LITE-QA Compression

Data Management

Lossy In-Situ Tabular Encoding for Query-Driven Analytics (LITE-QA) supports the data management by limiting the amount of data stored during the simulation phase and the amount of data loaded for the generation of visualizations. LITE-QA provides compact storage of full-resolution data grids and data indices as well as improved data access methods for visualization based on compressed contents.

For a typical CFD data set containing index and data for metal melt flow simulation, LITE-QA achieves a data reduction of three to four fold for index and data combined. For the aluminum simulations carried out with the LBM, the compression achieves a 5 fold average data reduction for stationary simulation grids, up to 12 fold data reduction for high-resolution temporal data grids and a average data reduction by factor of three for data indices while guaranteeing a maximum decompression error of 1%.

LITE-QA realizes compression and index generation based on a floating point encoding for absolute values and differences, which is optimized for a lossless compression backend and achieves high compression rates at high compression speeds (Lehmann, Werzner, and Degenkolb 2016; Lehmann and Jung 2018). The floating point compression and decompression operate directly inside the simulation code and the visualization application without noteworthy decline of the run-time as compared to using uncompressed data, therefore improving the data management (Lehmann 2018).

LITE-QA can store additional low-resolution data with 1/8 of the resolution. Storing low-resolution data uncompressed would demand for additional 12.5% storage space. LITE-QA provides access to low-resolution contents with only 3% of additional storage.

In combination with an in-situ generated index, LITE-QA improves the time-to-analysis by employing a query-driven approach for selective loading and decompression of only the data needed for the task at hand. The query-driven approach to decompression allows for efficient localization and visualization of interesting phenomena leaving large parts of the data untouched, which are not needed. The index-based localization and decompression accelerates data loading for typical visualization tasks by a factor of 16 as compared to loading uncompressed data and performing linear search (Lehmann et al. 2018).

For the compression of simulation grids and data indices, two schemes are used, which consist of grid linearization and tabular encoding (GLATE) and error-bounded binning and tabular encoding (EBATE). The compression performance competes with state-of-the art in-situ compression methods like ISABELA (Sriram Lakshminarasimhan et al. 2013), ZFP (Lindstrom 2014) and SZ (Di and Cappello 2016) as well as with in-situ indexing methods like ALACRITY (Jenkins et al. 2013), ISABELA-QA (S. Lakshminarasimhan et al. 2011) and DIRAQ (Sriram Lakshminarasimhan et al. 2014) with respect to data reduction while restricting the point-wise maximum error for decompressed contents. The temporal compression schemes t-GLATE (Lehmann 2018; Lehmann and Jung 2018) as well as the indexing sheme EBATE operate at high speeds and provide a trade off between data accuracy and compression rate based on a error bound in percent.

Linearization of Simulation Grid

In order to prepare the data for compression and index generation, the first step of the *Lossy In-Situ Tabular Encoding for Query-Driven Analytics* (LITE-QA) in-situ processing pipeline consists of applying a linearization procedure to the local simulation grids in each parallel process. The Hilbert-curve is used to construct a single polygonal chain always passing through one direct neighbor of each voxel cell without jumps as shown in Fig. 1 (1). Therefore, the spatial adjacency of grid cells is preserved to a large extent in the sequential values of the linearized data stream, which has positive effects on the resulting compression rate. The linearization, called X, is applied subsequently to all simulation variables contained in the local grid before the actual data compression is performed in parallel by each simulation process independently.

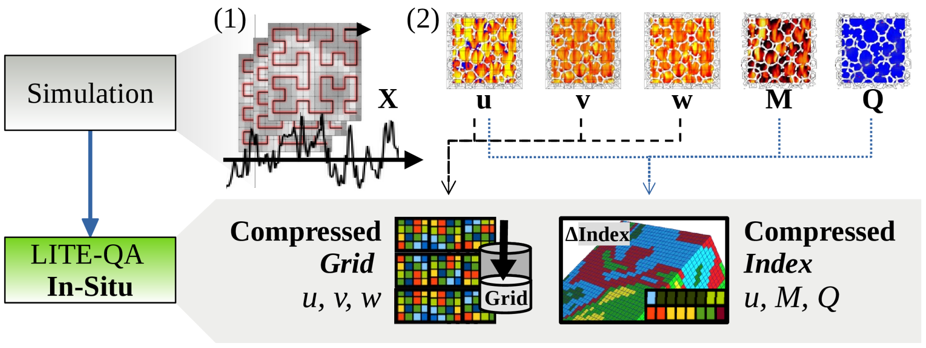

Figure 1: Linearization of voxel grid based on Hilbert-curve. (1) Local grids of simulation processes are linearized into a sequence X. (2) The smooth linear data sequences X are stored as a compressed grid for the variables u, v, w, and as compressed index for the variables u, M, Q accordingly.

For the present visualization task in the context of the virtual prototyping scenario, the data set consists of the flow field u, v, w and two additional flow field properties, i.e. velocity magnitude M and local vortex characteristics Q[1]. As shown in Fig. 1 (2), after linearization, the variables u, v, w are stored in a compressed grid, and u, M, Q are stored in a compressed index. In the experiments describes in (**Lehmann2023?**), the simulations are carried out by 64 parallel processes on a global grid with 512^3 voxels. Thus, for each grid corresponding to the variables u, v, w, and for each index corresponding to u, M, Q, each single process manages voxel linearizations X of length $l = 128^3$ for compression and index generation. However, before the actual compression and index generation takes place, the linearized data streams X are quantized for all variables as explained in the next section. Based on the quantized data, the compression and index generation is performed as explained in *LITE-QA Grid Compression* and *LITE-QA Index Compression*.

Floating Point Quantization

The second step of the in-situ processing pipeline concerns quantization of the linearized data sequences X for each simulation variable u, v, w, M, Q. The quantization of floating point values is performed on a global step function $S(n)$ based on the integer step numbers n given as $\ldots, -2, -1, 0, 1, 2, \ldots$. The function $S(n)$ restricts the quantization error in percent as a configurable point-wise maximum error of e.g. $0.01 \ldots 1\%$ for each value in the grid and the index. The step function uses an optimized data encoding based on a cyclic pattern of mantissa bits for compact representation of the step values n and differences between two quantized values $S(n') - S(n)$. The compact encoding allows for the application of standard lossless compression and bit-packing as a backend with high compression rates and high compression speeds for grid and index as well.

Cyclic Pattern of Mantissa Bits

The step function $S(n)$ is chosen such that the step size increases with respect to increasing n without violation of the error policy, i.e. maximum point-wise error of $e_\mathrm{R} \times 100$ in percent. Interestingly, particular choices of e_R imply a cyclic pattern of mantissa bits for values $S(n), S(n+\omega), S(n+2\omega), \ldots$ with cycle length ω as illustrated in Fig. 2. The condition holds for certain choices of the maximum error, e.g. 0.99%, 0.49% or 0.27% as shown in Fig. 3 (1).

Step Function Definition

The function $S(n)$ is defined using the so-called positive step function $T(n)$

[1]The velocity magnitude $M = \sqrt{u^2 + v^2 + w^2}$ and the Q-criterion Q are computed on-the-fly before compression. $Q = {}^1/_2(||\Omega||^2 - ||S||^2)$, where S and Ω are the symmetric and the anti-symmetric components of the velocity gradient tensor (Dong, Yang, and Liu 2016). They are also known as the strain rate tensor and the rotation tensor respectively. The unit for values of variable Q is omitted.

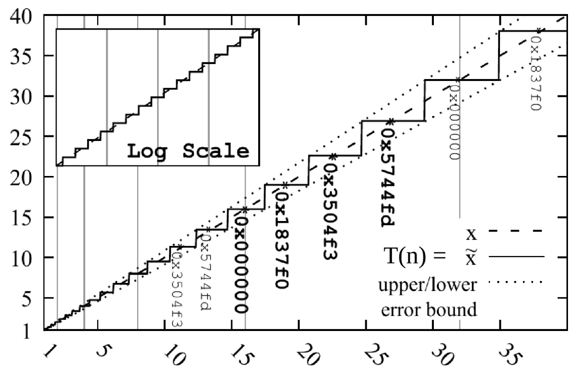

Figure 2: Illustration of the step function $S(n)$ for $n \geq 1$ with cyclic mantissa bits using a short cycle legth of $\omega = 4$ resulting in point-wise maximum error of $e_\mathrm{R} = 8.64\%$. Mantissa bits repeat for $n, n+4, n+8, \ldots$, and therefore allow for efficient compression. The log scale view illustrates the equal spacing of step function in logarithmic space.

according to the definition

$$\begin{aligned} T(n) &= x_\mathrm{Z} \cdot \left(\frac{1+e_\mathrm{R}}{1-e_\mathrm{R}}\right)^n, \\ S(n) &= \begin{cases} 0 & \text{, for } n = 0 \\ -T(|n|-1|) & \text{, for } n < 0 \\ +T(n-1) & \text{, for } n > 0. \end{cases} \end{aligned}$$

The positive step function starts at a small non-zero number $T(0) = x_\mathrm{Z}$ and uses the error bound e_R, for definition of the step width. The transition from step $T(n)$ to $T(n+1)$ occurs when the maximum quantization error of e_R is met according to $T(n) \cdot (1 + e_\mathrm{R}) = T(n+1) \cdot (1 - e_\mathrm{R})$. The function $S(n)$ extends the positive step function $T(n)$ for negative step numbers $n < 0$ and introduces a zero value as $S(0) = 0$ for all values $|x| < x_\mathrm{Z}$. The error between a value $|x| \geq x_\mathrm{Z}$ and $S(n)$ is defined in percent according to

$$|x - S(n)|/|x| \leq e_\mathrm{R} \times 100\%.$$

Given a floating point value x, the quantization is performed by mapping x to $\tilde{x} = S(n(x))$ by calculating the step number n according to

$$\begin{aligned} s(x) &= \begin{cases} 0 & \text{, for } |x| < x_\mathrm{Z} \\ \pm 1 & \text{, for } |x| \geq x_\mathrm{Z} \end{cases} \\ n(x) &= s \cdot \left\lfloor \frac{\log|x| - \log x_\mathrm{Z}}{\log(1+e_\mathrm{R}) - \log(1-e_\mathrm{R})} + \frac{3}{2} \right\rfloor, \end{aligned}$$

where s indicates the sign of x.

Cyclic Encoding with Mantissa Look-Up Table

The cyclic pattern of mantissa bits, as shown in Fig. 2, is arranged by choosing e_R and x_Z depending on ω and δ according to

$$\begin{aligned} e_\mathrm{R} &= (\sqrt[\omega]{2}-1)/(\sqrt[\omega]{2}+1) \\ x_\mathrm{Z} &= 2^{-\delta}. \end{aligned}$$

As illustrated in Fig. 3 (2), for a low resolution step function with maximum error 8.64%, the mantissa parts $m = 0, 1, 2, 3$ repeat after the cycle length $\omega = 4$ steps for $n, n+\omega, n+2\omega, \ldots$. ω and δ are specified by the user and determine the quality of the decompressed data as well as of binning during index creation. ω defines the length of one cycle of the mantissa bits and δ defines the actual values of $S(n) = \pm x_\mathrm{Z}$ for $n = \pm 1$.

(1)

ω	e_R×100%
4	8.640
35	0.990
70	0.495
128	0.271
139	0.249
347	0.099
700	0.049

(2)

n	e	m	Mantissa	
1	1	0	`0x000000`	Ω_0
2	1	1	`0x1837f0`	Ω_1
3	1	2	`0x3504f3`	Ω_2
4	1	3	`0x5744fd`	Ω_3
5	2	0	`0x000000`	Ω_0
6	2	1	`0x1837f0`	Ω_1
..	2	2	`0x3504f3`	Ω_2

Figure 3: Error bounded cyclic quantization of mantissa. (1) Cycle length ω of mantissa bits and corresponding resulting maximum error bound e_R in percent. (2) Illustration of cyclic mantissa look-up table Ω for a low resolution step function with cycle length $\omega = 4$. The exponent part e and mantissa part m are shown for values of $S(n)$ with $n \geq 1$, where n is obtained by $n = s \cdot [(e-1)\cdot\omega + m - 1]$ with $s = +1$.

Quantization and Reconstruction

The cyclic values of the mantissa bits are stored into the so-called mantissa look-up table Ω_m for $m = 0, 1, \ldots, \omega - 1$. Due to the cyclic encoding using the mantissa look-up table Ω, as shown in Fig. 3 (2), any step number $n \neq 0$ can be translated into sign $s = \pm 1$, exponent part $e > 0$ and the mantissa part $0 \leq m < \omega$ according to the following equations:

$$\begin{aligned} s &= \begin{cases} +1 & \text{, for } n > 0 \\ -1 & \text{, for } n < 0 \end{cases} \\ e &= \lceil |n|/\omega \rceil \\ m &= (|n|-1) \text{ modulo } \omega, \\ n &= s \cdot [(e-1)\cdot\omega + m - 1]\,. \end{aligned}$$

Given a linearized sequence $X = \{x_1, x_2, \ldots, x_l\}$, as explained in *LITE-QA Grid Linearization*, the quantization of each value is performed by computing the step number $n(x)$ under consideration of the step function parameters ω and

δ. Based on n, the sign s, the exponent part e and the mantissa part m are determined as described, whereas the combination $e = m = s = 0$ encodes the zero $S(n) = 0$ for values $|x| < x_Z$ with $n = 0$.

The fast implementation of the quantization procedure $n(x)$ directly determines s and e from the bits of the floating point value x using standard operations, while m is determined using fast search inside the cyclic mantissa look-up table Ω (Lehmann et al. 2018). For fast value reconstruction, $\tilde{x} = S(n)$ is computed given a triplet (e, m, s) using the following procedure:

- If $e = m = 0$, then set $\tilde{x}$ to zero,
- else, if $e > 0$, then initialize $\tilde{x}$ by
    1. setting the sign to s and the exponent to $e - \delta - 1$ and
    2. setting the mantissa bits to Ω_m.

Grid Variable Encoding

For grid data as well as temporal sequences of grids, *Lossy In-Situ Tabular Encoding for Query-Driven Analytics* (LITE-QA) employs a new compression scheme called *Grid Linearization and Tabular Encoding* (GLATE) and its temporal extension t-GLATE (Lehmann 2018; Lehmann and Jung 2018). Both allow for efficient lossy data reduction in the spatial and temporal domain of high resolution LBM simulations of metal melt.

Temporal Compression Policy

The t-GLATE temporal compression scheme differentiates between so-called key-frames and difference-frames as shown in Fig. 4. While key-frames are compressed and decompressed independently using the spatial compression policy, difference-frames are compressed with a higher efficiency based on temporal differences. The compression efficiancy is directly related to the time step size Δt of the exported data and to the difference-frame insertion rate f_d, i.e. export of every time step, every second, every fourt. $f_d = 0$ corresponds to non-temporal compression with key-frames only.

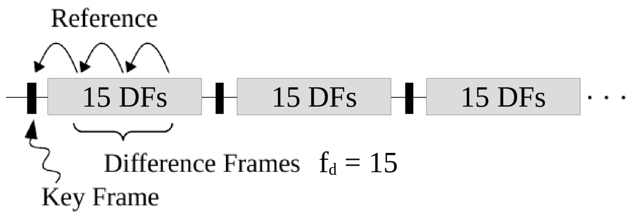

Figure 4: Data referencing during export of temporal data with $f_d = 15$ difference-frames following each key-frame. Key-frames are decompressed independently, whereas difference-frames reference the previous frame.

Spatial and Temporal Difference Policy

As explained in *LITE-QA Grid Linearization*, when using the Hilbert-curve for linearization, the spatial coherence between voxels in the three dimensional simulation grids is conserved to a large extent in the neighboring data values. For export of temporal data with high-resolution step width, neighbors in time exhibit strong temporal coherence, i.e. the differences between two data values of one grid cell along the temporal dimension are usally smaller than then absolute values itself. By encoding sign flips instead of absolute signs between neighboring data values the compression rate can be further improved, as the data values are likely to share the same sign.

In key-frames, neighbors in space are defined as two consecutive data values x_i^t and x_{i-1}^t in one linearization X^t at one time step t. The difference encoding is applied to the respective step numbers n_i^t on $S(n)$ for i and $i-1$ according to $\Delta_i^X = n(x_i^t) - n(x_{i-1}^t)$ as shown in Fig. 5 (1). In difference-frames, two neighbors in time x_i^t and x_i^{t-1} share the same position i in two linearizations X^t and X^{t-1} at two time steps t and $t-1$ exported consecutively. The differences are calculated between step numbers n_i^t for t and $t-1$ according to $\Delta_i^X = n(x_i^t) - n(x_i^{t-1})$ as shown in Fig. 5 (2).

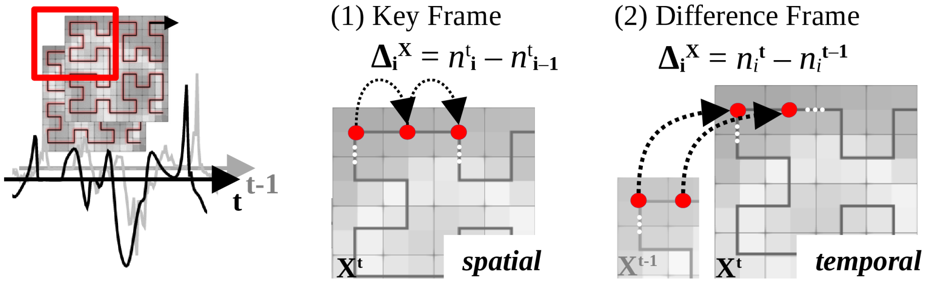

Figure 5: Difference policy for neighboring data values in (1) the spatial dimension and (2) the temporal dimension. The differences in space are computed as neighbors $\Delta_i^X = n(x_i^t) - n(x_{i-1}^t)$ in one time step t along the index i of the linearized sequence X^t, whereas the differences in time are computed as neighbors $\Delta_i^X = n(x_i^t) - n(x_i^{t-1})$ in two time steps t and $t-1$ at the same linearization index i for two linearizations X^t and X^{t-1}.

Compact Grid Data Encoding

GLATE improves the compression rate by encoding differences Δ_i^X and sign flips $s_i \neq s_{i-1}$ under certain circumstances. Given a linearized sequence X^t in key-frames or two sequences X^t and X^{t-1} in difference-frames, the data is transformed into four streams

$$\begin{aligned} X_S &= \{s_1, s_2, \ldots\}, \\ X_E &= \{e_1, e_2, \ldots\}, \\ X_M &= \{m_1, m_2, \ldots\}, \\ X_\Delta &= \{\Delta_1^X, \Delta_2^X, \ldots\}, \end{aligned}$$

composed of signs s_i, exponent parts e_i, mantissa parts m_i according to the quantization scheme, and differences Δ_i^X, according to the spatial and temporal difference policy. The streams X_S and X_Δ, containing signs and differences respectively, are merged into the streams X_E and X_M, containing exponent parts and mantissa parts respectively, according to two rules, as illustrated in Fig. 6:

1. Encode small differences $|\Delta_i^X| < \lfloor \omega/2 \rfloor$ by updating $e_i := 0$ in X_E, for indication of the difference flag, and by moving the difference Δ_i^X in the mantissa part in X_M, updating $m_i := q + 2|\Delta_i^X|$ with $q = 0$ or 1 according the sign of Δ_i^X.
2. Encode the sign flips between values, i.e. if a sign flip occurs $s_i \neq s_{i-1}$, then update $m_i := m_i + \omega$ into a higher order mantissa part.

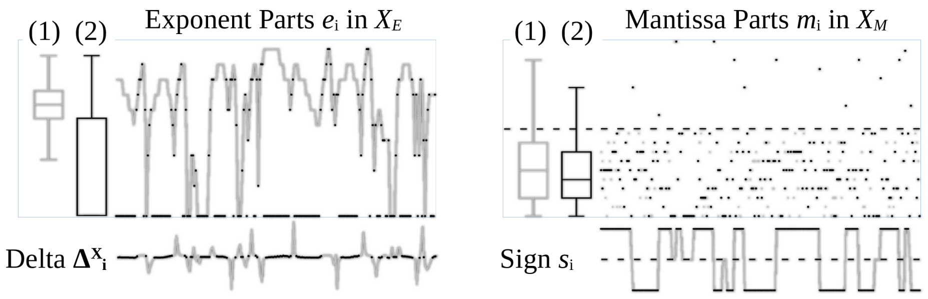

Figure 6: Compacting the encoding for grid compression by merging signs X_S and differences X_Δ into exponent parts X_E and mantissa parts X_M. The box plots show the data (1) before and (2) after the transformation. The encoding creates sequences of zeros in X_E and moves small noise Δ^X into X_M. The sign flip is merged into higher order mantissa parts $m + \omega$.

The first rule ensures, that sequences of differences are encoded as seuqnences of zeros in the exponent parts X_E, whereas small noisy numbers are moved to the mantissa parts X_M. The second rule, improves the compression efficiency, because the sign flip is encoded only for a small percentage of places where it occurs. The exponent parts X_E contain the low-entropy part of the floating point data, while the mantissa parts X_M contain the high-entropy part of the floating point data. As X_S and X_Δ can be reconstructed during decompression from the compressed format, only X_E and X_M are required and need to be compressed using the lossless compression backend, as described in the next section.

Index Variable Encoding

A challenge for in-situ indexing is its efficient integration with compression, in order to reduce storage requirements, while allowing for fast search of grid cells based on data values during post-processing. Where bitmap-based indices on floating point data often require a large storage footprint of $\geq 100\%$ additionally to the data (Wu et al. 2009), recent indexing methods for scientific data like ALACRITY (Jenkins et al. 2013) and DIRAQ (Sriram Lakshminarasimhan et al. 2014) reduce the storage footprint to 50–100% for both index and data.

The procedure used for index creation in *Lossy In-Situ Tabular Encoding for Query-Driven Analytics* (LITE-QA) is called *Error-bounded Binning and Tabular Encoding* (EBATE), which generates an inverted index through binning, similar to the differential encoding used in the lossy floating point compressor SBD (Iverson, Kamath, and Karypis 2012; Lehmann et al. 2018). EBATE uses bins with increasing width, according to the definition of the global step function $S(n)$, under consderation of the user-defined parameters ω and δ for value quantization. The bins are tagged with the global step numbers n, and therefore, the bin values underlie the error bound e_{R}, e.g. $0.01 \ldots 1\%$. The index allows for the fast reconstruction of the location of grid cells based on their value within the error bound.

Index Generation for Fluid Domain

The procedure for index generation, as illustrated in Fig. 7, consists of three steps, which are applied to all index variables u, M, Q on all simulation processes independently. First, the local grid in each simulation process is linearized using the Hilbert-curve and the quantization scheme is applied as described in *LITE-QA Grid Linearization* and *LITE-QA Float Quantization* respectively. The values x_i inside the fluid domain are transformed into the step numbers $n(x_i)$ on the global step function $S(n)$ for the complete local grid $i = 1 \ldots l$ in each process. During linearization for index creation, the voxels of the filter structure are skipped as they do not account to the fluid domain. Hence, unneeded voxels are not in the index and therefore do not create cluttered results with grid cells ususally blanked to zero as placeholders. The index i of the linearization sequence corresponds to the original location of grid cells and is counted consistently while filter voxels are skipped.

Second, after quantization is finished, all tuples $(\bar{n}, i)$ composed of integer step numbers $\bar{n} = n(x_i)$ and their corresponding locations i are sorted lexicographically in ascending order with respect to the first and second component. Due to the lexicographical sorting, large amounts of consecutive tuples in the sorted sequence, which share the same quantization step number $\bar{n}$, are being arranged in clusters $j = 1 \ldots L$ with size c_j, which reflects the amount of repetition of tuples in the sorted sequence with the same $\bar{n}$, hence $\sum c_j = l$. The amount of clusters L depends on the data and varies in every time step. For each cluster j, the L unique step numbers $\bar{n}_j$ and corresponding cluster sizes c_j are collected in arrays $N = \{\bar{n}_j\}$ and $C = \{c_j\}$ accordingly.

Third, for all values $k = 1 \ldots c_j$ in the clusters $j = 1 \ldots L$, the sorted locations i_k^j form sequences of monotonic increasing grid cell indices $i_1^j > i_2^j > \ldots > i_{c_j}^j$. Those indices are collected in index bins $B_j = \{i_1^j, i_2^j, \ldots\}$, which constitute a so-called inverted index.

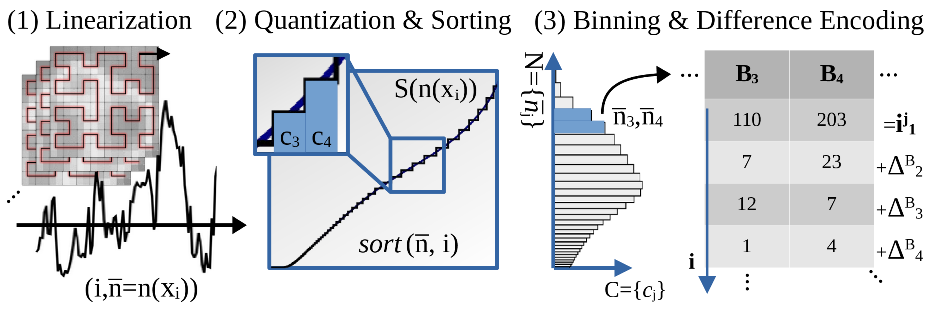

Figure 7: Index generation for fluid domain consists of four steps: (1) linearization of simulation grid and quantization of data values, (2) lexicographical sorting of data $(\bar{n}, i)$ for determination of the bin look-up table $N = \{n_j\}$ and the bin sizes $C = \{c_j\}$, and (3) collecting monotonic sequences of grid cell indices i for all bins B_j from the sorted data. Monotonic sequences of grid cell indices i in index bins B_j are encoded using differences Δ^B.

Compact Index Data Encoding

The results of the index generation procedure are the bin look-up table $N = \{\bar{n}_j\}$, bin sizes $C = \{c_j\}$ and index bins B_j, for $j = 1 \ldots L$ clusters. As scientific data usually concentrates within a smaller bounded value range, sorting the linearized data X yields smooth monotonic sequences (Iverson, Kamath, and Karypis 2012; Sriram Lakshminarasimhan et al. 2014). This allows for a compact representation of the bin look-up table $N = \{\bar{n}_1, \Delta_2^N, \Delta_3^N, \ldots\}$ and the grid cell indices in index bins $B_j = \{i_1^j, \Delta_2^B, \Delta_3^B, \ldots\}$ using difference encoding as shown in Fig. 8. The differences in N are computed between step numbers, i.e. $\Delta_j^N = n_j - n_{j-1}$, whereas the differences in B_j are computed between grid cell indices, i.e. $\Delta_k^B = i_k^j - i_{k-1}^j$.

Block-Wise Grid and Index Compression

Lossy In-Situ Tabular Encoding for Query-Driven Analytics (LITE-QA) applies a compact encoding for grid and index, and uses selected lossless compressors for efficient compression of the specific data parts X_E and X_M produced by GLATE, as explained in *LITE-QA Grid Compression*, and N and B_j produced by EBATE, as explained in *LITE-QA Index Compression*. For lossless compression of high-entropy parts, i.e. X_M, N and B_j, the codec `fastpfor` (Lemire, Boytsov, and Kurz 2016) is used, while low-entropy parts X_E of the floating point data are compressed using `zstd`, a fast general purpose lossless compressor with a very fast decoder.

In order to ensure a high degree of data accessibility and allow for partial decompression of simulation grids, the streams X_E and X_M, each of length $l = 128^3$ values, are split into 512 equally-sized chunks X_E^j and X_M^j for $j =$

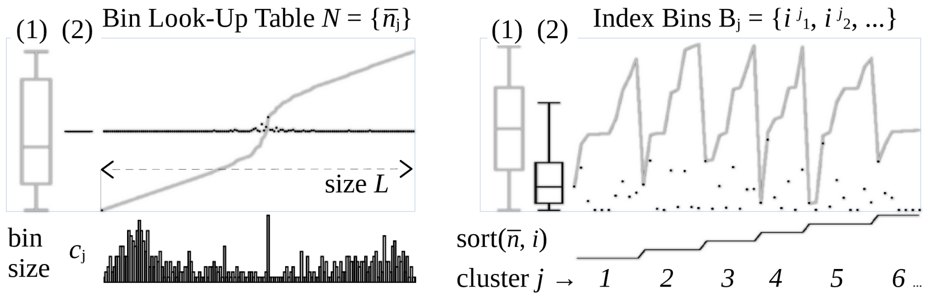
(1) (2) Bin Look-Up Table $N = \{\bar{n}_j\}$
size L
bin size
c_j
(1) (2) Index Bins $B_j = \{i^j_1, i^j_2, ...\}$
sort($\bar{n}$, i)
cluster j →
1
2
3
4
5
6 ...

Figure 8: Compact encoding for the index. Absolute values in the bin look-up table N and in the index bins B_j are replaced with differences Δ^N_j and Δ^B_i respectively. (1) Absolute values in N and B_j, before difference encoding. (2) Noise reduction after application of difference encoding Δ^N and Δ^B. The first value $\bar{n}_1$ in the bin look-up table N and i^j_1 in all index bins B_j is stored as absolute value.

$1, 2, \ldots, 512$. Chunks are arranged in an interlaced sequence of exponents and mantissa parts $O_X, X^1_{EM}, X^2_{EM}, X^3_{EM}, \ldots$, where each pair of chunks X^j_{EM} represents a cubic subgrid of $16^3 = 4,096$ voxels according to the Hilbert-curve linearization as described in *LITE-QA Grid Linearization*. O_X represents the offset table, which stores the size of the compressed blocks X^j_{EM} required for partial grid decompression, and is stored uncompressed.

The index data is arranged in the sequence $L, N, C, O_B, B_1, B_2, \ldots$, where L, N, C, O_B constitute the index header with look-up table size L, bin look-up table N, bin sizes C and offset table O_B. Each bin B_j is compressed independently, unless the size c_j is smaller than 128 indices. In the latter case the bin is merged with the next bin, until the size exceeds 128 (Lehmann et al. 2018). O_B contains the sizes of the compressed index bins, which are required for partial index decompression. L, C and O_B are stored uncompressed.

Quality of Decompressed Data

Fig. 9 shows the error distribution for decompressed data using LITE-QA and ZFP. For LITE-QA, different choices of ω resulting in a maximum error e_R between 0.01% and 1% are shown. ZFP is being operated on precision level 12–18. As can be seen, LITE-QA restricts the point-wise maximum relative error by using error-bounded quantization, while ZFP restricts the average precision of decompressed data, i.e. the average relative error. The GLATE non-temporal compression using $\omega = 35$ and $e_\mathrm{R} = 1\%$ is slightly worse by 2.5% as compared to ZFP in terms of compression ratio, however GLATE guarantees the error bound for all values.

Fig. 10 shows visualizations based on LITE-QA decompressed data with at most

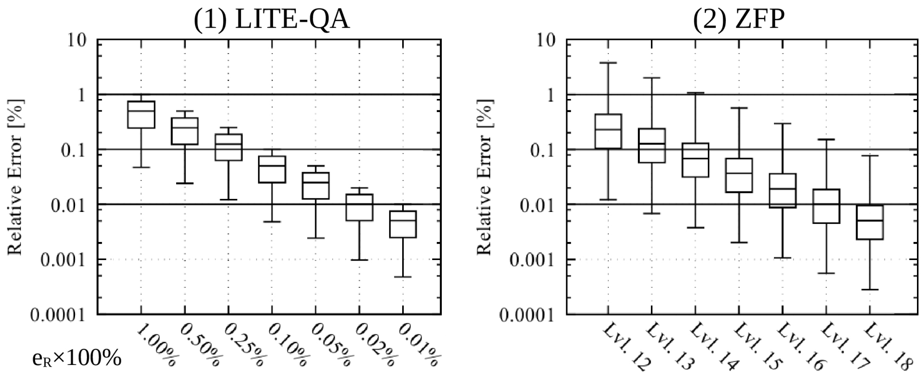

Figure 9: Error distribution for decompressed data. (1) LITE-QA using maximum point wise error e_R between 0.01% and 1% corresponding to different choices for ω as describes in *LITE-QA Float Quantization*. (2) ZFP using precision level 12–18. While GLATE bounds the error for all values, ZFP restricts the average relative error.

1% point-wise error, and raw uncompressed data. Streamlines are seeded in the inlet of the filter and traced along the complete filter depth. As can be seen, streamlines start to diverge after longer distances of tracing at the outlet of the filter. The visual difference for short paths in the inlet is merely perceivable, hence, the decompressed data is sufficient for the task of local visualization, e.g. for the visualizations local regions in the flow field.

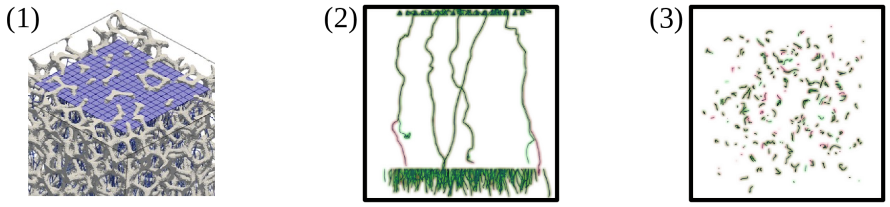

Figure 10: Error accumulation in visualizations based on decompressed data. (1) Streamlines are seeded in the inlet and traced along the filter depth. (2) Visual results for short distances of tracing is sufficiently accurate for local visualization of fluid flow. (3) decompression artifacts, i.e. diverging streamlines reaching the outlet of the filter after tracing along the whole filter depth. Visualization based on uncompressed data is shown in green, based on decompressed data with point-wise maximum error of 1% in red.

Di, Sheng, and Franck Cappello. 2016. "Fast Error-Bounded Lossy HPC Data Compression with SZ." In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 730–39. https://doi.org/10.1109/IPDPS.2016.11.

Dong, Yinlin, Yong Yang, and Chaoqun Liu. 2016. "DNS Study on Three Vortex

Identification Methods." Department of Mathematics, University of Texas at Arlington.

Iverson, Jeremy, Chandrika Kamath, and George Karypis. 2012. "Fast and Effective Lossy Compression Algorithms for Scientific Datasets." In *Euro-Par 2012 Parallel Processing*, 843–56. Springer. https://doi.org/10.1007/978-3-642-32820-6_83.

Jenkins, John, Isha Arkatkar, Sriram Lakshminarasimhan, David A. Boyuka, Eric R. Schendel, Neil Shah, Stephane Ethier, et al. 2013. "ALACRITY: Analytics-Driven Lossless Data Compression for Rapid in-Situ Indexing, Storing, and Querying." In *Transactions on Large-Scale Data- and Knowledge-Centered Systems x: Special Issue on Database- and Expert-Systems Applications*, edited by Abdelkader Hameurlain, Josef Küng, Roland Wagner, Stephen W. Liddle, Klaus-Dieter Schewe, and Xiaofang Zhou, 95–114. Berlin, Heidelberg: Springer Berlin Heidelberg.

Lakshminarasimhan, S., J. Jenkins, I. Arkatkar, Zhenhuan Gong, H. Kolla, Seung-Hoe Ku, S. Ethier, et al. 2011. "ISABELA-QA: Query-Driven Analytics with ISABELA-Compressed Extreme-Scale Scientific Data." In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, 1–11.

Lakshminarasimhan, Sriram, Neil Shah, Stephane Ethier, Seung-Hoe Ku, C. S. Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F. Samatova. 2013. "ISABELA for Effective in Situ Compression of Scientific Data." *Concurrency and Computation: Practice and Experience* 25 (4): 524–40. https://doi.org/https://doi.org/10.1002/cpe.2887.

Lakshminarasimhan, Sriram, Xiaocheng Zou, David A. Boyuka, Saurabh V. Pendse, John Jenkins, Venkatram Vishwanath, Michael E. Papka, Scott Klasky, and Nagiza F. Samatova. 2014. "DIRAQ: Scalable in Situ Data- and Resource-Aware Indexing for Optimized Query Performance." *Cluster Computing* 17 (4): 1101–19.

Lehmann, Henry. 2018. "Temporal Lossy in-Situ Compression for Computational Fluid Dynamics Simulations." PhD thesis, Faculty for Mathematics & Informatics, Technical University Bergakademie Freiberg. https://nbn-resolving.org/urn:nbn:de:bsz:105-qucosa2-314131.

Lehmann, Henry, and Bernhard Jung. 2018. "Temporal in-Situ Compression of Scientific Floating Point Data with t-GLATE." In *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*, 1386–91. https://doi.org/10.1109/CSCI46756.2018.00268.

Lehmann, Henry, Eric Werzner, and Christian Degenkolb. 2016. "Optimizing in-Situ Data Compression for Large-Scale Scientific Simulations." In *Proceedings of the 24th High Performance Computing Symposium*, 5:1–8. HPC '16. San Diego, CA, USA: Society for Computer Simulation International.

Lehmann, Henry, Eric Werzner, Cornelius Demuth, Subhashis Ray, and Bernhard Jung. 2018. "Efficient Visualization of Large-Scale Metal Melt Flow Simulations Using Lossy in-Situ Tabular Encoding for Query-Driven Analytics." In *2018 IEEE International Conference on Computational Science and Engineering (CSE)*, 123–31. https://doi.org/10.1109/CSE.2018.00024.

Lemire, Daniel, Leonid Boytsov, and Nathan Kurz. 2016. "SIMD Compression and the Intersection of Sorted Integers." *Software: Practice and Experience* 46 (6): 723–49. https://doi.org/https://doi.org/10.1002/spe.2326.

Lindstrom, P. 2014. "Fixed-Rate Compressed Floating-Point Arrays." *IEEE Transactions on Visualization and Computer Graphics* 20 (12): 2674–83.

Wu, Kesheng, Sean Ahern, E. Wes Bethel, Jacqueline Chen, H Childs, Estelle Cormier-Michel, C Geddes, et al. 2009. "FastBit: Interactively Searching Massive Data." *Journal of Physics: Conference Series* 180 (August): 012053. https://doi.org/10.1088/1742-6596/180/1/012053.