# A fast, highly reliable data compression chip and algorithm for storage systems

by J.-M. Cheng
L. M. Duyanovich
D. J. Craft

**Data compression allows more efficient use of storage media and communication bandwidth, and standard compression offerings for tape storage have been well established since the late 1980s. Compression technology lowers the cost of storage without changing applications or data access methods. The desire to extend these cost/performance benefits to higher-data-rate media and broader media forms, such as DASD storage subsystems, motivated the design and development of the IBMLZ1 compression algorithm and its implementing technology. The IBMLZ1 compression algorithm was designed not only for robust and highly efficient compression, but also for extremely high reliability. Because compression removes redundancy in the source, the compressed data become extremely vulnerable to data corruption. Key design objectives for the IBMLZ1 development team were efficient hardware execution, efficient use of silicon technology, and minimum system-integration overhead. Through new observations of pattern matching, match-length distribution, and the use of graph vertex coloring for evaluating data flows, the IBMLZ1 compression algorithm and the chip family achieved the above objectives.**

## Introduction

The addition of compression capability to a DASD subsystem facilitates more efficient use of subsystem resources such as cache, data path bandwidth, and disk capacity in a manner transparent to the system storing the data. Compression allows existing platforms and applications to benefit from a lower storage cost and potentially higher performance with no change to system hardware or software. The maximum benefit is achieved when the compression is performed without performance loss. This requires a technology which is capable of running at channel speeds (18MB/s for ESCON®[1]), and the ability to pipeline data through the compressor without significant store-and-forward penalties.

[1] ESCON®, or Enterprise Systems Connection Architecture®, defines full-duplex architecture between channels and control units; the maximum peak data rate is 19.62 MB/s.



**0018-8646/96/$5.00 © 1996 IBM**

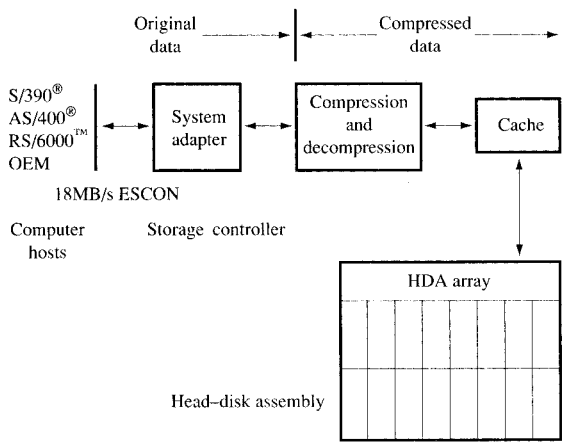


**Figure 1**

Storage controller with compression capability.

In addition to the obvious benefits in disk capacity (the IBMLZ1 compression algorithm described below typically achieves better than a 3:1 savings in DASD capacity for high-end systems), there are additional benefits to the subsystem. If the data are compressed as they enter the subsystem (see **Figure 1**), the cache resource has effectively been tripled. Customers can either benefit from improved performance due to better hit ratios, or reduce their cost by configuring smaller amounts of cache. An added benefit of data compression upon entry to the subsystem is reduced utilization of internal buses and DASD paths as data flow through the subsystem. Finally, sequential performance can be improved. In general, on high-end systems with ESCON-attached DASD, the throughput of sequential operations is gated by the DASD data rate, typically less than the 18MB/s capability of the channel. When the device is transferring compressed data, the transfer rate is effectively multiplied by the compression ratio, allowing the full capability of the channel to be realized.

The development of a new compression algorithm and technology was necessary for the direct pipeline support of 18MB/s ESCON, up to 40MB/s for tape, and for the emerging higher-speed communication protocols. The core of the 3490 IDRC[2] compression algorithm is the FileCOMP file compression model, with the BAC[3] compression chip as engine [2]. The FileCOMP and the BAC were developed between 1980 and 1983. The largest gate-array sizes for CMOS technology available in 1981 were 3–4K gates, which was one of the key limiting factors in the selection of the compression algorithm at that time. To date, the full functional IBMLZ1 compression chip has 120 times the BAC circuits, delivers 16 times BAC throughput at twice the BAC clocking rate, and achieves about 30% better compression.

## Overview of compression algorithm and system

A compression system (**Figure 2**) generally consists of a model unit and a coding unit. The objectives of the model unit are 1) to provide a context model for the data or to characterize the data as string symbols, and 2) to provide a statistics model for the distribution of the extracted symbols. Common methods used for redundancy reduction are run-length encoding, pattern matching, transformation, and transform coding. The outputs of the model unit are the extracted symbols, and possibly the statistics of the symbol distribution. The objective of the coding unit is to minimize the overall coded length by assigning an optimal code word for each symbol according to its assumed probability. It is important to note that *the effect of compression can take place in the model unit, the coding unit, or both*. Well-matched model and coding units give the most compression benefit. Nevertheless, the computational complexity of hardware or software often limits the practical choice of the model and coding units for intended applications.

Two common choices for the coder unit are Huffman coding [3] and arithmetic coding [4–6]. In Huffman coding, each input symbol is mapped to a code word composed of an integer number of bits. The coded stream is a concatenated sequence of code words. The Huffman code satisfies the Kraft inequality [7] and can be uniquely decoded. The worst-case redundancy, defined as expected code length less the binary Shannon's entropy[4] [8], occurs when the most probable symbol probability is greater than 50%. This case occurs quite often in the compression of black-and-white images. Arithmetic coding does well in broad binary symbol cases. The code length of the arithmetic coding can be made arbitrarily close to the Shannon's entropy of the information, and thus achieves very low redundancy. The arithmetic coding method can be thought of as a generalization of Huffman coding without the need for prefix codes or integer-length code words. The Huffman code, however, remains the most

[2] IDRC, or Improved Data Recording Capability [1], is a data compression and compaction standard.

[3] The BAC, or Binary Adaptive Coder, chip was developed in 1981–1982 on a 3Kb LSI logic CMOS array. The BAC chip is also referred to as the skew coder chip, because the twelve augends were $2^{-skew}$, or $2^{-1}$ through $2^{-12}$.

[4] Shannon's entropy defines the average amount of information, $\Sigma_i\, p_i \times \log_2 (1/p_i)$, for a binary system; $\log_2 (1/p_i)$ is the binary information for symbol $i$ with probability $p_i$. Information, in a sense, describes a degree of surprise. The more frequent symbol requires a shorter number of bits to code. If the symbols *0* and *1* appear 93.75% and 6.25% of the time, respectively, the optimal code lengths for *0* and *1* are 0.093 bit and 4 bits, respectively.

popular encoding method for its simplicity and its general effectiveness.

Arithmetic coding encourages [9–11] a clear separation between the model and coding units, and accommodates the adaptive model in a natural and coherent manner. The coded output stream of the arithmetic coder resembles a single number of extremely high precision. The result is the sum of sequences of addition and shift operations. Arithmetic coding, though conceptually more complex, lends itself well to adaptation and excellent coding efficiency [12].

Shannon conceived the notion of arithmetic coding, and in [13] the method was made symbolwise recursive. Development of the full arithmetic coding technique that is known today is due to Pasco and Rissanen. The complexity of the model, adaptor, and arithmetic coder, however, delayed the practical use of arithmetic coding in hardware and software until a sequence of significant computational simplifications was made. The successful integration of approximate counting [2, 14, 15] and probability estimation [16]; the simplification of multiplication and division by operator strength reduction to fixed-augend-based addition and subtraction; and a simplified common adaptive mechanism shared among large possible contexts for adaptation resulted in a hardware reduction of more than 50 to 1 and a software speed-up as well. The BAC chip, developed in 1981–1982 [2], and the black–white image-compression system (ICOM) [17] both benefited from the drastic reduction in complexity. A further major computational reduction which affected both hardware and software was the Q-coder algorithm [18]. The ABIC [19] chip, which is based on the ICOM model and Q-coder, is used in IBM high-speed check-processing products. The SUNSET gray-scale image-compression algorithm based on the BAC chip influenced the JPEG[5] standard. IBM also developed a simulated annealing method for the automated optimization of the arithmetic coder probability-estimation table [20].

• *Review of LZ1 and LZ2 compression algorithms*

Ziv and Lempel's compression algorithm 1 [21] and algorithm 2 [22] are commonly known as the LZ1 and LZ2 compression algorithms. The LZ1 and the LZ2, in their original form, expressed the notions of a coding model and bounds on compression. Professor Lempel noted that more than 90 percent of the compression software in the PC world is derived from either LZ1- or LZ2-class algorithms.

The basic data structures and operations of the LZ1 compression algorithm are the following:

[5] JPEG usually denotes the gray-scale image compression standard proposed and defined by the ISO/CCITT Joint Photographic Experts Group.

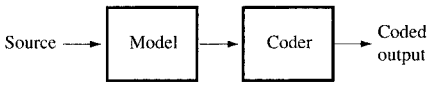


**Figure 2**

Compression system with a model unit and a coding unit.

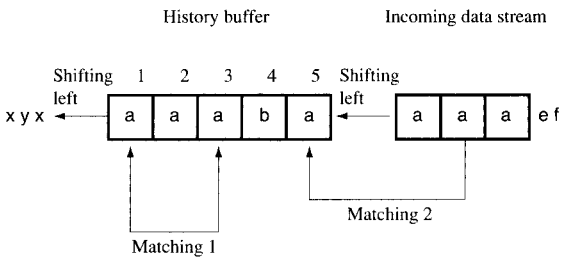


**Figure 3**

Pattern matching of string. Two matchings of the maximal incoming substring aaa are indicated in the above five-element shift register history buffer.

1. Construct a history structure of the past stream or of the most recent part of the past stream.
2. Use pattern matching to find the maximal incoming substream that matches a substream residing in the history structure.
3. Replace the incoming substream with a pointer into the history structure to the matching substream and length (or their equivalents) if the coded description is shorter than the original substring.

Conceptually, the history structure used in the LZ1 algorithm is a sliding window of fixed size. One can consider it as a shift register of fixed length, which contains the recent past symbols. **Figure 3** shows maximal (length) matches of the incoming substring found in two locations. Match number 1 starts at location 1; match number 2 starts at location 5 and extends into the inputting stream. The matching string is aaa, which usually takes 24 bits to code. Since aaa is also found in the history buffer, the alternative coding form is to use three

```
History buffer Input characters
000000000000000 t
0000000000000t h   note: t is in match point 1
00000000000000th e   note: t is still in match point 1
000000000000the r
0000000000ther e match length:  1, match points: 2,
0000000000there f
000000000theref o
00000000therefo r match length:  1, match points: 3,
0000000therefor e match length:  2, match points: 3,
000000therefore
00000therefore  t match length:  1, match points: 0,
0000therefore t h match length:  2, match points: 0,
000therefore th e match length:  3, match points: 0,
00therefore the   match length:  1, match points: 9,
0therefore the  t match length:  2, match points: 9,
therefore the t h match length:  3, match points: 9,
herefore the th e match length:  4, match points: 9,
erefore the the m
refore the them e match length:  1, match points: 4, 8, 12, 0,
efore the theme   match length:  2, match points: 8, 12,
fore the theme  t match length:  3, match points: 8, 12,
ore the theme t h match length:  4, match points: 8, 12,
re the theme th e match length:  5, match points: 8, 12,
e the theme the n
 the theme then   match length:  1, match points: 9, 13, 3,
the theme then  f
he theme then f o
e theme then fo r
 theme then for w
theme then forw a
heme then forwa r match length:  1, match points: 11,
eme then forwar d
me then forward   match length:  1, match points:  3,  8,
e then forward  t match length:  2, match points:  3,
 then forward t h match length:  3, match points:  3,
then forward th e match length:  4, match points:  3,
hen forward the r match length:  1, match points: 11, 14,
en forward ther e
n forward there
```

**Figure 4**

One-byte prefix extension and history buffer size = 15.

bits denoting the starting address, another three bits for the maximal matching length, and one bit tag to state whether there is a substitution or not. In this example, we can replace the 24-bit aaa substring with 7-bit coded words (1-bit tag, 3-bit length, and 3-bit starting address). In fact, in this example, if even a single byte matched it would be beneficial to use the coded form. The alternative provides a 1-bit saving, using 7 bits instead of 8 bits. In the event no match is found, the input byte is coded with 9 bits, a 1-bit tag denoting no match followed by the original byte.

In 1976, Jackson and Rackl described a data expansion apparatus with which, in a long data stream, the repeated sections of data can be saved by the substitution of tag, address, length, and number of repetitions for storage space saving. The idea is essentially the same as the LZ1. It was filed in a patent application in 1976, and became a U.S. patent in 1977 [23]. In the patent description, some types of the LZ2 algorithm were also described.

The LZ2 compression algorithm [22, 24] follows the same principle, but normally has a tree-structured history buffer for improved software efficiency in the search for matches.

- *Hardware flow speed-up*

Fast LZ1 hardware organization is normally based on two themes: the use of prefix extension [25] and the use of very powerful parallel combinatorial operators. For instance, the longest match in Figure 3 is aaa. Instead of finding the longest match in one operation, prefix extension is a simple divide-and-conquer approach. The operation is broken down into the smaller job of searching for one byte, two bytes, or any small fixed number of bytes in the input stream. For the one-byte case, prefix extension finds first letter a to be the first prefix, then tries to extend the prefix. The substring aa becomes the extended prefix, since a was the prefix for aa. Next, aaa becomes the extended prefix. However, aaa is not a prefix to a longer match; i.e., aaa cannot be further extended. Prefix extension then concludes that the prefix aaa is itself the longest match. For the powerful parallel combinatorial operator aspect, arrays of massive parallel comparators are used for the high-speed LZ1 hardware organization. Prefixes of one byte or multiple bytes are simultaneously compared against all storage locations in the history buffer. Since the above organization resembles the function of content-addressable memory, it is often referred to as CAM.

**Figure 4** shows the use of prefix extension of one byte and a history buffer storing the fifteen previous bytes. The prefix-extension algorithm used is greedy. If a byte match is found, it assumes that it is itself the first prefix and tries to extend the prefix from that byte on.

It is interesting to note that the use of prefix extension and parallel compare operators resembles the algorithmic notion of radix sort [26]. As in any two-input comparison-based sorting algorithm, the running time is bounded lower by the information theoretic $n \log n$, where $n$ is the input size. Radix sort, with its more powerful comparing operator and sequential sort through each index of each number, is able to achieve a linear relationship between run time and input size. Our development of the high-speed LZ1 compression data flow follows the radix-sort-like mathematical notion of using more powerful parallel and compounded operators to lower the number of machine cycles needed for compressing or decompressing a byte to one cycle, or even a fraction of a cycle.

## Development of the IBMLZ1 algorithm and implementing technology

In addition to its objectives of robust and efficient compression, the IBMLZ1 compression algorithm was designed for reliability. The objectives for the IBMLZ1 compression algorithm (**Figure 5**) which were established at the outset were

- *Extremely high reliability:* One undetected error in $10^{30}$. Because compression removes redundancy in the source, the compressed data become extremely vulnerable to data corruption. A single bit error in the coded stream could result in a total decoding failure from that bit location onward.
- *Hardware execution efficiency:* The hardware architecture should use as few machine cycles as possible to compress or decompress a byte. The architecture should maintain low complexity and use the silicon technology effectively. In addition, it is desired to minimize increases in complexity as the CPB[6] becomes small.
- *Robust compression:* Good coding efficiency should be achieved for broad applications.
- *Minimal system integration overhead:* The maximum benefit from compression is achieved when the compression can be performed without performance loss. This requires a technology that is capable of running at channel speeds (18MB/s for ESCON) and the ability to pipeline data through the compressor without significant store-and-forward penalties.

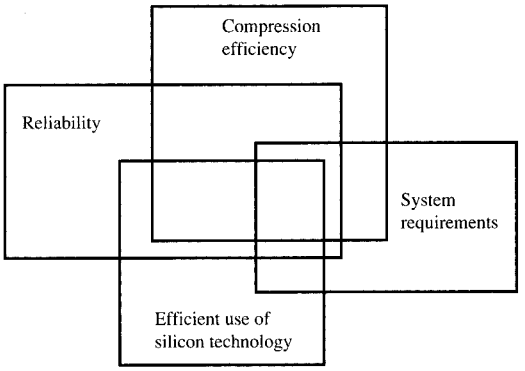


**Figure 5**

IBMLZ1 compression algorithm development objectives.

- *Extremely high reliability*

Extremely high reliability (one undetected error in $10^{30}$) is achieved through the combined use of highly reliable CMOS technology and the compression–decompression coupled-checking scheme. **Figure 6** shows that the CRC (cyclic redundancy code) of the original data is checked against the CRC of the decompressed data. The four-byte CRC check improves checking power by a factor of almost $10^{10}$. Two copies of the CRCs are compared by two independent comparators to avoid a single point of failure.

The compression–decompression pair does present system constraints. The compressor may not emit any code word for a long while during the passage of a highly compressible stream. The maximal compression latency is the number of original bytes the compressor takes before it emits a code word. At peak compression, the decompressor will be running behind the compressor by an amount equal to the maximum latency. This implies that the FIFO buffer between the compressor and decompressor must be at least the latency times the data expansion[7] factor in size; this constraint compels the design of IBMLZ1 to keep the latency small. A smaller compression latency also improves the response time of the storage control system.

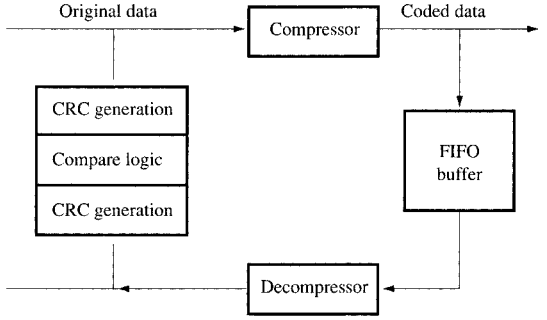


**Figure 6**

Compression and decompression pair for encoding reliability.

*CAM scrubbing operation* As pointed out earlier, the operation of the CAM is based on using very powerful parallel combinatorial operators, and the search for the longest match is done by prefix extension. What if the prefix extension hardware were to fail? The incoming stream would then be coded as raw bytes in every case, since no match longer than one byte could be found. The coded output stream would then be expanded by 12.5% owing to the tag bit used to indicate whether or not a substitution occurred. In this case, the compression–decompression pair would not be able to detect any error at all, since the coded stream could be correctly decoded. This is the class of performance

[6] The CPB (cycles per byte) is the number of machine cycles needed to compress or decompress a byte. CPB resembles the use of CPI, or cycles per instruction, for measurement of computer architectural effectiveness in RISC and CISC.

[7] Data expansion occurs when the size of the output stream generated by the compression algorithm is larger than the size of the input. The maximum compression achievable is bounded below by the entropy, which is data-dependent. In addition, there are factors that drive the coded size away from optimal: coding overheads and imperfect models. The worst-case expansion factor for IBMLZ1 is 12.5%.

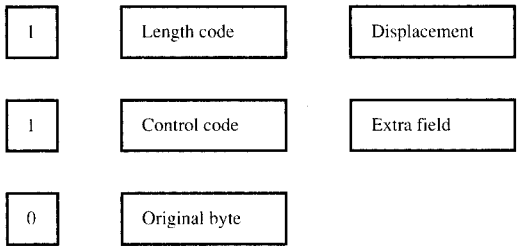


**Figure 7**

IBMLZ1 coding format.

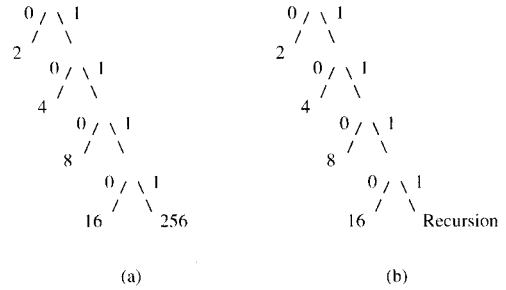


**Figure 8**

286LZ1 and K_code for I-length code assignment: (a) 286LZ1 limited match length; (b) K_code infinite match length.

degradation error that the CAM scrubbing is intended to prevent.

During CAM scrubbing, the CAM is split into two halves. A minimum of 768 test patterns are run through each half CAM. Outputs at every cycle are compared during this time.

• *Hardware execution efficiency: Low CPB and low complexity*

CPB, the number of machine cycles needed to compress or decompress a byte, is a measure of architectural effectiveness for the compression and decompression units. The first data flow developed had a mixture of $CPB = 3$ and $CPB = 5$ for compression. For decompression, the *CPB* was 1. With a chip running at 40 MHz, the data flow delivered about 10 MB/s in the compression mode, and 40 MB/s in the decompression mode.

The asymmetrical compression and decompression rates of the first data flow were undesirable. The variable encoding rate also presented undesirable control overhead. Efforts to improve the original data flow were not fruitful until the data flow was mapped to a data-dependency graph. The graph appeared to contain a clique (all-connected subgraph) of degree 3 and a self-loop on the exiting vertex of that clique. The clique corresponded to three irreducible computation cycles, and the self-loop represented the variable two additional cycles. As a result of these findings, the first data flow was dropped.

A new data flow was sought, one which would exhibit a bipartite data-dependency graph. That property allows a data flow to be converted directly into a fully pipelined, two-phase clocking design. Our approach was to seek new, more powerful parallel combinatorial operators which run two or more prefix extensions in parallel [27]. Encouragingly, these also removed the need for interstage control logic altogether. Control dependency, in a sense, is a function of distant past data dependency. In the new data flow, since each execution stage depends only on the data from the previous stage, the need for interstage control is removed. Our second data flow achieved rather impressive, *symmetrical CPB* = 1 for compression and decompression (most LZ1- and LZ2-type compressors today have CPB ranging from 2 to 5) without the need for interstage control.

*286LZ1, IBMLZ1, and ALDC code development* The IBMLZ1 compression coding algorithm (format) is a subset of the 286LZ1 algorithm [28]. The IBMLZ1 is used in IBM high-performance DASD controllers, tape drives, and the AIX® file system with compression. The ALDC (Adaptive Lossless Data Compression) algorithm is a smaller subset of the 286LZ1 algorithm. The ALDC algorithm has been approved as the Quarter-Inch Cartridge Drive Standard, or QIC-154.

The generic IBMLZ1 compression code word format is depicted in **Figure 7**. The tag, the first bit, is chosen similar to Jackson's [23]. When the tag bit is a *0*, the next field is the 8-bit original byte. The existence of this tag bit accounts for the worst-case 12.5% expansion for IBMLZ1 technology. When the tag bit is a *1*, the next field is either a variable-length field or a control field. If the length field is used, the next field is the displacement. The displacement field is a pointer to the history buffer, where the head of the match is located. If the control field is chosen, there might be 0, 1, or multiple extension fields. The control field is assigned for several important purposes. It allows messages to be embedded in (effectively, "pipelined" with) the compressed data stream. It can also be designated for future decoder redirection.

The displacement field is not encoded. One study[8] has shown that encoding the displacement has limited benefit for restricted types, such as PC object codes. For broader applications, and for hardware simplicity, the decision was made not to encode the displacement field.

The length/control field of the IBMLZ1 algorithm contains 286 code words (**Figure 8**) grouped in five "buckets." The bucket scheme appears in the SUNSET [29] gray-scale compression algorithm, JPEG, and other coding schemes. For the first four buckets, the number of code words in each bucket is twice the number in the preceding bucket, creating a "lopsided tree." The 30 code words in the first four buckets approximate the observed exponential distribution.

**Figure 9** shows analytical results of a simpler distribution from one test case. There were 738 counts of zero match found in the CAM; 7623 counts of match length = 1; 3000 counts of match length = 2; 798 counts of match length = 3; and so on. The exponential distribution (or fractal) of code length is clearly shown.

- *Empirical study of match-length statistics*

Forty-four test cases of expected data were chosen as the IBMLZ1 algorithm development test suite. They encompassed databases, programs, object code, system code, and documents in two languages from major applications on VM, MVS®, RS/6000™, and PC. K_code[8] was used as the basis for the experimental length/control code study (see **Figures 10** and **11**).

*Key new observations* The empirical experiment of analyzing the match-length distribution over the 44-case suite and also over large volumes of data produced two important observations:

1. If maximal match length is limited to 192, the increase in total compressed bytes of the test suite is less than 0.5% in comparison to the maximal match length = 2048 case.
2. For maximal match length = 286, the match-length distribution is exponential until the match length reaches approximately 27. Beyond 27, the distribution is very low and appears to have no order.

The 192-match-length-limit observation was encouraging, since it suggested that a smaller set of length codes would suffer only insignificant loss compared to the set with 2048 length codes. The shorter match length presented considerable (logic) savings for the compression–decompression checking mechanism. In addition, short match length corresponds to lower latency for the storage system controller (or decoder latency), where the response time is crucial. The maximal match length for the final IBMLZ1 algorithm is 271 (Figure 11).

The second observation was extremely crucial to the algorithmic decision. Since the distribution is exponential, it indicated that more complex adaptive arithmetic coding, in this case, could perform only slightly better than Huffman coding. The more complex adaptive coding scheme was thus discarded.

| Match length | Match count | Coded bits | | |
|---|---|---|---|---|
| 0 | 738 | 6642 | | |
| 1 | 7623 | 68607 | | |
| **Matched symbols:** | | 4514 | | |
| **Match length** | **Match count** | **Coded bits** | **Probability** | **Entropy** |
| 2 | 3000 | 6000 | 0.66460 | 0.39174 |
| 3 | 798 | 1596 | 0.17678 | 0.44195 |
| 4 | 301 | 1204 | 0.06668 | 0.26050 |
| 5 | 159 | 636 | 0.03522 | 0.17004 |
| 6 | 87 | 348 | 0.01927 | 0.10981 |
| 7 | 44 | 176 | 0.00975 | 0.06512 |
| 8 | 22 | 132 | 0.00487 | 0.03743 |
| 9 | 31 | 186 | 0.00687 | 0.04935 |
| 10 | 13 | 78 | 0.00288 | 0.02431 |
| 11 | 9 | 54 | 0.00199 | 0.01788 |
| 12 | 10 | 60 | 0.00222 | 0.01954 |
| 13 | 4 | 24 | 0.00089 | 0.00899 |
| 14 | 4 | 24 | 0.00089 | 0.00899 |
| 15 | 4 | 24 | 0.00089 | 0.00899 |
| 16 | 2 | 16 | 0.00044 | 0.00494 |
| 17 | 2 | 16 | 0.00044 | 0.00494 |
| 18 | 1 | 8 | 0.00022 | 0.00269 |
| 25 | 1 | 8 | 0.00022 | 0.00269 |
| 33 | 1 | 12 | 0.00022 | 0.00269 |
| 38 | 1 | 12 | 0.00022 | 0.00269 |
| 92 | 1 | 12 | 0.00022 | 0.00269 |
| 98 | 1 | 12 | 0.00022 | 0.00269 |
| 109 | 1 | 12 | 0.00022 | 0.00269 |
| 130 | 1 | 12 | 0.00022 | 0.00269 |
| 131 | 1 | 12 | 0.00022 | 0.00269 |
| 132 | 1 | 12 | 0.00022 | 0.00269 |
| 176 | 1 | 12 | 0.00022 | 0.00269 |
| 271 | 1 | 12 | 0.00022 | 0.00269 |
| 286 | 12 | 144 | 0.00266 | 0.02274 |
| **Entropy:** | **1.67951 bits/symbol** | | | |

**Figure 9**

Example of LZ1_FAST coding efficiency analysis.

[8] Ehud D. Karnin, "Evaluation and Enhancement of LZ-1 Based Data Compression Systems," IBM Science and Technology, Haifa Research Group, draft, 1991.

| Bucket prefix | Bucket size | Code words |
|---|---|---|
| 0 | 2 | 2 .. 3 |
| 10 | 4 | 4 .. 7 |
| 110 | 8 | 8 .. 15 |
| 1110 | 16 | 16 .. 31 |
| 11110 | 32 | 32 .. 63 |
| 111110 | 64 | 64 .. 127 |
| 1111110 | 128 | 128 .. 255 |
| 11111110 | 256 | 256 .. 511 |
| ... | 512 | 512 .. 1023 |
| | 1024 | 1024 .. 2047 |
| | ... | ..... |

**Figure 10**

K_code length and control fields.

| Bucket prefix | Bucket size | Code words |
|---|---|---|
| 0 | 2 | 00 .. 01 |
| 10 | 4 | 100 .. 111 |
| 110 | 8 | 110000 .. 110111 |
| 1110 | 16 | 11100000 .. 11101111 |
| 1111 | 256 | 111100000000 .. 111111111111 |

**Five code buckets are used: the prefixes are 0, 10, 110, 1110, and 1111; the numbers of *code words*, respectively, are 2, 4, 8, 16, and 256 in the five buckets.**

**Figure 11**

Length and control fields of the 286LZ1 algorithm.

The second part of the second observation led us to significant coding simplification and fast hardware operation. The observation suggested that all K_code words of length greater than 27 could be lumped into a single bucket. The resulting 286LZ1 algorithm has 286 code words in five buckets; 270 of them are used for the length description from 2 to 271, and 16 of them are assigned for controls and end of file.

Since there are only five buckets of code words, the encoding and decoding can be sped up by precomputing all possible code lengths. For instance, since the parser is on the speed-critical path for decoding, we can compute all five possible length variations and archive multibyte decompression in a single machine cycle.

There were questions raised regarding the theoretical reasoning for the second observation above. More analyses can be made to classify the distribution. A somewhat relevant study is the paper by Cleary and Witten on partial string matching [30]. The experimental results showed the optimal model order for compressing text, program, numeric data, binary code, gray-scale image, and so on.

- *IBMLZ1 compression results*

**Figure 12** shows the compress-to ratios of IBMLZ1 variants. The IBM-1K and IBM-2K variants result from the use of 1K and 2K history buffers, respectively. The IBM-1KF and IBM-2KF are the results from fast-attack [28] versions that aimed at improving the initial coding efficiency when the history buffer is partially filled. The block size denotes the size limit at which the compression is restarted. The blocking effect appears on the channel to the storage controller, where 4K is the most typical block size and the next most common block size is 2K. The CAM of a 1K history buffer is about 60K equivalent-cell area, suggesting that a 1K CAM instead of 2K will yield good compression while minimizing circuit size.

## IBMLZ1 compression technology for storage controllers

A family of compression chips based on the IBMLZ1 compression algorithm has been developed. An 0.8-$\mu$m 75 000-gate array achieved 40MB/s throughput [31]; a 0.5-$\mu$m version is expected to reach 50MB/s throughput. **Figure 13** depicts a comprehensive compression subsystem chip with the compression macro imbedded. The chip operates at 40 MB/s and is pipelined between the ESCON and the cache unit for the array storage controller. As mentioned earlier, pipelined operation yields the maximal system benefit.

Log-structured storage management provides efficient integration of compression for the storage controller. In CKD (count key data) environments, individual blocks of data are frequently updated. Since the compression effect is data-dependent, the newly compressed data cannot be guaranteed to fit in the space left by the old data. The log-structured technique for dealing with this unpredictability is not to attempt updates in place, but rather to collect changed data in a log and write the data to DASD in free space, maintaining a directory which maps the logical address of the data to the actual physical location. The directory can then be reviewed periodically to find sparsely populated areas on disk and collect the space for reuse.

## Summary and remarks on future development

The IBMLZ1 algorithm and technology have been designed for high-compression/decompression throughput with efficient hardware implementation, high reliability, low system overhead, and robust compression. Data integrity and reliability are ensured by coupled compression–decompression checking, the scrubbing operation, and extensive built-in checking. Extremely low ($CPB = 1$) compression and decompression have been achieved. The extremely high-compression/decompression throughput of 30–50 MB/s allows a transparent mode of operation and, thus, minimal system overhead. The IBMLZ1 algorithm compresses well over the VM, MVS, RS/6000, and PC test cases. Future tasks include developing format-compatible lower-*CPB* and low-overhead data integrity checking architectures.

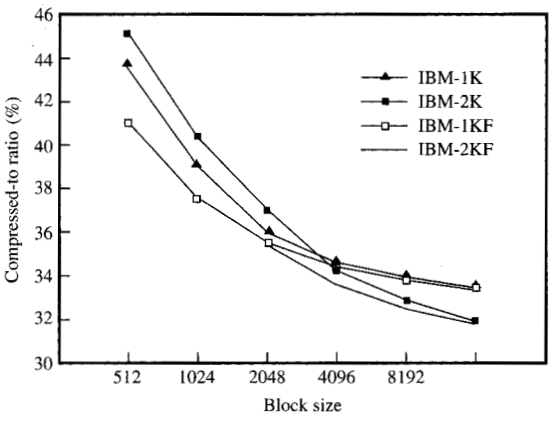


**Figure 12**

Compressed-to ratios vs. input block size over 44 test cases.

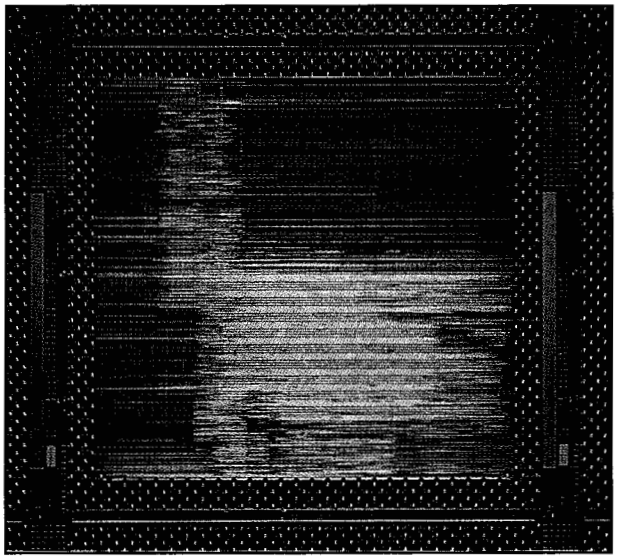

**Figure 13**

IBMLZ1 compression chip for disk array controller.


## Acknowledgments

The results reported in this paper could not have been achieved without the vision and leadership of T. R. Lattrell (IBM Microelectronics Division); the technical leadership of Ehud Karnin; the extensive coding performance analysis and verification by Larry Garibay and Mayank Patel; high-performance physical design by Kenneth Gray and Michael Digby; project leadership by T. H. Lee; outstanding data flow architecturing, design, verification, test generation, and bring-up efforts by Gary K. Chan, Rudy Farmer, Greg B. Bishop, Starla Miller, Sonny Nguyen, Yancy Cheng, Lisa Wang, and Steven Lee; and extended design automation support and consultation by Hugh McDevitt, Mitchell Tidwell, James Clark, Brad Peterson, Barinder Nijjar, and Catherine Chow. We wish to express our warm appreciation to them all and to those we might have overlooked, and our special thanks to IBM Fellow Arvind Patel for his in-depth guidance on CRC aspects and analysis, and to IBM Fellow James Brady for his assistance with control system performance aspects. The authors would like to express their gratitude to Dr. Glen G. Langdon, Professor at UC Santa Cruz and the developer of BAC, FileCOMP, ICOM, SUNSET, and many other key compression algorithms and systems, for his long-term guidance with respect to compression algorithms.




## References and notes


1. Mayank Patel, Neil MacLean, and Glen G. Langdon, Jr., "An Economical Hardware-Oriented High-Speed Data Compression Scheme," *Research Report RJ-9170*, IBM Research Laboratory, San Jose, CA, January 11, 1993.
2. Glen G. Langdon, Joe-Ming Cheng, Ronald B. Arps, and Patrick E. Mantey, "Hardware-Optimized Compression and the Skew Coder LSI Chip," described the BAC chip (completed in 1982) and the research efforts on three applications: file, binary image, and gray-scale compression, 1983–1984; *Research Report RJ-8611*, IBM Research Laboratory, San Jose, CA, February 6, 1992.
3. David A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE* **40**, 1098–1101 (1952).

4. J. J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding," *IBM J. Res. Develop.* **20**, No. 3, 198–203 (1976).
5. Richard C. Pasco, "Source Coding Algorithm for Fast Data Compression," Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, CA, 1976.
6. Glen G. Langdon, Jr., "An Introduction to Arithmetic Coding," *IBM J. Res. Develop.* **28**, No. 2, 135–149 (March 1984).
7. Robert G. Gallager, "Variations on a Theme by Huffman," *IEEE Trans. Info. Theory* **IT-24**, No. 6, 668–674 (November 1978).
8. Richard W. Hamming, *Coding and Information Theory*, ISBN 0-13-139139-9, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1980.
9. J. Rissanen and G. G. Langdon, Jr., "Arithmetic Coding," *IBM J. Res. Develop.* **23**, No. 2, 149–162 (March 1979).
10. Jorma Rissanen and Glen G. Langdon, "Universal Modeling and Coding," *IEEE Trans. Info. Theory* **IT-27**, No. 1, 12–23 (January 1981).
11. Ian H. Witten, Radford M. Neal, and John G. Cleary, "Arithmetic Coding for Data Compression," *Commun. ACM* **30**, No. 6, 520–540 (June 1987).
12. Joe-Ming Cheng and Glen G. Langdon, "Image Compression with QM-AYA Adaptive Binary Arithmetic Coder," *SPIE Proc.* **1771** (*Application of Digital Image Processing XV*), 413–423 (1992).
13. N. Abramson, *Information Theory and Coding*, McGraw-Hill Book Co., Inc., New York, 1963, pp. 61–62.
14. R. Morris, "Counting Large Numbers of Events in Small Registers," *Commun. ACM* **21**, 840–842 (1978).
15. Philippe Flajolet, "Approximate Counting: A Detailed Analysis," *BIT* **25**, 113–134 (1985).
16. Daniel R. Helman, Glen G. Langdon, N. Martin, and Stephen Todd, "Statistics Collection for Compression Coding with Randomizing Feature," *IBM Tech. Disclosure Bull.* **24**, 4917 (1982).
17. Glen G. Langdon and Jorma Rissanen, "Compression of Black–White Images with Arithmetic Coding," *IEEE Trans. Commun.* **COM-29**, 858–867 (June 1981).
18. W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr., and R. B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM J. Res. Develop.* **32**, No. 6, 717–726 (November 1988). W. B. Pennebaker and J. L. Mitchell, "Probability Estimation for the Q-Coder," *IBM J. Res. Develop.* **32**, No. 6, 737–752 (November 1988).
19. R. B. Arps, T. K. Truong, D. J. Lu, R. C. Pascoe, and T. D. Friedman, "A Multi-Purpose VLSI Chip for Adaptive Data Compression of Bilevel Images," *IBM J. Res. Develop.* **32**, No. 6, 775–795 (November 1988).
20. Joe-Ming Cheng and Glen G. Langdon, "Modified Metropolis Annealing Algorithm for QM-AYA Arithmetic Coder Design Optimization," *SPIE Proc.* **2028** (*Application of Digital Image Processing*), 2–12 (1993).
21. Jacob Ziv and Abraham Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Info. Theory* **IT-23**, No. 3, 337–343 (May 1977).
22. Jacob Ziv and Abraham Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Trans. Info. Theory* **IT-24**, No. 5, 530–536 (September 1978).
23. Rory D. Jackson and Willi K. Rackl (both from the IBM Poughkeepsie Laboratory), "Data Expansion Apparatus," essentially described the LZ1; patent filed on June 30, 1976; U.S. Patent 4,054,951, October 18, 1977.
24. Victor S. Miller and Mark N. Wegman, "Variation on a Theme by Ziv and Lempel," *Research Report RC-10630*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, July 31, 1984.
25. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, "String Matching," *Introduction to Algorithms*, McGraw-Hill Book Co., Inc., New York, 1990, Ch. 34.
26. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, "Radix Sort," *Introduction to Algorithms*, McGraw-Hill Book Co., Inc., New York, 1990, Ch. 9.
27. Joe-Ming Cheng and Yancy L. Cheng, "A Method and Means for Character String Pattern Matching for Compression and the Like Using Minimal Cycle per Character," U.S. Patent 5,525,982, June 11, 1996.
28. Joe-Ming Cheng, Ehud D. Karnin, David J. Craft, and Larry J. Garibay, "Effective 286 LZ1/JR Compression Coding Format for Hardware and Software and the Fast Attack Option," SA9-94-061, IBM filed U.S. patent application, 1995.
29. Glen G. Langdon, "Further Development in Lossless Gray-Scale Image Compression," presented at the IBM Conference on Pattern Recognition and Image Processing, Session C1, IBM Thomas J. Watson Research Center, November 14, 1984; also known as the "SUNSET algorithm," reprinted as *IBM Research Report RJ-6426*, September 1988.
30. J. G. Cleary and I. H. Witten, "Data Compression Using Adaptive Coding and Partial String Matching," *IEEE Trans. Commun.* **COM-32,** No. 4, 396–402 (April 1984).
31. Ted Lattrell, "40 MB/sec Throughput IBM Compression Chip and Macro Announcement," *EE Times*, front page, August 16, 1993.

*Received June 15, 1995; accepted for publication August 5, 1996*

**Joe-Ming Cheng** *IBM Storage Systems Division, 5600 Cottle Road, San Jose, California 95193 (joeming@vnet.ibm.com).* Dr. Cheng received a B.S. in physics from CYU, Taiwan, in 1971; the Air Force Teaching Credential, Taiwan, in 1973; an M.S. in (electronics) scientific instrumentation from the University of California at Santa Barbara in 1975; and a Ph.D. in computer engineering from UC Santa Cruz in 1996. Dr. Cheng was an LSI tester designer at Macrodata Inc. and a missile system design lead at Teledyne Systems. He joined IBM Research (San Jose) in 1978. Dr. Cheng designed the first binary arithmetic compression chip at IBM Research and the image compression system board. With Professor Glen G. Langdon, he developed the modified Metropolis simulated annealing algorithm for probability estimation state machine optimization, which achieved the highest-scored JPEG and JBIG QM coding; and an integrated Huffman-arithmetic code called AMSAC (Approximated Multi-Symbol Arithmetic Coding). Dr. Cheng also developed adaptive AMSAC coding, extended Huffman coding redundancy-bound formulations, and formulated the AMSAC coding redundancy bounds. He worked on related projects at the IBM Zurich Research Laboratory. Since 1991, Dr. Cheng has led compression-algorithm and chip-development teams at the IBM Storage Systems Division in San Jose. He has received three divisional awards, a first-level invention award, an Outstanding Technical Achievement Award, and an Outstanding Innovation Award.

**Linda M. Duyanovich** *MatriDigm Corporation, 47207 Bayside Parkway, Fremont, California 94538.* Ms. Duyanovich received her B.S. in mathematics from California State University, Stanislaus, in 1976, and her M.S. in operations research from Stanford University in 1978. She joined the IBM General Products Division in 1978, and spent several years involved in software and hardware performance and modeling. While a member of the hardware performance modeling group, she designed and led the implementation of DCAT, a disk subsystem modeling tool in use by IBM Marketing which utilizes performance information from existing systems to predict improvements that could be gained with IBM's newest products. In 1991, Ms. Duyanovich became manager of the Future Products Architecture and Design team, which completed a new architecture and high-level design for high-performance, large-capacity I/O subsystems. These subsystems were designed to support compression, and used Dr. Cheng's algorithm and chip. Ms. Duyanovich then worked as the leader of a cross-functional team involved in strategy and new product proposals. She left IBM in 1996, and now works for MatriDigm Corporation as executive assistant to the CEO and manager of Beta Projects.

**David J. Craft** *IBM Microelectronics Division, 11400 Burnet Road, Austin, Texas 78758 (dunstan@vnet.ibm.com).* Mr. Craft is a senior engineer/scientist, currently working in the areas of advanced ASIC chip architectures, data compression subsystems design, and compression algorithms. He received a B.S. in physics from the Imperial College, London University (U.K.), in 1963 and joined IBM in 1965. He has worked on many advanced development projects, in IBM laboratories in Hursley (U.K.), until 1978, and then subsequently in Boulder, Tucson, and Austin (U.S.). Mr. Craft holds 15 issued patents and has 10 more recent applications currently in process. He is a recipient of two IBM Special Contribution Awards and an IBM Outstanding Technical Achievement Award, the latter for his work on extremely fast hardware data compression designs. One 1974 patent (U.S. 3,818,447), on a serial bus arbitration mechanism, is now the basis for the arbitration in the PC industry "Plug and Play" standard, and also the J1850 and CAN serial bus protocols, now very widely used in the automotive, industrial, and consumer electronics industries.