly encodes the original 10 bit ADC values, yielding comparable compression ratios without applying any kind of lossy conversion.

In order to motivate the implementation of a 10 bit based Huffman compressor although there are already compression programs like gzip available, the compression ratios obtained with gzip are investigated as well and compared to the results of the Huffman compressor.

2.1 Lossless data compression with gzip

Gzip [Gzip] uses the so called DEFLATE algorithm to compress the incoming data stream. DEFLATE is a combination of a Lempel-Ziv compression [LZ77] and a Huffman encoder. As gzip can operate on any kind of data, searching equal bit sequences in the input stream and compressing them, the fact that the ALTRO data format bases on 10 bit ADC values is left aside here.

The following figure 1 shows the results obtained by applying the gzip program to the detector data link (DDL) files that are produced by the ALICE time projection chamber read out electronics. For each link to the detector, an ALTRO data file is created that contains the recorded data from one event. Neighbouring bundles of DDLs are merged to a larger read out partition, further on called patch. To be able to compare the gzip compression ratios with the ones from the entropy encoder shown later, the DDL-files are classified in simulated zero suppressed proton-proton data (left side) and non zero suppressed real cosmic data (right side). As can
be observed, the compression ratios (i.e. considering the output size with respect to the original input size) on the left side differ much from the ones for the non zero suppressed cosmic data which reach down to 0.33. This could have been expected beforehand because, as the name indicates, the latter data is not zero suppressed, which means that long sequences of baselines are still contained in these ALTRO files, which is a good starting point for any kind of Lempel-Ziv compression techniques. Comparing the output data size after the compression with the size of cosmic data with zero suppression, it can be observed that the latter one is in the range of a few percent of the size of the non zero suppressed cosmic data. Contrary to that, the results for the simulated proton-proton data show worse compression ratios, ranging from 0.51 to 0.57, since these data files do not contain long sequences of zeros anymore due to the zero suppression. In this case, the Huffman encoder will achieve better results (as will be shown in section 4).

On the whole, since only a limited processing time is granted in the HLT and the header and trailer information of each ALTRO file must be accessible at the end of the compression procedure, gzip is not useful for fast online processing, although it achieves a remarkable reduction in file sizes for both data types.

2.2 ALTRO adapted lossless compression
Respecting the result from the previous section any lossless data compression must at least be adapted to the requirements of the data format taking header and trailer information into account. As some code developing time is needed anyway, one rather settles for entropy encoders instead of Lempel-Ziv methods for the reasons described in the introduction.
The only question which is still left open is how much knowledge about the data format (apart from header and trailer lengths) is needed. As entropy encoders totally base on statistics, one might argue that it is of minor importance how many bits of the input stream form one ensemble to be encoded. For example, one could artificially split the ALTRO data blocks into 8 bit ensembles, again omitting that every 10 bits form one ensemble, namely one ADC value. That this argument does not hold can be seen in figure 2 for the example of one non zero suppressed cosmic data DDL file. While the left graph with the 8 bit sampling depicts a broad exponential distribution of the input values, the graph on the right side shows that the distribution for the 10 bit sampling decreases faster, leading to lower entropies and therefore to better compression ratios because the lower the mean number of bits needed for encoding, the better the compression ratio.

![Histograms showing comparison between 8 and 10 bit sampling](image)

Figure 2: Comparison between the 8 bit sampling of ALTRO data (left side), leading to a high entropy and a respectively high compression ratio and the data adapted to 10 bit based input (right side) with a much smaller entropy and a better compression ratio. The example shown here was created out of one DDL-file containing several ALTRO blocks with non zero suppressed cosmic data.

The final conclusion can then be drawn that even statistical encoders should adapt their reading of the input at least to the right number of bits per sample coming from the source.

3 Implementational details

3.1 Changes in Huffman’s algorithm

Assuming the 10 bit wise reading with solely copying the header and trailer information discussed above, one can then start to build a translation table from the ADC values into their Huffman codes by means of the proper Huffman algorithm.
Usually, the binary tree for the code construction is built of those values with non-zero occurrence only. The so created Huffman code table is then used to compress the same data which was used before to create the code. For an online run in ALICE, this method would be too time consuming. Therefore, the Huffman code table consisting of the ADC values, the codes themselves, and the valid code lengths is set up prior to the actual data run, saved into a ROOT file and serves as a look up table in the compression and decompression processes. Together with the translation table, the occurrences of the ADC values are also saved to disk in a separate array, since, in some cases, it might be useful to continue filling an existing occurrence table and then create the code. This set of data contains an increased amount of events and therefore diminishes the statistical deviations in the compression process.

For the ALICE time projection chamber (TPC), tests on the non zero suppressed cosmic data and on the simulated zero suppressed proton-proton data revealed that the entropies of the different DDL-files vary less than 3\% for each patch, averaged over all sectors\(^1\) and analysed events (see figure 3). This means all data coming from

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Root mean square deviations from average entropy over all sectors and analysed events for simulated proton-proton data (left side) and cosmic data (right side). The analysis was carried out with a total of 7200 DDL-files in the proton-proton case and 1200 files for the cosmics}
\end{figure}

the 216 DDLs can be compressed by means of six different look up tables, created from several other events in a ’training phase’ prior to the run\(^2\).

In order to assure stability, one has to evade the case, when it comes to compress an ADC value which did not occur during the training phase and could not be assigned a code. Therefore, the original Huffman algorithm is changed such that values with

\footnote{A sector is called a unit of 5 patches as described in \cite{Tpc00}.}

\footnote{This principle gives the best compression ratios, while one code table for all DDL-files would suffice with slightly worse results. But due to the small table sizes (ca. 10kB), six tables can easily be afforded.}
zero occurrence are treated as any other sample and taken into the binary tree. Since their occurrence is zero, they do not affect the normal code construction and are just appended at the very end of the tree. This leads to long (not optimal) codes for those data, but having created a representative Huffman code table, the case should appear very rarely and is better than the abortion of the entire process, anyway. In the case of the 1200 DDL-files of non zero suppressed cosmic data, this problem has not occurred at all, neither for the 7200 DDL-files of proton-proton data so far.

A different way out of this problem could have been to initialise every ADC value with occurrence one. In this case, there are no ADC values with zero occurrence, thus all values are taken into account in the binary tree. Yet, this procedure leads to a non-optimal binary tree, since the ones are summed up in the creation of the tree, leading to shorter codes for these values than they actually deserve and lengthening the others at the same time. In the former case, initialising the ADC values to zero and then create the binary tree, the summation of entries with zero occurrence will again lead to a zero occurrence, such that the binary tree for the non-zero values is not affected at all.

3.2 Run time optimisation

In addition to the changes in principle, a special programming technique is used to accelerate the performance of the C++ code. Hardware oriented, all arithmetic operations are translated into simple logic instructions, whenever possible. For example, the time consuming multiplication of an integer value by a factor of two is represented by a simple left shift of all bits by one digit.

Furthermore, the occurrence table of the ADC values and the Huffman code table packed together in a ROOT file are of such a small size (ca. 10kB) that they fit into the Level 1 Cache of the analysing PCs, guaranteeing a very fast processing of the table look ups. Although the standard Linux gzip program yields better compression ratios for the cosmic data, this method bases on table look ups with large table sizes, slowing down the look up speed. This is another reason why the Huffman encoder excels over gzip here.

The optimisations described above play the most important role, as they decisively contribute to speed up the online compression. The function to create the Huffman code table is programmed in the same hardware oriented way, in order to grant a fast creation of the code tables, too. Additionally, the sorting of the occurrence table according to the frequency of occurrence of the ADC values is done by merge sort because, with a complexity of the order of \( n \cdot \log (n) \), it is the fastest and stablest sorting for any case of incoming list.

The merge sort function can be reused to sort the entries in the Huffman code table. As they are saved in the OCDB, the entries are sorted by increasing ADC values for a fast table look up while compressing. Yet, this sorting is inefficient for decompression, since then the codes are searched in the table in order to find the corresponding
ADC value. Resorting the table entries by increasing valid code lengths, the decoding speed increases over a factor of 25 because the most frequent codes (with the shortest code lengths) are now at the beginning of the search for the matching code in the Huffman table.

3.3 Usage in HLT and AliROOT

Since all processing components developed for HLT can also run offline within AliROOT, it suffices to describe the usage in the online framework. The conversion into AliROOT syntax can then be performed as described in [Ric07].

The main functionality of the Huffman compression is shown in the UML-diagram fig. 4. For the sake of simplicity, details of minor importance are omitted. The Huffman code table creation, compression and decompression are executed within one class, namely the HuffmanAltro class, respective instances are created by setting the proper componentids as described below.

The consimilar compression and decompression are executed within one component, while the code table creation is performed separately in a calibration component because this step is usually done prior to any (de)compression.

After its creation, the code table together with the occurrence table of the ADC values is saved in an instance of the HuffmanData class, which is then converted into a ROOT file and transferred to the offline common data base or preliminarily written to disk.

As can be seen in figure 4, the calibration component takes responsibility for the creation of a Huffman code table. Called *COMP*HuffmanTrainer as componentid, the HuffmanAltro class processes the incoming data to calculate the respective code from the occurrence table. The latter ones will be admitted to the offline common data base (OCDB), but can also be written to a ROOT file for debugging purposes.

By means of the componentids *COMP*HuffmanCompressor and *COMP*HuffmanDecompressor HuffmanAltro class instances for compressing and decompressing processes are created to encode or decode data streams which are treated by the HuffmanAltroComponent.

The HuffmanData class contains the Huffman code table as well as the occurrence table of the 10 bit ALTRO values and is written to a ROOT file as mentioned before. As the table handling (reading the ROOT file, extracting the Huffman code table out of it and preparing it for processing) is completely executed within the components, the user only has to know which table is needed for which detector partition.

In order to specify the name of the ROOT file (or the OCDB entry) to be initiated in the calibration, one needs the command line arguments

```
-origin, -runnumber, -dataspec, -tablepath, -trailerwords
```

The first, second and fourth arguments are transient. The first two can later be set in the course of the information interchange with the experiment control system (ECS) but currently one should specify the detector origin and the run number, while the path to which the ROOT file is written is superfluous when using the OCDB. The
Figure 4: Unified modeling language (UML) diagram summarising the main Huffman compressor architecture
specification contained in -dataspec is identical to the common meaning of dataspec describing the DDL specification:

\[0x<\text{last sector}><\text{first sector}><\text{last patch}><\text{first patch}>\]

Creating for example a HuffmanData ROOT file for the 4th TPC patch for all sectors from 0x23 to 0x00, -dataspec is supposed to be followed by the hexadecimal code 0x23000404.

Furthermore, the number of RCU trailer words (normally ranged from 1 to 3) is also needed as input.

Thus, an example for a command line creating a ROOT file for the 4th TPC patch in runnumber 2 to be written to /test/ out of data with one trailer word looks like

\[-origin TPC -runnumber 2 -dataspec 0x23000404 -tablepath /test/ -trailerwords 1\]

The same command line can be used by the component that is to compress and decompress data during runs. In this case, the arguments are taken to read in the respective Huffman code table which then serves as lookup table for the incoming 10 bit ALTRO data.

Concerning the data types of the data streams, one should choose DDL_RAW if the incoming stream consists of data in the ALTRO format and ENC_HUFF as the data type of Huffman compressed data.

Taken all this information together, the following examples show the usage within the HLT SimpleComponentWrapper (SCW):

- The following command loads a code table from run 2 for TPC patch 4 for compressing a TPC DDL file called test.ddl from sector 0 patch 0 containing one trailer word. The compressed output is written to the file compressedtest.ddl.

  ```bash
  SimpleComponentWrapper -alicehlt -V 0x3F
  -datatype DDL_RAW TPC -dataspec 0x00000000
  -infile test.ddl -outfile compressedtest.ddl
  -componentid COMPHuffmanCompressor
  -componentlibrary libAliHLTComp.so
  -componentargs "-origin TPC -runnumber 2 -dataspec 0x23000404
  -tablepath /test/ -trailerwords 1"
  -eventiterations 1
  ```

- Having compressed test.ddl into compressedtest.ddl, the decompression command for the same configuration reads:

  ```bash
  SimpleComponentWrapper -alicehlt -V 0x3F
  -datatype ENC_HUFF TPC -dataspec 0x00000000
  -infile compressedtest.ddl -outfile decompressedtest.ddl
  -componentid COMPHuffmanDecompressor
  -componentlibrary libAliHLTComp.so
  -componentargs "-origin TPC -runnumber 2 -dataspec 0x23000404
  -tablepath /test/ -trailerwords 1"
  -eventiterations 1
  ```
If no Huffman code table is available, a training run has to be set up with the following command. Giving more than one input file to the SCW, the creation of the code table takes all of them into account by gathering all occurrences in one occurrence table before constructing the code out of it.

```
SimpleComponentWrapper -alicehlt -V 0x3F
-datatype DDL_RAW TPC -dataspec 0x00000000
-infile test.ddl
-componentid COMPHuffmanTrainer
-componentlibrary libAliHLTComp.so
-componentargs "-origin TPC -runnumber 2 -dataspec 0x23000404
-tablepath /test/ -trailerwords 1"
-eventiterations 1
```

In a xml-configuration for the task manager of a real pub-sub-chain, the arguments remain the same, but additionally, the user has to specify the memory usage. As this depends on the incoming data, the following example demonstrates the general syntax how to include the Huffman components with the specifications given in the example above. Identifier of processes as shown in the first line of the xml-configuration below refer to TPC sector 0 patch 0. Thus, the parent processes for the Huffman components are either File- (FP) or read out receiver card (RORC) publishers (RP) from this partition (e.g. FP_0_0).

- Inclusion of a Huffman compressor works by adding a process like

```
<Proc ID="HC_0_0" type="prc">
  <Cmd>
    AliRootWrapperSubscriber
      -componentid COMPHuffmanCompressor
      -datatype DDL_RAW
      -dataspec 0x00000000
      -dataorigin TPC
      -componentlibrary libAliHLTComp.so
      -componentargs "-origin TPC -runnumber 2
                    -dataspec 0x23000404
                    -tablepath /test/ -trailerwords 1"
  </Cmd>
  <Parent>FP_0_0</Parent>
  <Node>0</Node>
  <Shm blocksize=(XX) blockcount=(XX) type="sysv"/>
</Proc>
```

- Analogously, the decompression can be included replacing the COMPHuffmanCompressor in the third line by COMPHuffmanDecompressor. The datatype of the incoming stream in the fourth line also has to be changed to ENC_HUFF.

- Since, except for the componentid, the specifications are also valid for the component that creates the code table, it can be also integrated in the same way as the compression processes but with the identifier COMPHuffmanTrainer.
4 Application results

Presented are the analysis of non zero suppressed cosmic data, simulated zero suppressed proton-proton data and non zero suppressed noise measurements. While the first two results focus on the compression ratios achievable for future TPC runs, the last item is dedicated to the timing aspect, showing the online performance within the HLT framework.

- **Non zero suppressed cosmic data**
  The lowest boundary for the compression ratio is set by the entropy divided by ten, omitting that the header and trailer information are not considered in this calculation. Testing the difference between the actual compression ratio and the lowest boundary for each single sector and patch (i.e. for each DDL) averaging over all events, a maximal deviation of smaller than 0.005 is observed for the compression ratios of about 0.54. This can be regarded as a good result, being able to compete with arithmetic encoding techniques (see following table of results).

<table>
<thead>
<tr>
<th>DDL with min. deviation</th>
<th>compr. ratio ± RMS</th>
<th>lowest boundary ± RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5447 ± 0.0010</td>
<td>0.5418 ± 0.0011</td>
</tr>
<tr>
<td>DDL with max. deviation</td>
<td>0.5488 ± 0.0011</td>
<td>0.5439 ± 0.0011</td>
</tr>
</tbody>
</table>

Table 1: Experimental results for non zero suppressed cosmic data, averaged over all events for each DDL. Shown are the compression ratios for the DDLs with maximal and minimal deviations from their respective lowest boundary.

- **Zero suppressed proton-proton data**
  Doing the same analysis for the proton-proton data, slightly worse results are achieved with a maximal deviation below 0.012 from the lowest boundary (shown in the next table). Nevertheless, this is still a very good compression gain, taking into consideration that the header and trailer information carry more weight due to smaller DDL-file sizes compared to the cosmos.

<table>
<thead>
<tr>
<th>DDL with min. deviation</th>
<th>compr. ratio ± RMS</th>
<th>lowest boundary ± RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5078 ± 0.0524</td>
<td>0.5030 ± 0.0169</td>
</tr>
<tr>
<td>DDL with max. deviation</td>
<td>0.5256 ± 0.0290</td>
<td>0.5139 ± 0.0263</td>
</tr>
</tbody>
</table>

Table 2: Experimental results for zero suppressed proton-proton data, averaged over all events for each DDL. Shown are the compression ratios for the DDLs with maximal and minimal deviations from their respective lowest boundary.

- **Non zero suppressed noise measurements**
  Very similar in event sizes and specifications to the cosmic data are the non zero suppressed noise measurement data from the TPC commissioning in September 2007. Also similar to the cosmos are the DDL-file sizes for the baselines of about
three MB per event. But the creation of the Huffman code table for this kind of data is not the same as the one used so far. As these data are not baseline corrected\(^3\), the occurrence table built from several events establishes a rather equally distributed occurrence of ADC values ranging from ADC value 40 to 60. Therefore, the Huffman compression ratios using such a code table are worse than the theoretical compression ratios expected by entropy analysis. This can be observed in figure 5.

In order to improve the compression ratios, one can create a code table out of one single DDL-file and use that to compress all others. By means of this, one evades producing a table out of an uniform distribution and obtains compression ratios which deviate only about 1% from the theoretical expectations.

- **Timing measurements**

  In the so called HLT Simple Component Wrapper, which implements a simulation of real runs, one MB could be compressed within 220 microseconds. But this number should only be regarded as a rough upper boundary for the real case, since other processes on the analysis computer could have influenced the time measurement.

  Considering real runs, the HLT Huffman processing component could handle the data from the noise measurements going up to 2 Hz.

---

\(^3\)i.e. all values lie above a certain threshold, which is later normalised to be zero
5 Summary and outlook

The suitability for online compression within the HLT framework is probably the first point to emphasise. Furthermore, compression ratios comparable to the results gained in [Ber02] are achieved without any loss at all. The ability to surpass the Linux gzip program and compete with arithmetic encoding techniques make this implementation a powerful tool for lossless data compression, not only for TPC data but all kind of data written in the ALTRO format. Within the near future, investigations about the impact of higher multiplicities in Pb-Pb data on the compression ratio are planned, as well as further online tests above 2 Hz running speed.

6 Acknowledgments

J. Wagner would like to thank Ivan Kisel for his comments when reviewing the paper.

References

[ALT02] ALICE collaboration:
ALICE TPC Readout Chip User Manual
ALICE-EP/ED, Draft 0.2, CERN, June 2002

[Bek96] H. Beker, M. Schindler:
Data compression on zero suppressed High Energy Physics Data
ALICE-INT-1996-03, CERN, 1996

[Ber02] J. Berger et al.:
TPC data compression
Nuclear Instruments and methods in physics research, A 489 (2002), pages 406-421,
March 2002


[Huf52] D. A. Huffman:
A method for the construction of minimum-redundancy codes
Proceedings of the I.R.E., pages 1098-1102, Sept. 1952

[LZ77] A. Lempel, J. Ziv:
A Universal Algorithm for Sequential Data Compression
IEEE Transactions on Information Theory, 23(3), pp. 337 to 343, 1977

[Ric07] M. Richter:
High Level Trigger @ AliRoot simulation/ reconstruction
Talk presented during ALICE Offline Week on October 11th, 2007
http://indico.cern.ch/
materialDisplay.py?contribId=57&sessionId=9&materialId=slides&confId=7108

[Tpc00] ALICE Collaboration:
Technical Design Report of the Time Projection Chamber
ALICE TDR 7, CERN/LHCC 2000-001, CERN, 2000