Lossless preprocessing of floating point data to enhance compression^{*}

Francesco Taurone, Daniel E. Lucani, Marcell Fehér, and Qi Zhang

DIGIT, Department of Electrical and Computer Engineering, Aarhus University {francesco.taurone,daniel.lucani,sw0rdf1shqz,qz}@ece.au.dk

Abstract. Data compression algorithms typically rely on identifying repeated sequences of symbols from the original data to provide a compact representation of the same information, while maintaining the ability to recover the original data from the compressed sequence. Using data transformations prior to the compression process has the potential to enhance the compression capabilities, being lossless as long as the transformation is invertible. Floating point data presents unique challenges to generate invertible transformations with high compression potential. This paper identifies key conditions for basic operations of floating point data that guarantee lossless transformations. Then, we show four methods that make use of these observations to deliver lossless compression of real datasets, where we improve compression rates up to 40 %.

Keywords: Compression · Lossless · Floating point.

1 Introduction

Data management is a key aspect of most IoT related applications, since data generated by sensors usually has to be transmitted, stored and analyzed. A common practice to store time-series datasets is to compress them before archival, so to limit the needed database space. Before compression, data is usually preprocessed, undergoing a transformation process where the content of the dataset is adjusted for compression, which can potentially enhance the compressibility of the dataset. There are various preprocessing approaches that the user could adopt, ranging from rounding the decimals, to changing domain of the signal with the discrete wavelet transform [2], or dropping bits that carry minimal information [8]. One key distinction among preprocessing transforms is whether they are lossy or lossless. Given a transformation function f and a dataset DS, we say that f is lossless when $f^{-1}(f(DS)) = DS$, lossy otherwise. The goal of this paper is to characterize lossless operation on floating point data and to propose 4 lossless preprocessing techniques enhancing the compressibility for

^{*} This work was supported by the IoTalentum Project within the Framework of Marie Skłodowska-Curie Actions Innovative Training Networks (ITN)-European Training Networks (ETN), which is funded by the European Union Horizon 2020 Research and Innovation Program under Grant 953442.

2 F. Taurone et al.

mono-dimensional floating point datasets, meaning that the preprocessed dataset f(DS), once compressed, is smaller in size than the compressed vanilla DS. We quantify compression in terms of compression ratio (CR), defined as

$$CR = \frac{Compressed \text{ dataset size in bits} + Compression \text{ metadata}}{Uncompressed \text{ dataset size in bits}}$$
(1)

where the metadata might be needed to be able to undo the compression. Since these transformations operate on each individual floating point number, they are naturally compatible with compression methods with fine-grained random access capabilities, such as Generalized Deduplication [12], which are useful for computing analytics directly on the compressed data [6].

1.1 Compressing floating point numbers

The most common standard for floating point numbers is the IEEE-754 [1], where a number x is represented by three components: the sign (S), the exponent (E) and the mantissa (M). In this paper, we assume the use of double precision, namely 64 bits per number. When interpreted as unsigned integers, x is

$$x = (-1)^{S} \cdot 2^{E-B} \cdot (1 + M \cdot 2^{-l}), \tag{2}$$

where B = 1023 is the bias of the exponent, and l = 52 is the length of the mantissa in bits. Out of the 64 bits, 1 is the sign bit, 11 are the exponent and the remaining 52 bits are the mantissa. Each mantissa bit is m_i , with $i \in \{1, 2, \ldots, 52\}$ and m_1 being the most significant one. From Eq.(2), we see that all x such that $|x| \in [2^{E^*}, 2^{E^*+1})$ have the same exponent $E^* + B$.

As studied in [11], a possible idea to improve compressibility is to maximize the number of bits shared by all numbers in the collection we want to compress. We say that the i-th bit is *shared* when all numbers int eh dataset have the same value for the i-th bit, meaning that if all bits are shared, the dataset is composed of a series of equal numbers. The work in [11] empirically shows that having more shared bits generally results in better compression, with both standard compressors, like zlib [4], and ones based on deduplication, used in Sect. 4. However, the preprocessing technique \bar{f} showed in [11] is lossy: in the following, we elaborate on it to propose lossless preprocessing techniques.

2 Key observations

2.1 Lossless operations on floating point numbers

Numbers represented with a finite amount of bits will necessarily have finite precision. IEEE-754 can represent a subset of all numbers on the real line, while values that are not in this subset get approximated to one that is. In this context, we refer to the precision of a number $x = (-1)^{S_x} \cdot 2^{E_x - 1023} \cdot (1 + M_x \cdot 2^{-52})$ as

the distance between x and the next representable floating point number. We call this quantity *unit in the last place*, or ULP(x), calculated as

$$\mathrm{ULP}(x) = 2^{E_x - 1023 - 52} \tag{3}$$

The reason why manipulating floating point numbers can result in losses is that ULP(x) is not a constant in floating point, but rather a function of the number's unbiased exponent. Transformations whose output results in ULP(x) different from the original could potentially lead to information losses. In order to explain this phenomenon, we introduce the operations \oplus , \otimes , \ominus , \oslash , which represent addition, multiplication, subtraction and division according to IEEE-754. For our purposes, we can think of them as being the theoretical operations, followed by rounding to the nearest representable number.

An example of potential losses due to floating point precision can be shown by considering x = 3.5, the transform $f(x) = x \oplus 10^{16}$ with its inverse $g(x) = x \oplus 10^{16}$ and their theoretical counterpart $f'(x) = x + 10^{16}$ with its inverse $g'(x) = x - 10^{16}$. While $g'(f'(x)) = x, \forall x \in \mathbb{R}$, the same is not true in floating point, since $g(f(3.5)) = 4.0 \neq 3.5$.

While there are multiple strategies to limit the effects of this approximation error, as described in Section 5.3 of [10], this paper aims at devising strategies to avoid it completely. In Sect. 2.1, we discuss particular scenarios for which summation does not produce error, and in Sect. 2.1 we analyze multiplication.

Lossless addition In the scenario where $x \in [2^{E^*}, 2^{E^*+1})$ and $f(x) = A \oplus x$ is such that $f(x) \in [2^{E^*+1}, 2^{E^*+2})$ and $A \in [2^{E^*}, 2^{E^*+1})$, the operation is lossless as long as both A and x have the same least significant mantissa bit m_{52} . The reason is that during an addition under these conditions, a bit called *guard bit* has to be rounded away, since $\text{ULP}(\cdot)$ goes from $\text{ULP}(x) = 2^{E^*-52}$ to $\text{ULP}(f(x)) = 2^{E^*-51}$. However, if both addends have the same m_{52} , the guard bit is guaranteed to be equal to zero, requiring no rounding. This result is summarized in Table 1 and more details on how addition works in floating point in Section 7.3 of [10].

Table 1: Mantissa least significant bits for lossless addition $y = x \oplus A$. The blue results indicate a lossless transformation, namely $y \oplus A = x$.

m_{52}^{y}		$m_{51}^{x}m_{52}^{x}$			
		00	01	10	11
$\iota^A_{51}m^A_{52}$	00	0	0	1	0
	01	0	1	0	0
	10	1	0	0	0
й	11	0	0	0	1

Another particular useful scenario is the addition $y = x \oplus A$ where $x, y \in [2^{E^*}, 2^{E^*+1})$ and $A \in [2^{\tilde{E}}, 2^{\tilde{E}+1}) < 2^{E^*}$. It can be shown that given the floating point addition process detailed in [9], this operation is lossless when

$$m_i = 0 \quad \text{for } i \in \left\{ 52 - \left(E^* - \tilde{E} + 1 \right), \dots, 52 \right\}.$$
 (4)



Fig. 1: Representation of the loss with $f(x) = y = x \odot M$ and $f^{-1}(x) = \tilde{x} = y \oslash M$. Assuming the use of the *round to nearest* approach, all real numbers in the same color region get rounded to the same floating point number.

Lossless multiplication In the scenario where $x \in [2^{E^*}, 2^{E^*+1})$ and $f(x) = x \otimes M$ is such that $f(x) \in [2^{E^*+1}, 2^{E^*+2})$, the operation is guaranteed to be lossless as long as $M \geq 2$. To prove it, we consider Fig. 1, where $y^* = x \cdot M$, $y = x \otimes M$, $\tilde{x}^* = y/M$ and $\tilde{x} = y \oslash M$. In order for this transformation to be lossless, we need

$$|x - \tilde{x}^*| = \Delta_x < \mathrm{ULP}(x)/2 \tag{5}$$

since this would result in $\tilde{x} = x$ under the *round to nearest* rounding scheme. Since $y = \tilde{x}^* \cdot M = (x + \Delta_x) \cdot M$, we have $y - y^* = M \cdot \Delta_x$, which combined with Eq.(5), results in the condition for lossless multiplication being

$$y - y^* < M \cdot \frac{\mathrm{ULP}(x)}{2}.$$
 (6)

Since $\text{ULP}(y) = 2 \cdot \text{ULP}(x)$ and $y - y^* < \frac{\text{ULP}(y)}{2}$ because of the rounding method, we have $y - y^* < \text{ULP}(x) < M \cdot \frac{\text{ULP}(x)}{2}$, fulfilling Eq.(6) when $M \ge 2$.

3 Lossless dataset manipulations for better compression

In order to increase the number of shared mantissa bits in the dataset, our goal is to ensure that all numbers after preprocessing lie in the portion of the real line where the mantissa bits are guaranteed to be as we desire. Given a dataset DS, in order to guarantee that the D most significant mantissa bits are shared, the datapoints x would all have to be such that

$$|x| \in [2^E, 2^E + 2^{E-D}]$$
 with $E \in [-1022, 1023] \quad \forall x \in \text{DS}.$ (7)

Next, we present four techniques to losslessly place the numbers in the dataset in these preferred regions, supposing without loss of generality and for clarity sake to have a dataset where all numbers have the same exponent. This can be easily generalized by storing as metadata the information on the original exponent of each sample. We named these techniques *compact bins, multiply and shift, shift and separate even from odd, shift and save evenness.*



Fig. 2: Illustration of method *compact bins*. In this example, by using 3 bins all numbers in the new dataset are guaranteed to have $m_1 = 1$ and $m_2 = 1$, while there was no such guarantee in the original dataset.

3.1 Compact bins

In the scenario where $x \in [2^{E^*}, 2^{E^*+1})$, if we cluster this set of values into bins and shift the bins so they they are closer to each other, we are effectively reducing the range of values of the transformed dataset, possibly making it fit to a region where some mantissa bits are guaranteed to be shared. An example of this binning process is in Fig. 2, where we use 3 bins to ensure $m_1 = 1$ and $m_2 = 1 \ \forall x \in \text{DS}$. The optimal amount of bins depends on the distribution of values in the real axis. Two conditions are needed in order for this process to be lossless. First, supposing to use k bins on a dataset with ℓ unique values, we have to store $(k-1) \cdot \lceil \log_2(\ell) \rceil$ bits as metadata representing the boundaries of the bins, as well as the k values $\{A_1, \ldots, A_k\}$ used to shift the bins. The size of metadata in bytes is $Z = (k \cdot 64 + (k-1) \cdot b)/8$. Secondly, the A_i have to be chosen so that the operation $y = x \oplus A_i$ is lossless. We can use Eq.(6) to select the correct A_i values.

3.2 Multiply and shift

As per Eq.(7), there is one portion of the real axis per exponent region guaranteeing that the D most significant bits are equal to 1. Using the lossless operations in Subsect. 2.1, we can manipulate the dataset so that the modified DS has all values lying in those regions. The *multiply and shift* transform we propose, represented in Fig. 3, involves one multiplication and one addition to move to the next exponent region, applied interatively until all modified numbers lie on the portion of the real axis where the most significant D mantissa bits are guaranteed to be shared. Assuming that $x \in [2^{E^*}, 2^{E^*+1}) \quad \forall x \in DS$, the operation is

$$f(x) = (2.0 \otimes x) \oplus A$$
, with $A = 2^{E^* - D + 1} - 2 \cdot \text{ULP}\left(2^{E^* + 1}\right)$, (8)

6 F. Taurone et al.



Fig. 3: **Illustration of method** *multiply and shift*. From the original dataset in black, we iteratively apply Eq.(8) to obtain a transformed dataset in red lying on regions in green, where all numbers are guaranteed to share at least the first D mantissa bits. We can recover the original dataset with the inverse transform since all operations are lossless.

where A has to be rounded down if necessary to the first value fulfilling Eq.(4). As we can see in Fig. 3, after an iteration, only the values out of the region with desired common bits have to go through another one, progressively shaving the dataset under analysis. A limitation of this algorithm is that the multiplication by a factor of 2.0 effectively enlarges the window of values of the transformed portion of the dataset, increasing the number of needed iterations. However, M = 2 is the smallest lossless factor, as per Eq.(6). In terms of metadata, we need to store D and A_1 , since all A_i with $i \neq 1$ can be computed by knowing the exponent of the original dataset and D. In order to invert the transformation, given the transformed dataset, we can iteratively apply the inverse transformation $f^{-1}(y) = (y \ominus A) \otimes 2.0$ on each element belonging to the rightmost exponent region, concluding with the inverted shift stored as metadata.

3.3 Shift and separate even from odd

In order to alleviate the drawbacks of multiply and shift in Subsect. 3.2, we can substitute the multiplication with an addition, while carefully selecting the addendum so that it fulfills the conditions in Sect. 2.1. Specifically, given the operation $y = x \oplus A$ with $x, A \in [2^{E^*}, 2^{E^*+1})$, their mantissas M_x and M_A have to have the same evenness, namely $m_{52}^x = m_{52}^A$. We do that by ensuring that all y resulting from x with even mantissas lie in a portion of the real axis that does not overlap with the one for the odds. In this way, we can distinguish the two during the inverse transformation. As depicted in Fig. 4, the transformation differs depending on the evenness of the mantissas as input, resulting in

$$f(x) = \begin{cases} x \oplus A_i^{\text{even}} & \text{if } \mod(M_x, 2) = 0\\ x \oplus A_i^{\text{odd}} & \text{if } \mod(M_x, 2) = 1. \end{cases}$$
(9)

 A_i^{even} and A_i^{odd} at the i-th iteration are computed so to fulfill Eq.(4). In particular, at th i-th iteration, with $x, A_i^{\text{even}}, A_i^{\text{odd}} \in [2^{E^*}, 2^{E^*+1})$ and D desired shared



Fig. 4: Illustration of method *shift and separate even from odd*. We iteratively apply Eq.(9), ensuring that the output of x with even and odd mantissas lie in different portions of the real axis ath every iteration.

most significant mantissa bits, they are

$$A_i^{\text{even}} = 2^{E^* + 1} + 2^{E^* - D}, \quad A_i^{\text{odd}} = A_i^{\text{even}} - W_i, \tag{10}$$

where W_i represents the length of the portion of the dataset to be processed at the i-th iteration. It is calculated starting from the original dataset DS with $W_{i+1} = 2 \cdot W_i - 2^{E^*-D}$, where $W_0 = \max(DS) - \min(DS)$. In terms of metadata, we need to store the initial shift A_{align} , D and W_0 , since all A_i can be reconstructed from those. The i-th step of the inverse transformation of $y \in [2^{E^*}, 2^{E^*+1}]$ is

$$f^{-1}(y) = \begin{cases} y \ominus A_i^{\text{even}} & \text{if } y > 2^{E^* + 1} - W_i \\ x \ominus A_i^{\text{odd}} & \text{if } y \le 2^{E^* + 1} - W_i. \end{cases}$$
(11)

With this method we strategically choose A_i^{even} and A_i^{odd} so that we can guess the evenness of the original x by checking the position of y on the real axis. The drawback is that we need to enlarge the range of values of the transformed dataset at every iteration, similarly to what happened with the multiplication by 2.0 in Subsect. 3.2.

3.4 Shift and save evenness

In order to limit the scaling up of the transformed dataset at every iteration in *shift and separate even from odd*, we could store the evenness information as a single metadata bit per sample, meaning n bits per iteration for a dataset of n elements. The increased size of metadata is compensated by a drastic reduction in the number of iterations needed to end the procedure. The direct transformation formulas is the same as per Eq.(9), where the only difference is that the calculation of A_i^{odd} becomes $A_i^{\text{odd}} = A_i^{\text{even}}$.

8 F. Taurone et al.



Fig. 5: Illustration of method *shift and save evenness*. While similar in concept to Fig. 4, we use metadata to distinguish whether to apply the inverse transformation with A_i^{odd} , namely having odd mantissas, or with A_i^{even} , which has it even.



Fig. 6: Comparison of the best results from Fig. 7a and Fig. 7b, where lower $\delta_{\rm CR}$ are better.

4 Results

In this section we analyze the benefits of the four proposed techniques for lossless preprocessing of floating point datasets, namely *compact bins, multiply and shift, shift and separate even from odd, shift and save evenness.* In order to quantify their effectiveness, we compare the compression ratio achieved with preprocessing (CR_{PREP}) and without preprocessing ($CR_{NO-PREP}$), using the metric

$$\delta_{\rm CR} = \frac{\rm CR_{\rm PREP} - \rm CR_{\rm NO-PREP}}{\rm CR_{\rm NO-PREP}}$$
(12)

where negative values mean that our techniques resulted in better compressed datasets. To measure the impact of metadata on the final size, we define Z as

$$Z = \frac{\text{Metadata size in bytes}}{\text{Compressed dataset size in bytes}}.$$
 (13)

As compressor, we use Greedy-GD [7], since the number of common bits is particularly beneficial for its effectiveness in terms of CR. Here, D_M is the number of desired most significant guaranteed shared mantissa bits, whereas the total



Lossless preprocessing of floating point data to enhance compression

(b) UCI-gas-turbine-emissions [5].

Fig. 7: Performances of the 4 techniques for lossless preprocessing in terms of CR on the datasets Chicago-taxi-trips-fares and UCI-gas-turbine-emissions.

number of shared mantissa bits is S_M , S_E is for shared exponent bits and S_{TOT} for the total number shared bits. As input, we use the first 1000 elements of the datasets Chicago-taxi-trips-fares [3] and UCI-gas-turbine-emissions [5], having a single dimension of non-negative floating point values: these technique could be extended to mixed signed datasets. In Fig. 6, we summarize the best results of all 4 techniques with both datasets, while in Fig. 7 we report more detailed performances.

Looking at Fig. 6, we see that in all scenarios we were able to find a transformation that improved compression, since all optimal results are below zero. We also notice that the best transformation was able to achieve an improvement of 40% in CR, meaning that the final size of the compressed dataset was 40% smaller in size than the compressed vanilla version.

Regarding the *compact bins* technique, we notice that S_{TOT} is always larger than D_M . It plateaus when all 52 mantissa bits are shared, causing CR to plateaus as well. We also see that S_E is constant, since this technique preserves the original exponents. We see a plateau in S_{TOT} also with *multiply and shift* and *shift and separate even from odd*. Here, after a certain D_M value, for every additional mantissa bit we want to be shared, we lose a shared exponent bit, making S_{TOT} constant: as a consequence, CR plateaus as well.

9

10 F. Taurone et al.

We also notice that with Compact bins and shift and save evenness we can impose a much higher number of common mantissa bits compared to both multiply and shift and shift and separate even from odd, since the first two require less iterations to terminate. The trade-off is that Z increases for larger D, potentially becoming larger that the size of the compressed dataset itself.

5 Conclusions and future work

This paper identifies key conditions for floating point addition and multiplication operations to be invertible, i.e., without introducing errors. Using these observations, we proposed four techniques to preprocess arrays of floating point data in a lossless fashion to enhance their compression potential. We discussed their implementations and analyzed their application on real datasets and showed their efficacy and limitations. Our numerical results show possible compression improvements of up to 40 %, without introducing losses. In the future, we plan to expand their use to mixed sign datasets and investigate their combination.

References

- 1. Ieee 754-2019 standard for floating-point arithmetic (2019)
- 2. Batal, I., Hauskrecht, M.: A supervised time series feature extraction technique using dct and dwt. In: 2009 international conference on machine learning and applications (2009)
- 3. City of Chicago: Taxi trips dataset (2022), https://tinyurl.com/4rypurjp
- 4. Jean-loup Gailly, M.A.: Zlib compressor, https://www.zlib.net/
- 5. H. Kaya: Gas turbine co and nox emission dataset (2019), https://tinyurl.com/2ubk63ra
- 6. Hurst, A., Lucani, D.E., Assent, I., Zhang, Q.: Glean: Generalized-deduplicationenabled approximate edge analytics. IEEE Internet of Things Journal (2023)
- Hurst, A., Lucani, D.E., Zhang, Q.: GreedyGD : Enhanced generalized deduplication for direct analytics in IoT (2023), arxiv.org/abs/2304.07240
- 8. Klower, M., Razinger, M., Dominguez, J.J., Duben, P.D., Palmer, T.N.: Compressing atmospheric data into its real information content. Nature Computational Science (2021)
- 9. Mark D. Hill: Notes on floating point arithmetic, part of course cs/ece 354 at university of wisconsin, https://tinyurl.com/mvzcyc3x
- Muller, J.M., Brisebarre, N., De Dinechin, F., Jeannerod, C.P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S., et al.: Handbook of Floating-Point Arithmetic (2010)
- Taurone, F., Lucani, D.E., Fehér, M., Zhang, Q.: Change a bit to save bytes: Compression for floating point time-series data. In: IEEE ICC (2023), arxiv.org/ abs/2303.04478
- Vestergaard, R., Lucani, D.E., Zhang, Q.: A randomly accessible lossless compression scheme for time-series data. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications (2020)