Fast and Efficient Entropy Compression of ALICE Data using ANS Coding

Michael Lettrich[1,*], for the ALICE collaboration

[1]CERN, Technische Universität München

Abstract. In LHC Run 3, the upgraded ALICE detector will record 50 kHz Pb-Pb collisions using continuous readout. The resulting stream of raw data to be inspected increases to ~1 TB/s - a hundredfold increase over Run 2 - must be processed with a set of lossy and lossless compression and data reduction techniques to decrease the data rate to storage to ~90 GB/s without affecting the physics.
This contribution focuses on lossless entropy coding for ALICE Run 3 data which is the final component in the compression stage. We analyze data from the ALICE TPC and point out the challenges imposed by the non-standard data with a patchy distribution and symbol sizes of up to 25 Bit. We then explain why rANS, a variant of Asymmetric Numeral System coders is suitable for compressing this data effectively. Finally we present first compression performance numbers and bandwidth measurements obtained from a prototype implementation and give an outlook for future developments.

1 Introduction

ALICE (A Large Ion Collider Experiment) [1] is a heavy-ion collision detector at the LHC (Large Hadron Collider) [2] at CERN, built to study the physics of strongly interacting matter. Throughout the Long Shutdown 2 (LS2) of the LHC, the ALICE detector will receive a substantial upgrade [3]. The upgraded detector will record Pb-Pb collisions at a rate of 50 kHz using continuous readout. This is necessary to cope with the increased Pb-Pb interaction rate and thus exploit the full scientific potential of the LHC after LS2. As a direct consequence the volume of data to be inspected increases by a factor 100. In order to decrease the raw data rate of ~3.5 TB/s to the targeted storage rate of ~90 GB/s, a sequence of highly effective compression and data reduction steps are applied by the new ALICE Online-Offline (O^2) software [4]. Most of these steps perform lossy compression based on replacing raw data with results of reconstruction methods such as track finding, clusterization and pattern recognition without affecting physics.

In this paper we will focus on entropy coding, the final stage in the compression chain where input data is transformed in a lossless and reversible manner that reduces the required space in permanent storage [5]. After briefly introducing general concepts and notation, we will discuss the challenges imposed by our data. This will be followed by a concise summary of our literature search and explain why we believe that rANS - a variant of Asymmetric Numeral System coders suits our needs best. We then will present compression performance

[*]e-mail: michael.lettrich@cern.ch

numbers and bandwidth measurements obtained from a prototype implementation to back the theory and give an outlook for future developments.

2 Variable Range Entropy Coders

In order to compress data, an entropy coder reduces the redundancy contained in the source data without altering the information content. This is generally achieved by interpreting the source data as a message consisting of a concatenation of symbols s_i from a finite alphabet $\mathcal{A}$ of length n (e.g. the Latin alphabet or the signals provided by a sensor). Counting the frequency of each symbol f_i with $\sum f_i = M$, where M is the length of the message, we can determine the probability of each symbol $Pr[s_i] = \frac{f_i}{M}$ occurring in the message. It is then possible to construct a coding function C which will transform the source symbol into a representation in a (binary) variable length coding alphabet $\mathcal{X}$ where highly probable source symbols are assigned a shorter length code x_i than less probable symbols [5]. The best know example for this is Huffman Coding [6].

The lower bound of bits needed to represent a message is called Shannon entropy [7], defined as

$$H = -\sum_{i=1}^{n} Pr[s_i]log_2(Pr[s_i]) \tag{1}$$

The closer the length of the coded message is to the entropy of the source message, the more efficient is the coder. The difference is called redundancy R.

3 Description of the Environment and Source Data

The raw detector data of the ALICE detector coming from continuous readout will be split into so called time frames of 20 ms and processed inside an Event Processing Unit (EPN), a heterogeneous CPU and GPU server within 30 s. The resulting, compressed time frame (CTF) is then entropy coded on the same EPN within 35 s before going to storage. The data passed to the entropy coder are flat structures of arrays containing unsigned integer values - i.e. the arrays can can be interpreted as columns of a table. Each row of this table then describes an object [4].

For the sake of simplicity we restrict ourselves to the analysis of the source data from the ALICE Time Projection Chamber (TPC) as with an expected overall contribution of over 92% , it will account for the majority of recorded data [4]. Furthermore the principles can be transferred easily to the other sub-detectors.

The TPC data is a structure of 23 arrays of unsigned integers that describe 5 different objects of which only 4 will be relevant to us. Each array has a specified range of values between 8-25 Bits. To make assumptions on how well this source data can be compressed we look at a sample dataset of $\mathcal{O}(10^7)$ objects from the processing of 130 Pb-Pb interactions simulated under LHC Run 3 conditions - the biggest dataset available to us at that point. Creating per array frequency tables and plotting the resulting symbol distributions, leads to the following observations:

- The sample data does not follow any common probability distribution.
- In most cases the distribution is patchy, i.e. only a fraction of the values from the value set are taken, despite the $\mathcal{O}(10^7)$ samples.

To rule out insufficient statistics or simulation parameters as a root cause for these observations, cross checks with actual LHC run 2 TPC data have been conducted. The results show significant similarity for almost all parameters and deviations can be explained by the

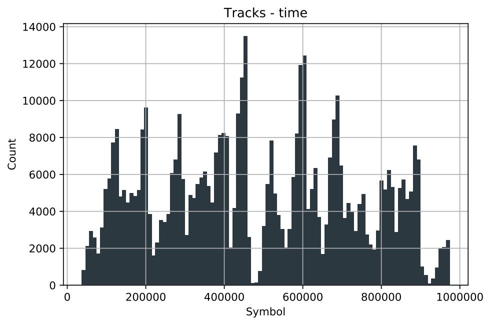

Figure 1: 482419 symbol samples from Run 3 simulation containing 344987 unique values in a 24 Bit symbol range for object `Tracks`, array `time`. Entropy of the sample data is 18.24 Bit. Visualized with 100 bins.

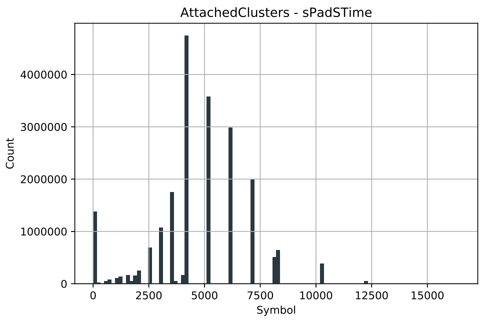

Figure 2: 2072849 symbol samples from Run 3 simulation containing 419 unique values in a 16 Bit symbol range for object `AttachedCluster`, array `sPadSTime`. Entropy of sample data is 6.06 Bit. Visualized with 100 bins.

Objects	Count	Fields	Bit/obj	H [Bit/obj]	Hconcat [Bit/obj]
AttachedClusters	21072849	4	41	17.15	15.75
AttachedClustersRed	20590430	4	55	17.60	17.59
Tracks	482419	5	73	53.90	53.90
UnattachedClusters	50745911	5	81	39.77	38.37

Table 1: Description of Sample Dataset. Entropy H of the samples can be reduced by concatenation of correlated fields.

changed running conditions between Run 2 and Run 3. Furthermore experiences from previous runs show that the distributions of source symbols stay rather static and only have to be recalculated if run parameters change. As we can see in figures 1 and 2, not all values within the symbol ranges have occurred neither in our sample dataset nor the run 2 data we compared it to despite the large amount of samples. Nevertheless all values in the specified range are possible and need to be accepted by the entropy encoder.

As can be expected from looking at the distributions, the calculated entropy (see table 1) shows a high redundancy and indicates, that data can be compressed by a factor 2-3 by an optimal compression algorithm. A further reduction of the entropy is possible by identifying correlated arrays and concatenating the individual entries i.e. $C[i] = A[i] \circ B[i]$.

A suitable entropy coding algorithm thus should be able to efficiently compress source data with non-standard, patchy distribution of up to 25 Bit per symbol and is able to process data in the foreseen time window of 35 s. Since each array can be individually encoded, we can parallelize over the arrays. An algorithm that can additionally make use of SIMD vectorization or even the GPGPUs of the heterogeneous EPNs is preferable.

4 Entropy Coding Algorithms using rANS

There are different types of entropy coding algorithms available. Huffman coding - one of the most widely used entropy coding algorithms - constructs a prefix-free binary code c_i for each source symbol s_i. It can be shown that this c_i is the shortest code with an integral amount of bits that can be uniquely decoded [6]. Furthermore both coding and decoding can be implemented very efficiently without the need for costly arithmetic operations. The drawback of this algorithm however is that it is only optimal if $-log_2(Pr[s_i]) \in \mathbb{N}$. Information theory tells us that the shortest code for a source symbol $-log_2(Pr[s_i]) \in \mathbb{R}$ is real valued though and thus would require "fractional bits". This leads to an inefficiency of Huffman coding of up to one Bit per symbol.

This inefficiency is overcome by Arithmetic coding [8], where the coding function $C : \mathcal{A} \mapsto [0, 1)$ encodes all M symbols of the message to a single, real valued number x in the [0,1) interval. The gain in efficiency comes at a cost in complexity though: to keep x within the bounds of finite precision floating point arithmetics, renormalization steps are needed and Arithmetic coding - as the name suggests - requires arithmetic operations which will naturally result in slower compression/ decompression compared to Huffman coding. Finally the legal situation in the past has been difficult since parts of the algorithm were protected by patents.

A new family of variable range entropy coders called Asymmetric Numeral Systems (ANS), has been introduced [9] [10]. Comparable to arithmetic coding the coding function $C : \mathcal{A} \mapsto \mathbb{N}$ encodes all M symbols of the message into a single state variable x which in the case of ANS is a positive integer. This state variable grows with the probability of the encoded symbol, i.e. $x_{i+1} \approx x_i/Pr[s_i]$. The coding function $C(x_i, s_i)$ by construction has the property that the decoding function $D(x)$ inverts the coder such that $D(C(x_i, s_i)) = (x_i, s_i)$. This property requires that the coder operates in LiFo order, i.e. it runs backwards from the last to the first source symbol such that the decoder can output the symbols from first to last again.

Similarly to arithmetic coding, renormalization is needed as otherwise x_i will grow beyond the bounds of what can be efficiently handled by a computer. For ANS, the state variable is kept within a range I and bits are streamed out if the upper limit is surpassed during encoding and streams in when the lower limit is surpassed during decoding.

There are two variants for ANS that have gained interest in the community: rANS (range ANS) and tANS (table ANS). tANS works with a coding table and constructs a finite state automaton that works without costly arithmetic operations. It is favourable for small alphabets and can even be implemented efficiently in hardware [11]. Since it delivers almost the speed of Huffman Coding at the compression of arithmetic coding, tANS is being used in more and more applications. rANS on the other hand works more like arithmetic coding and uses a simple, easy to implement arithmetic formula that can be applied on large alphabets. The advantage rANS has over arithmetic coding is that its decoding function exactly reverts the step done by its coding function. This concept opens the door to performance optimizations such as interleaved encoders, SIMD vectorization and GPGPU [12] or functional additions such as handling incompressible data.

Based on literature research, rANS was therefore chosen as a candidate to replace the Huffman entropy coder used in Run 2 [13] for Run 3 and will undergo further evaluation. The necessity of this step becomes evident if we consider that the yearly volume of compressed timeframes is in the range of 40 Pb [4]. Even improvements in compression rate in the range of one percent will in total save hundreds of TB of storage space and bandwidth.

5 Benchmarking rANS on ALICE TPC Data

To understand if the strong points rANS has shown in the literature research translate into practice, we tested a prototype implementation of rANS on the ALICE TPC data discussed in section 3. Starting from the ryg_rans rANS implementation in C++ [12] [14] and re-engineering it to suit our needs, our testing and benchmarking scenario (depicted in figure 3)

Objects	H [Bit/obj]	rANS [Bit/obj]	rANS/H	BW [MiB/s]
AttachedClusters	15.75	15.75	1.00	649.70
AttachedClustersRed	17.59	17.64	1.00	776.06
Tracks	53.90	53.90	1.00	365.56
UnattachedClusters	38.37	38.37	1.00	589.80

Table 2: Performance of rANS entropy coding for the ALICE TPC sample data.

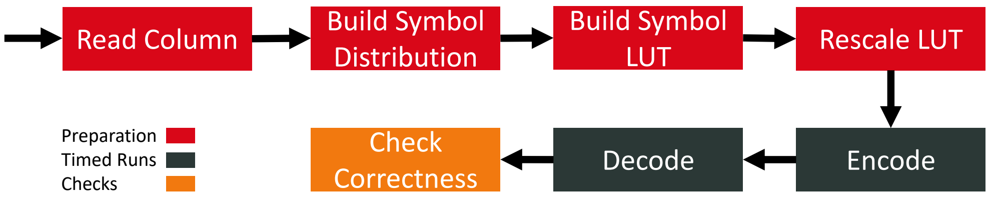

Figure 3: Testbed for benchmarking rANS on the ALICE TPC sample data.

is as follows: The source array from the flat structure produced by the previous data reduction steps and its metadata is read in and a fixed-size encode buffer is allocated. A first pass over the data determines the distribution of source symbols followed by a rescaling of the later such that the total count of symbols is a power of two. This allows bit shifts instead of costly arithmetic operations in the encoder/ decoder. These symbol statistics then are used to build the actual lookup tables (LUT) needed for encoding and decoding. Once the setup phase is completed, a timed encoding run is performed and the actual size of the encoded data is measured followed by a timed decoding run. Finally, the decoded date is compared with the source data to ensure correct output. For both encoding and decoding, the bandwidth is then calculated as an average over 5 runs.

The test system is a Ubuntu 18.04 LTS Linux desktop PC with a six core Intel Core i7-8600k, 32 GB RAM and turbo mode enabled. Both rANS coder and decoder are executed with a two fold interleaving on a single thread to exploit CPU pipelining but without SIMD vectorization. The performance of the entropy coding and the bandwidth for the sample ALICE TPC source data (table 1) is shown in table 2. We can clearly see that rANS is capable to compress the source data close to entropy limit at a very high encoding bandwidth. Extrapolations show that the data contained within a Pb-Pb timeframe can easily be processed by the entropy coder within the given time envelope of 35 s compressing data by an overal factor between 2 and 3. Since a simulated and reconstructed ALICE timeframe under Run 3 conditions was not available at the time of writing, we cannot give numbers. The extrapolation however is reasonable as the distribution symbols for the real run is assumed static across timeframes for production runs and the source data is already available in memory. Therefore the measured encoding performance is a very good indicator for the production system. For decoding of course all data has to be read into memory from external sources and prepared, so decoding performance is just one of the relevant factors.

During our benchmarks it became evident that the rescaling of the symbol statistics used to construct the encoder/ decoder LUTs has a major impact on the compression achieved in the entropy coding as well as the encode/decode bandwidth. Since no references exist for large alphabets (symbol size >8 Bit), we need to conduct a parameter study to understand which LUT precision is required for 8 Bit, 16 Bit and 25 Bit symbol alphabets. The LUT holds the frequency f_i for each source symbol s_i. For performance reasons the frequencies f_i are rescaled to $\hat{f}_i$ such that $\sum \hat{f}_i = \hat{M} = 2^p$ as this allows replacing arithmetic multiplication and division with bitshifts. Logically, $\hat{M}$ has to be at least the size of our source alphabet $\mathcal{A}$ if each symbol occurs at least once and the better the rescaled frequency table approximates the underlying symbol distribution the better the compression. This relation shows in figure 4. What was surprising is that the approximation rANS needed for good compression could already be achieved by a rather rough LUT resolution compared to the $O(10^7)$ samples provided to build the frequency table which we believe is due to the sparse structure of our data.

The precision of the LUT has only a small impact on the encoding bandwidth as can be seen in figure 5. Instead we can see, that larger symbols are encoded faster than smaller ones. This is due to the fact that the encoder is bound by the arithmetic and bitwise operations per transferred byte and scales almost linearly with larger symbols until the problem starts becoming memory bound.

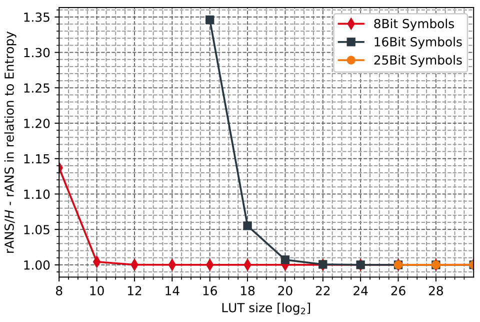

Figure 4: Relation between LUT precision and rANS encoding efficiency.

For the decoding speed we see the contrary effect as for decoding we need to construct an inverse Lookup table (iLUT) which maps the respective fraction of the cumulative frequency back to the original symbols. Up until this iLUT fits into CPU cache, the decompression bandwidth again is limited by arithmetic and bitwise operations and drops considerably for all symbol sizes once cache misses in the iLUT start occurring. It is therefore necessary to come up with a more compact datastructure for an iLUT array of size $\hat{M}$ which still has fast access times in practice. This is also important to decrease the memory footprint of decompression as an iLUT array for large alphabets is too big for practical applications.

A comparison of rANS and Huffman on the same dataset, using a Huffman coder with a per array code tree as in [13] shows that rANS indeed is able to outperform Huffman for every single encoded array. While in most cases the difference was in the order of one percent, we

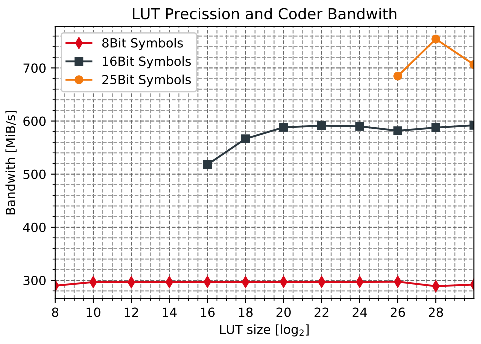

Figure 5: Effects of LUT precision on coding bandwidth.

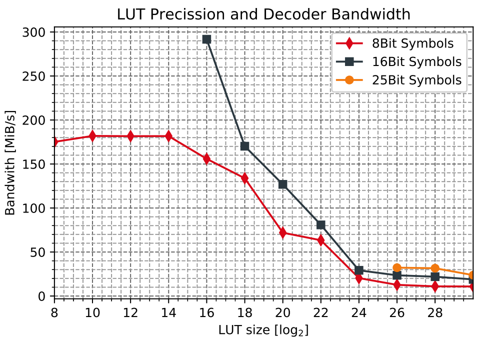

Figure 6: Effects of LUT precision on size of inverse LUT and decoding bandwidth.

could observe that with particular arrays of our dataset, rANS could outperform Huffman by 8% or even 30% due to its better approximation of the underlying probabilities. The robustness of rANS to compress all input data we provided, is very close to the entropy limit irregardless of the underlying symbol distribution is what we perceive as the biggest gain compared to Huffman coders. This allows us to guarantee that even with changing symbol distributions caused by changes in running conditions of the physical systems, the data will be compressed optimally.

6 Conclusion and Outlook

After extensive literature research and a test of a prototype implementation on simulated ALICE Run 3 TPC sample data we can conclude that rANS is a powerful entropy coding algorithm that delivers close to optimal compression at high encoding bandwidths also for large alphabets. These results encourage us to continue work on a production version of a rANS entropy coder for ALICE in LHC Run 3. To achieve this goal the code robustness has to be improved and decoding speeds need to be massively increased. Finally a full integration into ALICE O^2 is needed including full system measurements and tuning once all input data is available. Furthermore we would like to increase encoding and decoding bandwidth even further by vectorization and a GPGPU implementation of the algorithm.

References

[1] K. Aamodt et al. (ALICE), JINST **3**, S08002 (2008)
[2] L. Evans, P. Bryant, JINST **3**, S08001 (2008)
[3] B. Abelev et al. (ALICE), J. Phys. **G41**, 087001 (2014)
[4] P. Buncic, M. Krzewicki, P. Vande Vyvre, Tech. Rep. CERN-LHCC-2015-006. ALICE-TDR-019 (2015), `https://cds.cern.ch/record/2011297`
[5] D. Salomon, D. Bryant, G. Motta, *Handbook of Data Compression* (Springer London, 2010), ISBN 9781848829039
[6] D.A. Huffman, Proceedings of the IRE **40**, 1098 (1952)
[7] C.E. Shannon, The Bell System Technical Journal **27**, 379 (1948)
[8] G.G. Langdon, IBM Journal of Research and Development **28**, 135 (1984)
[9] J. Duda, *Asymmetric numeral systems* (2009), `0902.0271`
[10] J. Duda, *Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding* (2013), `1311.2540`
[11] J. Duda, G. Korcyl, *Designing dedicated data compression for physics experiments within fpga already used for data acquisition* (2015), `1511.00856`
[12] F. Giesen, *Interleaved entropy coders* (2014), `1402.3392`
[13] J. Berger, U. Frankenfeld, V. Lindenstruth, P. Plamper, D. Röhrich, E. Schäfer, M.W. Schulz], T.M. Steinbeck], R. Stock, K. Sulimma et al., Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **489**, 406 (2002)
[14] F. Giesen, *ryg_rans* (2014), `https://github.com/rygorous/ryg_rans`