Battle of the Bits:
Evaluating Lossless Data Compression Algorithms and Cores

Dr. Calliope-Louisa Sotiropoulou, CAST

Abstract

Data compression plays a critical role in modern computing, enabling efficient storage and faster transmission of information. Among lossless data compression algorithms, GZIP, ZSTD, LZ4, and Snappy have emerged as prominent contenders, each offering unique trade-offs in terms of compression ratio, speed, and resource utilization. This white paper evaluates these algorithms and their corresponding hardware cores, providing an in-depth comparison to help developers and system architects choose the optimal solution for their specific use case.

Introduction

The exponential growth of data has made efficient data compression an essential requirement in domains ranging from cloud storage to real-time data processing. Lossless compression algorithms ensure that original data can be perfectly reconstructed from the compressed output, making them indispensable for scenarios where data integrity is paramount.

This paper explores four widely used algorithms—GZIP, ZSTD, LZ4, and Snappy—focusing on their performance in terms of:

- **Compression Ratio**: The degree to which data can be reduced.
- **Compression Speed**: The time required to compress the data.
- **Hardware Resource Utilization**: The computational and memory resources required, particularly in FPGA and ASIC implementations.

Why Do We Use Lossless Data Compression?

High-Volume Data Reduction: AI applications, especially in fields like machine learning, require the storage and processing of massive amounts of data. Lossless compression IPs reduce the data volume without any loss of information, which is crucial for applications where precision and accuracy are essential.

Storage Optimization: In data centers and edge devices, lossless compression IPs help reduce the amount of storage required, enabling more efficient use of space and lowering costs. This is particularly beneficial for storing large datasets and other high-precision data.

High-Speed Data Transmission:

- **Bandwidth Efficiency**: Lossless compression IPs reduce data size, allowing for faster data transfer across limited-bandwidth networks. This is critical for automotive applications, where sensor data have to be transferred in real time with limited bandwidth. On the other hand,

there are applications such as live data streaming where much larger amounts of data need to be transmitted and lossless data compression is essential to reduce the volume.

- **Reduced Latency**: Hardware-accelerated compression and decompression directly on silicon minimizes latency, a significant advantage over software-based compression methods. This low-latency data handling is crucial in AI applications that require real-time responses, and financial trading applications, where response time can make the difference between bad losses and important gains.

Energy Efficiency

- **Power Savings**: Data compression IPs designed for efficient silicon implementation consume less power compared to general-purpose processors running compression algorithms in software. This efficiency is essential in edge AI devices and mobile platforms, where power consumption is a critical concern.
- **Thermal Management**: Reduced power consumption also minimizes heat generation, which can be challenging in high-density computing environments or on small, compact devices like wearables.

Choosing the Right Algorithm

Selecting the right algorithm is not a simple one-size-fits-all decision.

When the selection is made for a software implementation, characteristics such as compression ratio, software latency, and software speed come into play.

LZ4 and Snappy are two compression algorithms that have fast software implementations with low latency but generate a low compression ratio. On the other side, there is Zstd which can offer a higher/configurable compression ratio at a lower speed, and GZIP which, based on its configuration with or without Dynamic Huffman tables, can offer high/moderate compression ratios at a moderate speed.

Algorithm	Compression Ratio	Software Latency	Software Speed	Search algorithm/ Encoding
GZIP/Dynamic	High↑	High	Moderate	LZ77/Huffman
GZIP/Static	Moderate↔	Moderate	Moderate	LZ77/Huffman
LZ4	Low↓	Low	High	LZ77/Dictionary based
Zstd	High↑	High	Moderate	LZ77/FSE
Snappy	Low↓	Low	High	LZ77/Literal

Table 1. Comparing lossless compression algorithms.

When choosing an algorithm implementation for hardware, other factors, such as logic and memory resources, come into play and become fundamental. On HW a high compression ratio comes at the cost of area and latency.

The lower compression ratio algorithms such as LZ4 and Snappy have low latency and a small footprint. Zstd has a much larger footprint and latency than either of these solutions.

GZIP can be configured as both a low footprint, low latency, low compression ratio core and a larger footprint, higher latency, high compression ratio core.

Algorithm	Compression Ratio	Latency	Resources	Search algorithm/ Encoding
GZIP/Dynamic	High↑	High↑	High↑	LZ77/Huffman
GZIP/Static	Moderate↔	Low↓	Low↓	LZ77/Huffman
LZ4	Low↓	Low↓	Low↓	LZ77/Dictionary based
Zstd	High↑	High↑	High↑	LZ77/FSE
Snappy	Low↓	Low↓	Low↓	LZ77/Literal

Table 2. Configuring compression cores for different goals.

Selecting the correct algorithm requires study and experience because it must be based on the data set, the data type, the average and maximum file size, and the correct algorithm configuration.

Our Study and the Silesia Compression Corpus

To evaluate different compression algorithms, we used the Silesia Compression Corpus.

- This dataset includes a variety of file types and sizes, such as text files, executables, MRI images, and databases.
- It provides a comprehensive basis for understanding compression performance across different data types.
- The different files from the Silesia dataset have different types, sizes, and different compression ratios, thus they are a very suitable set to help us determine algorithm performance,

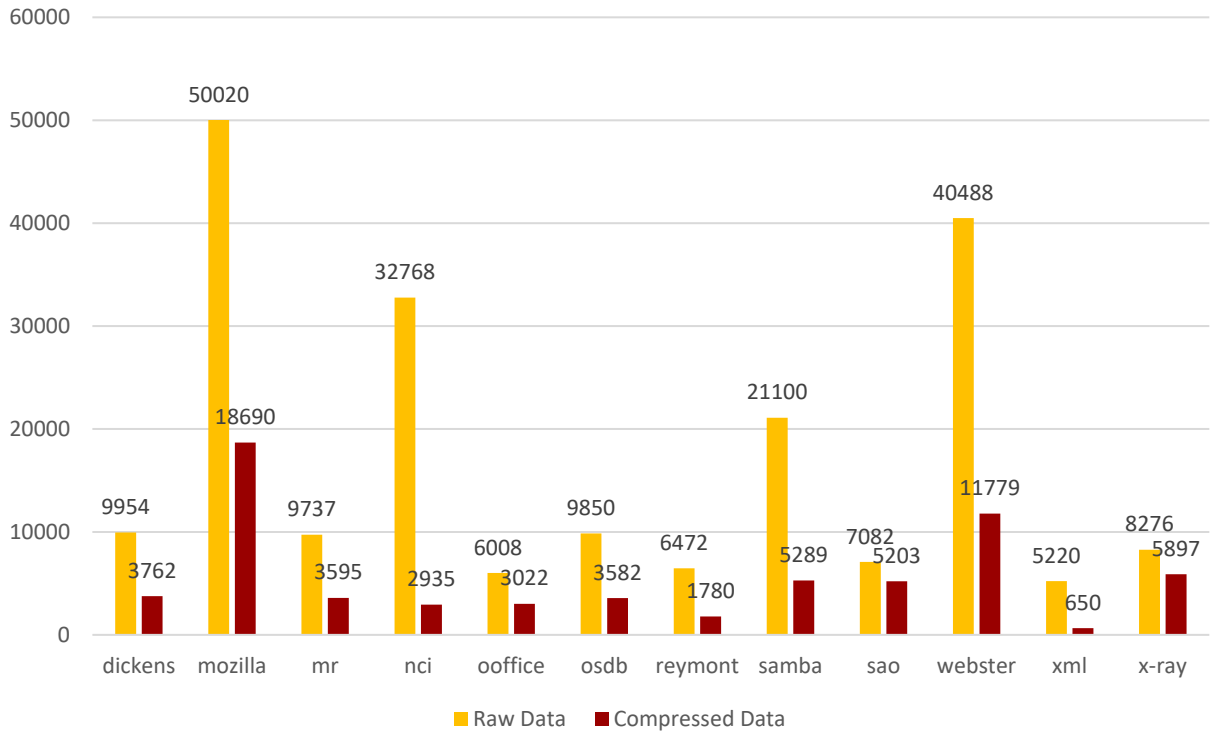

Figure 1. Silesia Corpus Files Compression Gzip Dynamic 32kB Window

To compare the performance of each algorithm, we use the aggregate compression ratio, which is calculated following the formula below:

- n is the number of files (or data blocks)
- $original_i$ is the original size of the i-th file
- $compressed_i$ is the compressed size of the i-th file

$$Aggregate\ Compression\ Ratio = \frac{\sum_{i=1}^{n} original_i}{\sum_{i=1}^{n} compressed_i}$$

The Aggregate Compression Ratio is more accurate than the Average Compression Ratio in this study because it weighs the Compression Ratio based on file size. This has the most important impact on the final core throughput.

Silesia Corpus Results

We first study the aggregate CR of the complete Silesia corpus by using all four algorithms and distinguishing GZIP in two variations: with Dynamic Huffman and with Static Huffman. For LZ4, Snappy, and Zstd we use open and freely available software, and for the GZIP algorithm, we used the CAST ZipAccel compression software model.

The first study we did was to calculate the Aggregate Compression Ratio for each of the algorithms by using the maximum search window permitted by the algorithm. Most of the algorithms have a similar

maximum size (32kB-64kB) with the exception of Zstd, which uses a whopping max window size of 128MB.

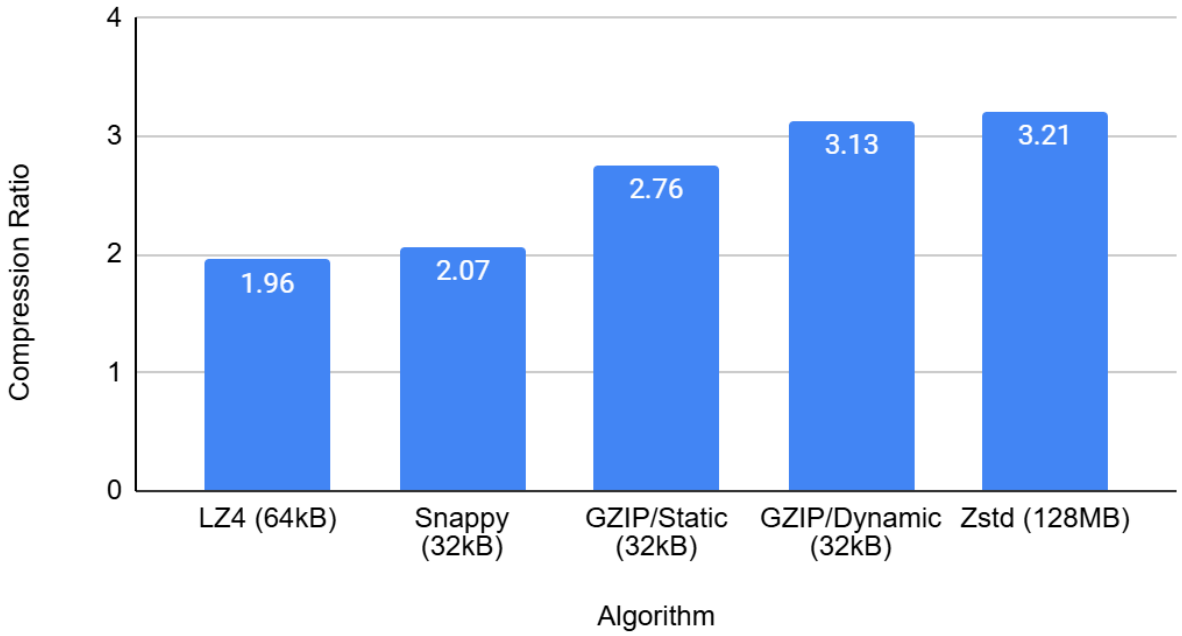

Figure 2.Compression Ratio for Max Window Size

GZIP with Dynamic Huffman tables and Zstd are the algorithms with the highest compression ratios (3.13 and 3.21, respectively). All the above algorithms use the LZ77 search algorithm and a coding method as a second step. The exception is Zstd, which uses two coding method steps (Finite State Entropy Coding and Huffman). The memory requirements, and to a large extent the overall silicon resources, for their custom-hardware implementation are dominated by the History Window of LZ77. To make a fair comparison, we compare the different algorithms using the same search window, to see what kind of compression efficiency we get for roughly the same memory resources.

GZIP with Dynamic Huffman tables offers the highest compression ratio: 3.11. GZIP with Static Huffman tables and Zstd have a similar compression ratio (2.76 and 2.79, respectively) and LZ4 and Snappy have the lowest compression ratios.

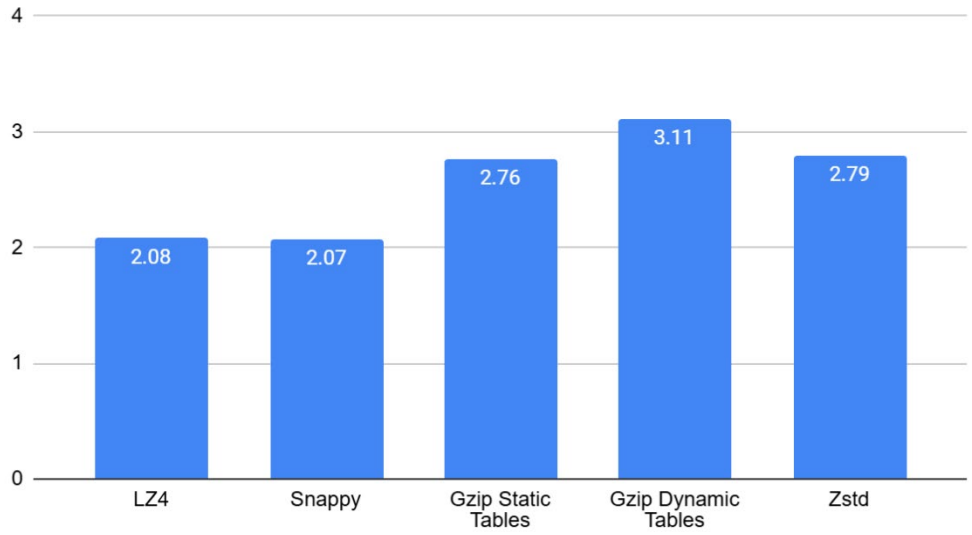

Figure 3.Silesia Corpus CR History Window 32kB

A high throughput hardware implementation will require multiple cores. The cost of memory resources from using a big search window is high, and it gets multiplied by the number of processing engines required to reach the desired throughput. The main strategy to minimize the use of memory resources is to limit the size of the search history window. The size of the history window impacts the compression ratio.

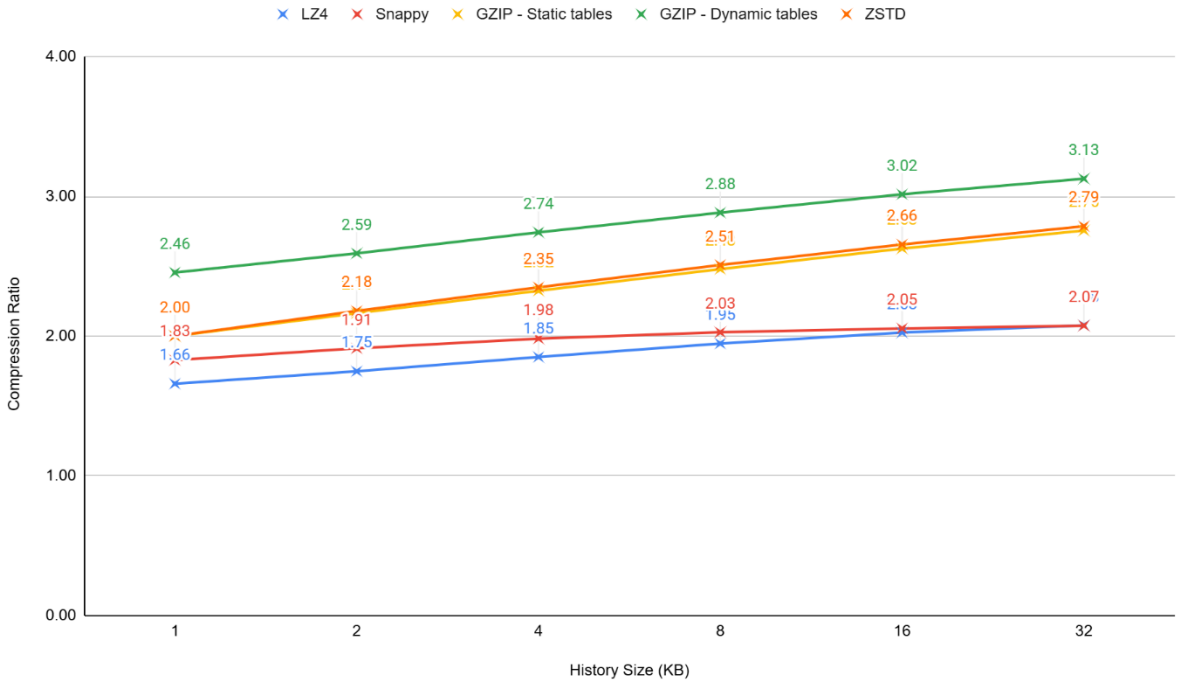

Figure 4.Compression Ratio/History Window

Limiting the History Window size has a lower impact on lower CR algorithms. The choice of the History Window size is a compromise between the availability of resources and the target CR for the application.

Compressing Limited-Size Files

Compression is, in many cases, applied at the file-system level, where each sector (or a group of a small number of sectors) of the disk is compressed independently of the others, and sectors are compressed "on-the-fly" during writes and decompressed during reads. Compressing each sector independently allows for "random" access on a sector level. Sectors are relatively small, typically 4kB. Even at the application level, massive amounts of data are typically organized in chunks of relatively small size (e.g., 4kB, 8kB, or 64kB) that are independently compressed to ease transfers and enable parallel processing and random data access. To explore the efficiency of the algorithms when limited-size chunks or SSD sectors are compressed, we performed tests using 16kB, 8kB, and 4kB files, derived by slicing the Silesia Corpus files accordingly.

For these tests, the size of the history window is limited to the maximum file size for each one of the measurements because it is algorithmically useless to extend the search further than the actual file size. As we can see below, for smaller files, reducing the history window has a smaller impact on compression performance. The history window can be less than half the size of the target input file size without a significant impact on the compression ratio. See Figures 5–7.

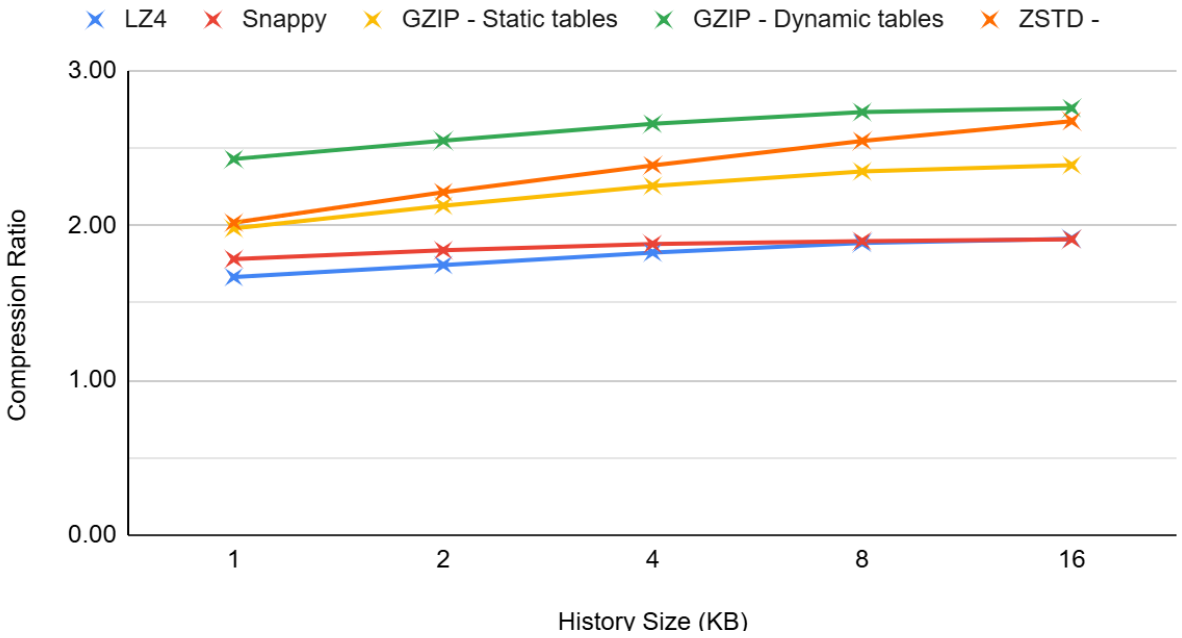

Figure 5. Compression Ratio/History Window for 16kB Files

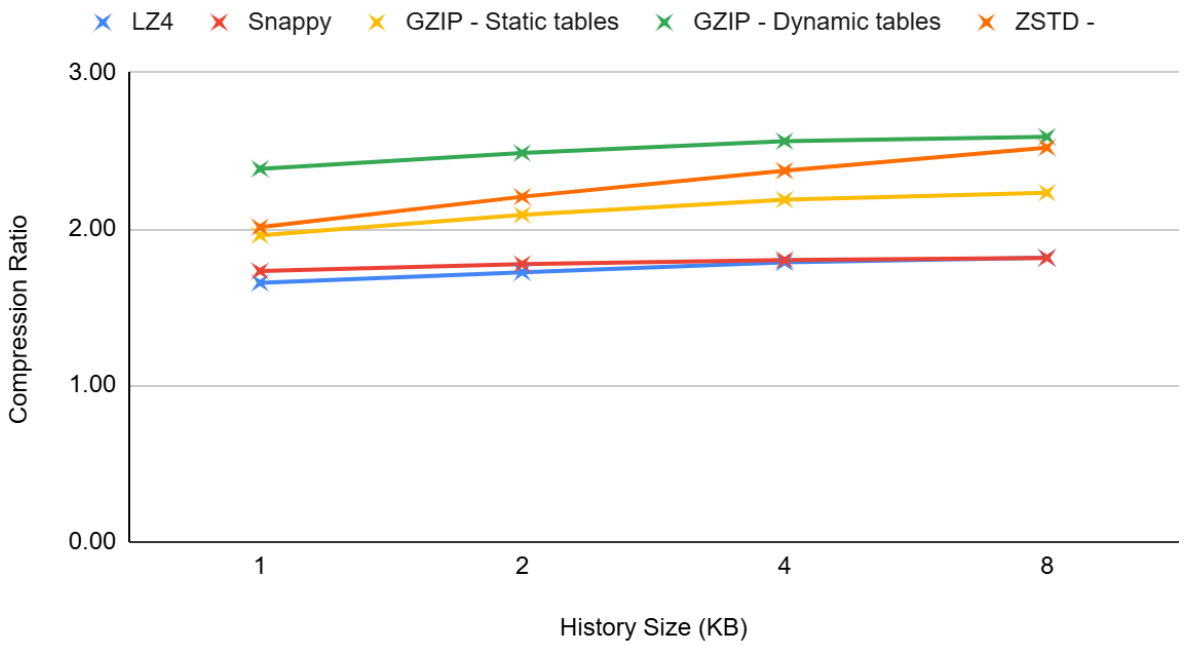

Figure 6. Compression Ratio/History Window for 8kB Files

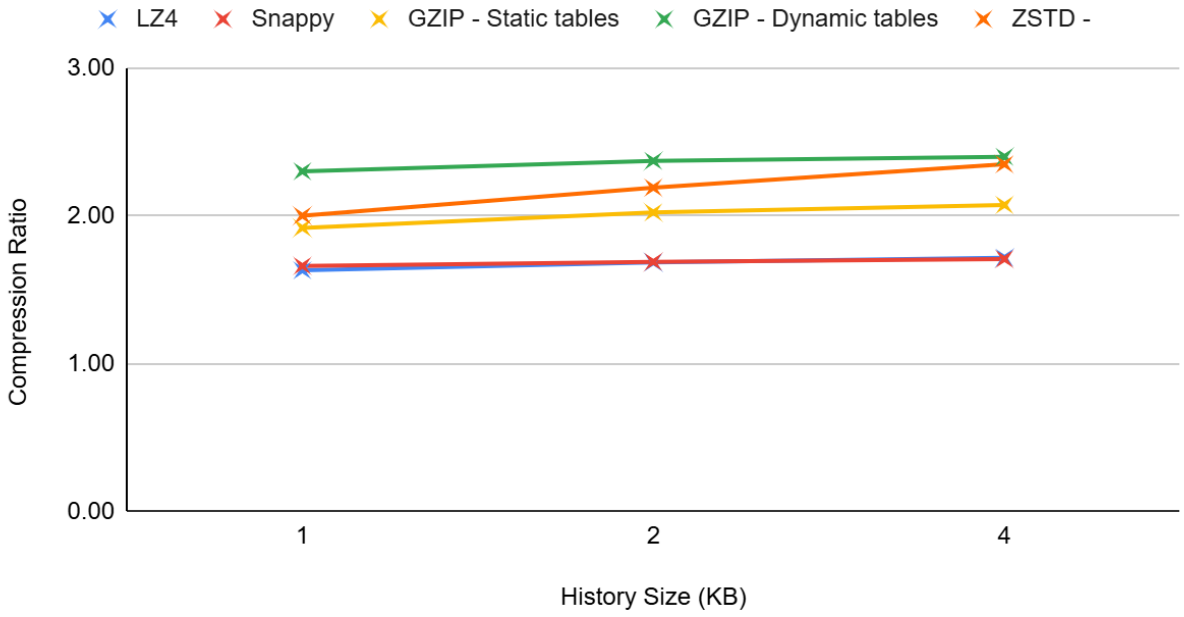

Figure 7. Compression Ratio/History Window for 4kB Files

Figure 8 demonstrates the maximum compression ratio that can be achieved with each one of these lossless algorithms based on the selected Maximum File Size.

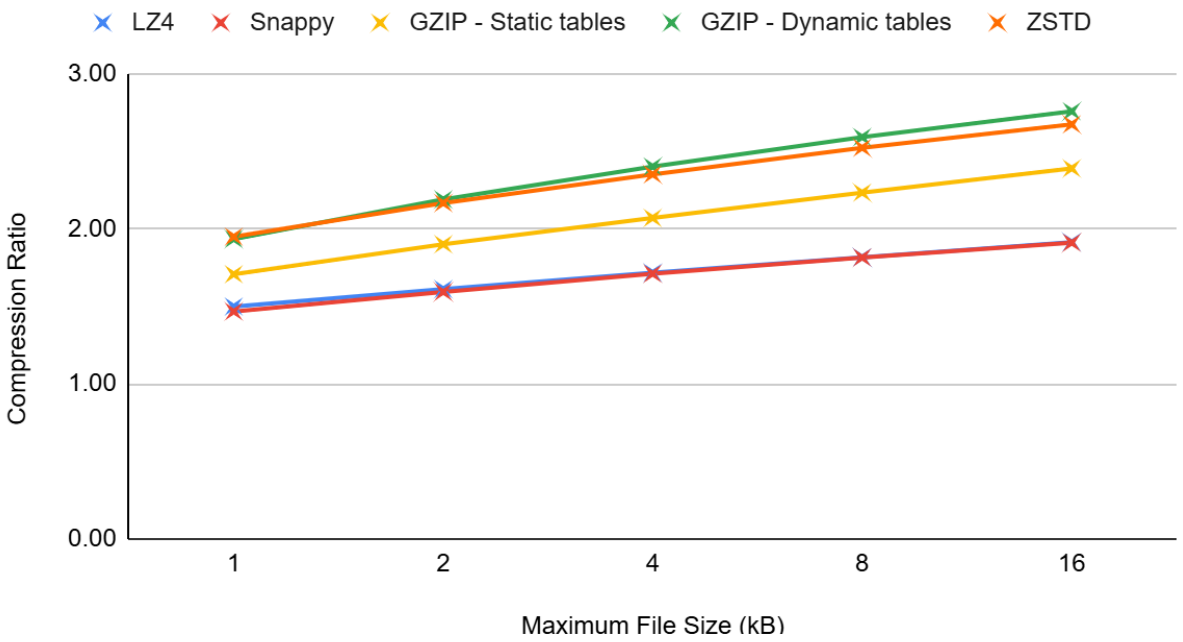

Figure 8. Compression Ratio/Maximum File Size

A smaller file size significantly reduces buffer sizes but it also reduces the maximum compression ratio that can be reached.

Choosing the Right Hardware Solution

Based on the results of our study, we can see that for any kind of History Window that is realistic on a hardware implementation, GZIP with Dynamic Huffman tables will offer the highest compression ratio.

The GZIP Dynamic Huffman core comes at the cost of more resources and greater latency, but the latency will always be lower than the latency of a Zstd core. Zstd, other than LZ77, uses two different coding algorithms to reach high compression efficiency with a higher latency cost and a larger footprint. If Zstd is implemented with a single coding algorithm, then its compression efficiency drops significantly.

If low-latency and/or a small footprint is a requirement, then GZIP with Static Huffman tables will offer the highest compression efficiency. LZ4 and Snappy implementations have latencies and footprints comparable to the GZIP/Static Huffman cores for the same throughput. LZ4 and Snappy can be a hard requirement for compatibility with external software.

Lossless Data Compression Cores from CAST

CAST offers a range of cores for lossless data compression to answer the customer's every need, including:

- The ZipAccel Compression and Decompression cores that support the GZIP, Zlib, and Deflate protocols.
- The LZ4SNP core, combining LZ4 and Snappy algorithms.

The CAST ZipAccel-C and ZipAccel-D cores have been in the market for more than a decade and they have been productized tens of times on both FPGAs and ASICs, even on the smaller/newer 4nm and 3nm nodes. The CAST ZipAccel cores can be configured to support Dynamic and Static or Static-only Huffman tables.

The LZ4SNP cores are the newest addition to the CAST compression portfolio. They offer the choice of using LZ4, Snappy, or both for a single implementation.

Both core families support throughputs >100Gbps on modern FPGA devices and >400Gbps on ASICs.

CAST can offer solutions that cater to various needs, from low-latency cores/small footprint cores to high-compression-ratio/high throughput cores.

At CAST, we focus on understanding customer requirements, such as:

- Throughput, technology, file size, and input data types.
- Priorities like latency, resources, and compression ratio.

Based on these parameters, we offer tailored solutions, supported by simulation results for throughput and PPA.

We can proudly say that we are experts in the Lossless Data Compression market with more than 10 years of experience and customer-proven satisfaction.

For more information, please visit the CAST website or contact sales@cast-inc.com.