Data compression on zero suppressed High Energy Physics Data

H. Beker, M. Schindler

CERN-PPE/ALICE

Dept. f. Algorithms and Programming Methodology, Technical University Vienna

Contents

1 General Remarks on compression **2**
1.1 How does compression and decompression work? 4
Simple Methods 4
Advanced Methods 5
Data specific compression algorithms 6

2 What is zero suppression? **6**

3 So how can zero suppressed data be reduced any further? **7**

4 Data transformation at the example of the TPC **9**
4.1 General considerations 9
Digitisation accuracy 9
Dynamic range 10
Channel equalisation 11
Digital and analogue filters 11
4.2 Compression modelling for the TPC 11
Lossless Waveform modelling 12

5 Where to implement data compression **13**

6 Conclusions **14**

Abstract

Future High Energy Physics experiments will produce unprecedented data volumes (up to 1 GB/s reference ALICE TP). In most cases it will be impossible to analyse these data in real time and they will have to be stored on durable mostly magnetic linear media (e.g. tapes) for later analysis. This threatens to become a major cost factor for the running of these experiments. Here we present some ideas

developed together with the Department for Algorithms and Programming on how this volume and the related can be reduced significantly. The algorithms presented are not general ones but aimed in particular to physics experiments data. Taking advantage of the knowledge of the data they are highly superior to general ones (Huffman, LZW, arithmetic coding) both in compression rate but more importantly in compression as a to keep up with the speed of modern tape drives. Above standard algorithms are, however, used after the data have been transferred in a more 'compressible' data space. These algorithms are now available in hardware notably from IBM (IBM ALDC1-xxS) with (de)compressions speeds of up to 40 MB/s which can perform this compression in real time the required transfer rates.

Data may either be compressed just before being written to tape or it might prove advantageous to compress the data already before the recording machine and before sending them on various network media in the distributed Data Acquisition System reducing also the number and performance requirements on the LDCs, the links, the GDCs and finally on the number of tape stations.

1 General Remarks on compression

Compression algorithms are divided in two major classes, lossy and lossless ones. The first category is usable where it is permissible to loose quality in transmission and storage while retaining the general contents of the data such as in picture, movie and sound transmission or storage (e.g. jpg, mpg, various FFTs). The kind of permissible loss is mostly determined by physiological factors as the capabilities of the human eye, ear and brain. (frequency response). Without further discussion we want to assume that this is not permissible for experimental data and that it is required to have a bit per bit identical copy of the original data. It should, however, be interesting to investigate which loss would be permissible on physics data and create according models. Often this will lead to a partial event reconstruction. It is likely that a profound knowledge of both the detectors and the studied physics is necessary for the modelling process and hence physicists should look into this field rather than computer scientists as myself.

The lossless algorithms use the fact that certain data symbols on the input data are more frequent than others. Variable length codes are assigned to the input data with shorter codes being assigned to more frequent symbols or symbol sequences and longer codes to rare symbols. We propose some of these algorithm. We refer to LZW, Huffman, Arithmetic coding.

The best know code using a similar method is the Morse code which assigns "short" to the frequent letter "e" but "short short short long" to the infrequent symbol ";". It is an intuitive even if imperfect example.

Without going into details here we just want to reiterate Shannon's

theorem. He defines the information contents of a message in the following way:

Given a message which is made up of N symbols in total containing n different symbols the information contents measured in bits of the message is the following

$I = N \sum_{i=1}^{n} -p_i log_2(p_i)$

where p_i is the occurrence probability of symbol i.

What is regarded a symbol depends on the application, it might be an ASCII code, 16 or 32 bit words, words in a text etc.

A practical illustration is the following:

Assume you measure a charge or any other quantity using an 8-bit digitiser. Very often the quantity you measure will be distributed approximately exponentially. Let's assume that the mean value of your distribution is one tenth of the dynamic range i.e. 25.6.

Each value between [0..255] is regarded a symbol. Applying above formula with $n = 256, p_i = \frac{e^{\frac{i+0.5}{25.6}}}{25.6}$ you obtain a mean information contents of 6.11 bits per measured value which is almost 25 % less than the 8 bits you need saving the data as a sequence of bytes. Even if you had increased the dynamic range by a factor of for using a ten bit ADC it turns out that the mean information contents expressed as the number of bits per measurement would have been virtually the same and hence the possible compression gain even higher (39 %). This might be surprising but considering that an exponential distribution delivers a value beyond ten times it mean only every $e^{10} = 22.026$ samples it is clear that even a quite long code for such measurements cannot have an appreciable influence on the on compression rates. Considering that with all likelihood in a realistic architecture you would have had to expand the 10 bit to 16 you gain is impressive 62 % in the latter case.

The exponential distribution is a good approximation of the raw data in many cases and in particular for the distribution of the time charge buckets measured in the TPC (before clustering). Above information contents does not depend very strongly on the exact form of the distribution but rather only on the RMS.

Comparing various probability distributions it seems that the exponential distribution is particularly hard to compress. For instance a discrete spectrum being distributed according to a Gaussian with the same RMS as above exponential only has an information contents of 4.75 bits. A bounded Landau distribution corresponding to the charges sampled in the silicon strips of the MSD of WA97 presenting the same width as above exponential gives a mean information contents of 5.7 bits per strip.

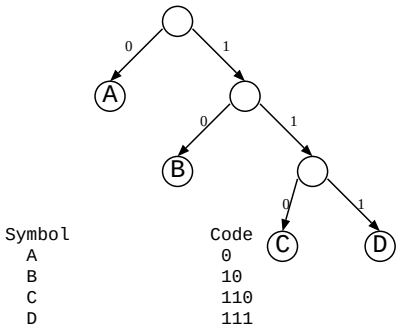

Figure 1: Huffman Code Tree

1.1 How does compression and decompression work?

Simple Methods

So far we have only used Shannon's theorem to establish what the information contents of our data set is without wondering how to practically achieve this coding. The formula indicates that we must find a code for every data word with the code length being about equal to $log_2 \frac{1}{p_i}$.

Apart from the obvious restriction that two different symbols must have different codes we also need to ensure the Huffman condition which states that no code must be equal to the beginning sequence of zeroes and ones of another code, as otherwise the decoding would get ambiguous.

The best way to achieve this is by imagining the symbol to code table as a tree as seen in Fig 1

Each node is connected with either exactly two underlying nodes or none. A node with no further connections is called a leaf and associated with a symbol of the data to be compressed.

Encoding proceeds as follows:

1. Read a symbol
2. look for the node associated with the presented symbol
3. find a (there is only one) way from the root to the leave. When ever you pass to the left output 0 when you pass to the right output 1
4. go to 1

Decoding is even simpler:

1. Goto the root
2. read 1 bit

3. if the bit is one go to the left else to the right branch
4. if the present node is a leaf output the associated symbol and go to 1. else go to 2.

The fact that symbols are associated only with leaves assures the Huffman condition. It is also relatively simple to produce the optimum code tree.

1. count the frequency of all symbols in the input stream.
2. Sort all symbols by their frequencies. Make leaf nodes for each occurring symbol. Mark the frequency for every node.
3. Create an intermediate node which points to the two nodes with the least frequency. The frequency of the new node is the sum of the frequency of the nodes it points to.
4. Take out the two least frequent node of the list. Add the new node to the list.
5. resort the list
6. Goto 3 until there are at least two nodes left in your list.

Without proof we state that the produced code compresses messages very close to there Shannon information contents in almost all practical cases. It only fails to perform well if the most frequent symbol occurs significantly more frequent than in 50 % of the cases. In this case the Shannon criterium would require to assign the symbol a code of less than one bit length which is evidently impossible. This occurs only on data which would compress extremely well (¿¿80% compression factor on byte symbols).

A more advanced algorithm (arithmetic coding) calculates in a very intelligent way the probability of every possible message and assigns a binary bit stream code whose length is inverse proportional to the probability of a given symbol stream. The probability is calculated as the product of the probabilities of each symbol.This algorithm renders compression rates which are practically identical to the the information contents even in extreme cases.

Advanced Methods

Above algorithms in typical applications render compression rates of rarely more than 50% most often 30%. The compression speed can be pushed up to about 10 MB/s on today's high end microprocessors.

Their basic defect is that they only consider single symbols and not their order. In many cases it is highly advantageous to also consider sequences of data symbols. For instance in compressing this text one gains a lot in compression by introducing meta symbols being made up of symbol sequences such as 'the', 'compression', 'symbol' and so forth.

The well known Lempel/Ziv/Welch algorithm autonomously searches for frequent substrings and Huffman codes them. It achieves compression rates which are typically higher than 60%. One most modern high end microprocessor the implementation inside the popular program *gzip* achieves a throughput of up to 2 MB/s in compression.

Data specific compression algorithms

In general terms above algorithms proceed in the following three steps.

1. create the statistics for symbols and symbol sequences (first pass through data)
2. Create an optimal but at least good symbol-¿code table Output this table to the output data stream.
3. Encode the data according to this table (second pass thorough the data)

In most cases step 2) is the most processor time consuming one. In many cases and notably in experimental data the statistical properties of the data might be a priori known allowing to immediately perform step 3) on precalculated symbol code tables.

General algorithms have a basic disadvantage in our application:

They make no assumption on the structure of their input data. They perform at time tedious searches for features such as symbol sequences which might not be present at all. They don't know what to regard a symbol (1-bit, byte, word, long word etc) and usually assume bytes; in our case data will more often than not be 32 bit quantities.

As will be shown in many cases it will be possible to transform the data bijectively into a space where they are less evenly distributed and hence compress much better. In the following we will show two examples.

2 What is zero suppression?

Today's experiments are made up of an enormous number of sensitive channels (10 9 in ALICE). All these channels should be recorded for each "event". With up to thousands of events occurring inside the detector each second one immediately ends up with completely unreasonable amounts of data.

Therefore in most cases the analogue front end electronics and connected read out busses record only data above a certain significance level.

Instead of retrieving all data values in a stream alternating sequences of address and measurement are presented to the readout bus systems. This method is of course not lossless but detector designers always have a very clear idea on how much above zero a signal must be as to be distinguishable from the inherent detector noise.

In many cases only a few percent at times even only a few per mille of all connected channels are effected in a given event, therefore enormous reduction factors can be achieved in zero time, as the alogrithms is executed in hardware and in parallel over all detector channels.

However, the resulting output very often shows little statistical correlation in particular in the address part or the correlation is hard to detect by standard algorithms as data and addresses often are divided in compound data words (16 or 32 bits) and the limit often does not coincide with a byte boundary a byte being the smallest code unit considered by most standard algorithms. Therefore compression algorithms, - either pure software as gzip, or hardware assisted compressors in tape stations, - in general perform quite poorly on zero suppressed data. (no more then 10 -30 % in most experiments)

A minor and sometimes not so minor inconvenience is that in events which have many data channels firing the amount of data can actually increase as the addresses are added to the data. Such events occur frequently during the testing and calibration phase of the detector when one has to lower the significance levels (thresholds) used in zero suppression.

3 So how can zero suppressed data be reduced any further?

To do so it is first advisable to split the measurement values from the added address. The data values by themselves often follow statistical distributions which can be exploited in variable length codes assigned to them. Often it is necessary to perform the statistics on non standard symbols spaces such as 12 bits corresponding to the resolution of the sensors (e.g. ADCs). In this chapter instead we present an algorithm which significantly reduces the amount of data constituted by the addresses coming from zero suppression by a simple transformation

As to keep the following considerations simpler we will make the example of a detector which produces only addresses and no measurement values as such, the Silicon Pixel Detector (reference Pixel). To remind you: it is made up of a piece of silicon divided into microscopic pixels (200x50 microns each).

The front end electronics will produce (refer Chesi, Darbo) only a list of pixel numbers through which a particle has passed (16 bit addresses for a so called ladder 6x60mm made up of 64 k pixels). The binary data value is implicit and zero suppression lossless; if an address does not show up in the stream it means that the associated data value is zero.

In ALICE the occupancy (the average percentage of effected pixels) will in all cases be less than 2 %. It is immediately evident that at more than 6 % occupancy zero suppression is counter productive because instead of saving a 16 bit word for every hit pixel one might as well just save all the 64 Kbits = 8kBytes as a stream of zeros and ones with

a fixed event size.

Given the lower occupancy, the envisaged front end electronics instead produces the before mentioned pixel address list. We do not expect major inhomogeneities of occupancies neither on the single event level (only a minor clustering effect) or over many events, that is to say that the hit frequency of each pixel will be close to uniform over a single ladders. The distribution of addresses will be uniform between 0 and 65535.

Therefore standard compression methods render absolutely no success. The only redundancy or useless information in the data is the order of the pixel numbers. It is therefore licit to sort the pixel addresses (either rising or falling). In actual fact one does not even have to sort them as the on detector electronics renders them in sequence by design.

This useless information can be squeezed out of the data, by instead of saving

sequence 1:

[address1] [address2] .....[address n]

sequence 2:

[address1] [address2-address1] [address3-address2] .....

It is immediately clear that sequence 1 is perfectly reconstructible from sequence 2) in all cases when the subtraction is performed in the two complements space.

While sequence 1 has a uniform distribution of symbols the symbols in sequence sequence 2 are distributed approximately exponentially with $p(i) = \frac{e^{\frac{-i}{A}}}{A}$ $WithA = \frac{1}{occupancy}$

We have tried the algorithm on generated events pixel events. We chose the typical ALICE values of nr_pixels = 65535 and occupancy = 2%.

It is clear that an optimum Huffman Code on the transformed data will render compression. With above parameters the compression factor was indeed 42 %.

With physical events it is likely that there will be some clustering. Indeed in 20 % of the cases when a pixel is hit also the pixel with an address difference of 32 will be hit due to the geometry of the detector. This will produce a peak in the address difference spectrum at 32. The compressor will automatically assign a shorter code to the value 32 without explicitly using this information in the algorithm.

Due to this effect and other correlations in the data we are confident that an compression of up to 50 % can be achieved in the pixel data volume. With higher occupancies the yield will even be bigger as the exponential distribution will get narrower.

The algorithm is simple enough as to be implemented directly on the detector in hardware saving not only tape cost but also link band width. The algorithm uses no information peculiar to the pixel detector

and should be easily extendible to the address part of the data of other detectors in ALICE.

4 Data transformation at the example of the TPC

Presently we assume that the data from the TPC will be zerosupressed. As the charge time buckets are highly clustered it is not necessary to apply an address to every word but the following data structure should suffice:

charge of bucket 1 (10 bit linear or 8 bit logarithmic)

.

.

charge of bucket “

# of first time slot in cluster (requires 10 bits=[0..1023])

# of buckets in cluster (6 bits=[1..64])

next cluster

It is reasonable to assume the structure is back linked as this is easier to achieve in pipelined hardware and presents no problem to the decoding software.

Assuming a mean data volume of 35MB/event, about 12.000 tracks in the acceptance of the TPC, logarithmic 8 bit coding and that a track crossing one of the 75 planes on average effects 4-5 neighbouring pads in this plane the mean cluster length should be about:

$\frac{510^7}{1610^3 * 3 * 70} - 2 = 5.5$

Additional 2 bytes are needed for encoding cluster start and length.

As show in Fig. 2, a cluster can be defined in several different ways: either a single threshold determines whether a time charge bucket is inside a cluster; it is possible and advantageous to implement a hysteratical behaviour which decreases the threshold once a time charge bucket exceeds the threshold; an number of samples before and after a relatively high threshold crossing might be added to the cluster etc. etc. The latter two refinements avoid that slight oscillations just before or more often after the actual pulse produce a separate cluster with the connected overhead. Before adding a cluster to the output stream a global threshold can be applied to the cluster charge suppressing low amplitude oscillations The definition of the cluster will have a significant influence on the data rate but not so much on the implementation of compression algorithms.

4.1 General considerations

Digitisation accuracy

Whereas in non compressing mode accuracy is irrelevant as long as a measurement fits in a given byte/word/long unit, excessive accuracy

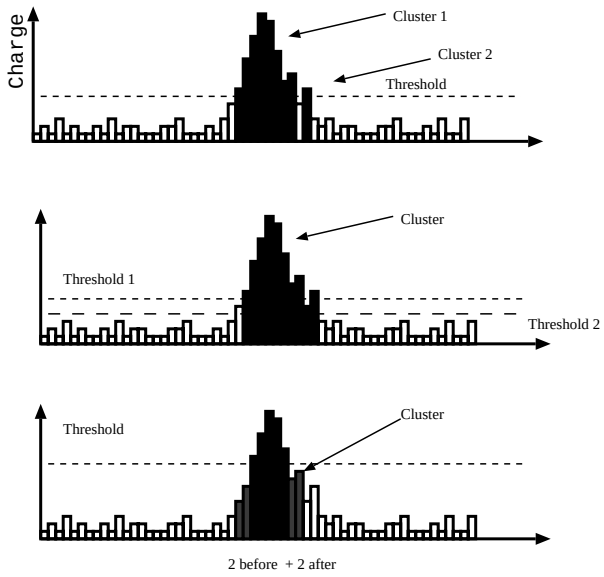

Figure 2: Cluster definition

is a cost factor in compressing mode. Measuring the width of the distribution of the pedestal can easily increase the produced amount of data by a significant fraction.

In general the digitisation accuracy should be chosen so that the RMS of the sampled pedestal is no wider than 1 digitiser count. In this case the digitisation noise introduced is $\frac{1}{\sqrt{12}}$ When added in quadrature to the electronics noise ($\sqrt{1+\frac{1}{12}} = 1+\frac{1}{24}$ this increases the RMS by the totally insignificant amount of 4%. Even when the accuracy is further reduced by a factor of two, leading to an apparent pedestal RMS of 0.5 digitiser counts the digitisation noise contributes only to about 16 % of the total apparent noise, while one bit per measurement is saved in this way.

Dynamic range

In non-compressing mode the width of the dynamic range often sensitively determines the net amount of data. In most cases going from 8 to 9 or 10 bit dynamic range will mean to double the data rate (you will have to pass anyway to 16 bit data words).

As stated already in the initial example this is *not* the case in compressing mode: If, as almost invariably is the case, the frequency of measurements at the upper limit of the dynamic range is low, the length of the Huffman code assigned to these values will be very long

but on the other hand this long codes will be used very rarely and not contribute significantly to the average compression rate.

Once you have decided to compress your data it makes very little difference whether your ADC has an 8 or 10 bit range. Therefore the argument for logarithmic coding is much weaker when using compression.

Channel equalisation

In non compressing mode the equalisation of channels in gain and more importantly in pedestal can be performed off-line at very little cost. When compressing data which come from different electronics channels it is, however, highly preferable to do this on the front end as not to widen the spectra of the measured quantities which immediately results in worse compression rate. The alternative of having a separate code table modulated to the exact spectrum for every channel is prohibitive for a number of reasons: decoding and encoding complexity, the bandwidth used for transmitting the code tables can become important etc. etc.

Digital and analogue filters

While above considerations will be applicable to most detectors the one on frequency response is peculiar to the TPC data.

The time frequency domain of a valid charge cluster will be relatively narrow somewhere around 1-10 MHz. It is evident that a high pass will remove base line shifts and low frequency pickups (50 Hz and harmonic) and actually improve the data quality. It will also prevent a long duration positive base line shift to produce extremely long fake clusters as well as loosing detection efficiency with negative base line shifts.

Low pass filters will remove the noise from high frequency emitters on the detector and 'on the air'. Also the method presented in the following will highly benefit in compression rate after high frequency noise has been eliminated.

The filter can either be implemented analogically or digitally using a DSP or a custom filter ASIC. In the digital case it is very easy to adjust the transfer function not only to a given gas and high voltage but it can be adjusted dynamically over the drift time and take into account that distant (from the cathode pads) clusters will present lower frequencies. A digital solution should be able to keep up with the 10 MHz clock rate.

4.2 Compression modelling for the TPC

Based on the assumptions in our initial example it is reasonable to assume 25% data reduction coming from a naive Huffman Coding of

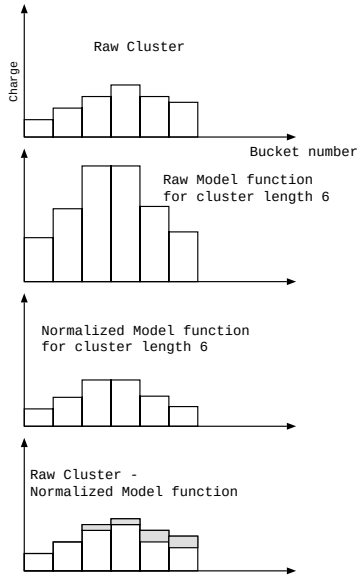

Figure 3: Lossless Waveform modelling

the time charge buckets. The cluster addresses and length might be encoded using their own specific code trees. The incremental address storing approach shown for the pixel detector might also reduce the amount of data coming from addresses.

We suggest to try the algorithms on experimental data (coming from NA49) and compare them to the results of standard compression programs such as gzip. It is not necessary to implement the specific algorithms in detail but the Shannon information contents applied to the spectra can give a quick idea on attainable compression rates you can trust that arithmetic coding will yield a practically identical compression rate.

To obtain higher compression rates it will be necessary to develop detector specific models. In the following we present an first idea and invite to experiment it on realistic TPC data.

Lossless Waveform modelling

In Fig. 3 We propose to change the cluster data format presented previously in the following way:

Instead of saving only cluster address, length and the time buckets as they are, it should be advantageous to add the integral charge of the cluster or the mean bucket contents (which is the same except for a multiplicative factor) to the data stream.

Instead of saving the time charge buckets it is better to save the the

residuals of the time charge buckets with respect to a model function which is know to the encoder and decoder (and hence not part of the data stream). One only needs one model function for each possible cluster length as the detector response will be in first approximation linear and the function can be scaled to the total cluster charge stored in the cluster.

Even though scaling involves (integer) divisions, rounding errors are not relevant as the encoder and decoder will commit the same rounding errors and the residuals will reproduce the bit by bit identical reproduction of the original data.

These model functions can either be derived from analytical considerations or by just measuring the mean normalised wave form of uncompressed data for every given cluster length. The residuals should in any case be distributed much narrower than the raw measurements and hence yield much better compression and more than compensate the added cluster integral. This will be even more so if high frequency noise is removed by filters as explained above.

Evidently clusters coming from the overlap of close track will not compress very well but they should be sufficiently rare as not to contribute to a significant fraction of the total data volume.

It is even conceivable that the residuals form typical sequences which can be encoded more efficiently by methods such as LZW than mere Huffmann or arithmetic coding.

Statements on the compression efficiency can be made only after trying it on realistic data but there is the distinct hope to reduce the TPC data by at least 50%.

5 Where to implement data compression

Data compression can be implemented directly in front of the tape station and probably not inside the tape station as commercial devices will not be re-programmable for our special data features. While this has a number of architectural advantages (e.g. data never have to be decompressed for intermediate analysis, code trees are kept in central points etc.) it does not reduce the requirements on the number and performance of links and hence not the number and performance of the GDC and LDCs.

Therefore data should be reduced as close as possible to the detector or inside the LDC digitiser cards. As the step of proposed data modelling schemes are very experiment and detector specific they should be handled in software or in re-programmable hardware. The final step of encoding (and even compression code generation) can be handled by general purpose chips such as the IBM ALDC1-40s which perform a combination of LZW and arithmetic coding in real time maybe just in front of the outgoing data link from the LDC to the GDC via the switch. A software emulation of this chip exists which allows apriori

measurements of achievable compression rates.

6 Conclusions

Even though efficient online data compression increases the complexity of the experiment and require a not totally insignificant development effort, the potential cost gain justifies this. The cost of most components of the general DAQ (excluding detector front end and detector link receivers) scales linearly with the achieved compression factors. Even the robotics used for off-line analysis scales with the same factor.

Maybe more importantly the magnetic media might constitute an important contribution to the experimental running cost. Even under the probably optimistic assumption that in the year 2004 it will be possible to buy linear magnetic media at a price of 1 Sfr / GByte, 1 month of uncompressed running at a rate of of 2.5 GB/s will cost about 5 MSfr.

The additional cost of encoding and decoding computing power will be totally negligible compared to these sums.