Fast and Highly Reliable IBMLZ1 Compression Chip and Algorithm for Storage

Joe-Ming Cheng and Linda M. Duyanovich

IBM Storage Systems Division and Micro Electronics Division

Fast and Highly Reliable IBMLZ1 Compression Chip and Algorithm for Storage

- Overview: Fast and Highly Reliable Compression for Storage Systems
- DASD Compression Attributes and Objectives
- Effects of Compression Errors and Special Handling
- Fast Hardware Pattern Matching, 1Byte/Clock, 2Bytes/Clock, ..
- New Observations for Simpler Compression Algorithm (Accepted as QIC 154 Standard)
- Summary: IBMLZ1 Compression System on a Chip

Fast and Highly Reliable Compression for Storage Systems

Tape Compression and IDRC [1]

DASD (DISK) Controller Compression and IBMLZ[2,3]

- Lossless and General-Purpose Compression
- Simple Algorithm Format and Robust
- Optimized for High-Thruput Hardware Execution: 1Byte/Cycle, 40MH/sec.
- Algorithm Accepted as QIC 154 Standard

1 IDRC, Improved Data Recording Capability, 0.8u, 2.5MB/sec, 1990

2 Fast Pattern Matching Data Flow, J. Cheng and Y. Cheng, IBM Filed patent application.

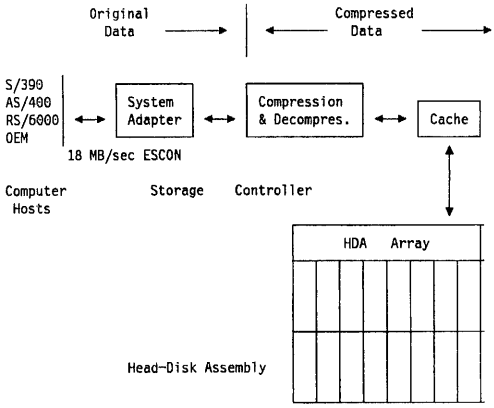
3 IBMLZ1 Algorithm, J. Cheng, E. Karnin, D. Craft, and Larry J. Garibay, IBM Filed Patent application.

DASD Compression Attributes and Objectives

- Speed Aspects
  - Speed Challenge
  - Broad System Performance Benefits
- Reliability
  - Compression Amplifies Errors
  - Resolutions
- Latency
- Robustness in Compression
  - Test Suite
- Effective on Small Blocks
- 1991, Top-speed compression chips: 2.5MB/sec, 1.5MB/sec, 1.2MB/sec.
- Early Objectives: 16X speed-up, 80% cost reduction

Compression Benefits for Storage System

- *Compression* x *Compaction* = 4.5 X - 6.0 X
- System Performance



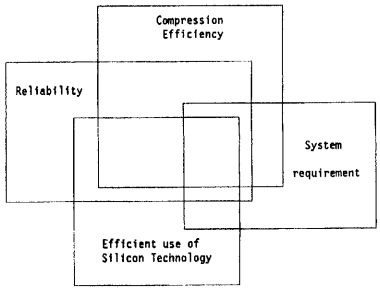


Figure 2. IBMLZ1 Compression Technology Development Objectives.

Figure 1. Storage Controller with Compression

Compression and Decompression

Figure 3. Boat1 in Halftone

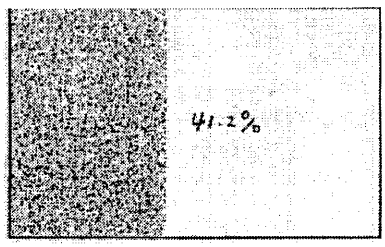


Figure 4. Compressed Boat1

[4]

[4] LanRis81, Glen G. Langdon, Jr. and Jorma Rissanen, "Compression of Black-White Images with Arithmetic Coding," Volume COM-29, No. 6, June 1981, IEEE Transaction on Communications.

Effects of Compression Errors

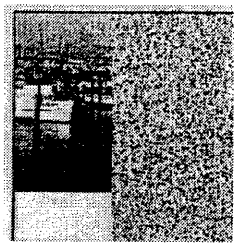


Figure 5. Decompressed Boat 4

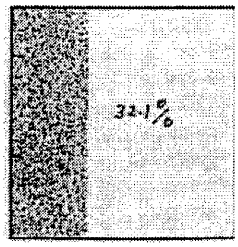


Figure 6. Compressed Boat 4

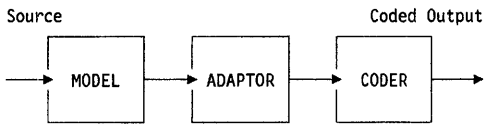
IBMLZ1 Algorithm and Hardware Development Paradigm

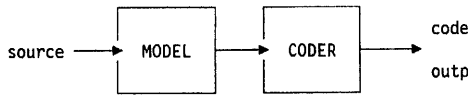
Extremely High Reliability	One undetected Error in 10^{30}
Efficiencies	Robust Compression, Speed, Latency
System Requirement	Intra-Parallel instead Inter-Parallel, good Compression, Speed
Silicon Technology	Regularity

Compression Algorithm Overview

Model Unit Use of Context, Pattern matching removing redundancy (correlation), xmp: Lempel-Ziv history structure on text.

Coding Unit Assigned code word or augend; desired codelengths to be as close approximation to source entropy (neg. inverse probability log).





Compression can take place in two functional stages:

Compression Algorithms

LZ (LZ1, LZ2, and LZW) Jacob Ziv and Abraham Lempel[5][7][8]

Arithmetic Coder Jorma J. Rissanen and Rich C. Pasco

Inherent Algorithmic Speed

Algorithm-Hardware	Symbol Base	Speed
LZ1	Byte, Char	1
Bi-Level Arithmetic Coder (BAC, QCoder chips)	Bit	1 / 8

5 LZ1: Ziv and Lempel, "A Universal Algorithm for Sequential Data Compression", IEEE Transaction on Information Theory, May 1977. pp. 337-343.

6 JacRac76, Rory D. Jackson and Willi K. Rackl, IBM, "Data Expansion Apparatus," filed June 30, 1976; US Patent 4,054,951, Oct. 18, 1977.

7 LZ2: Ziv and Lempel, "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transaction on Information Theory, September 1978, pp. 530-536.

8 LZW: Terry A. Welch, "A Technique for High-Performance Data Compression" IEEE Computer, June 1984, pp. 8-19.

String (Pattern) Matching Algorithms

Detecting a pattern, P, in another text string, T.

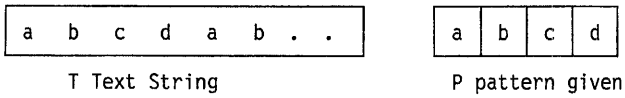


Figure 7. Pattern Matching of String. T, text string; P, fixed length pattern.

Generic Brut-Forced Straightforward

Knuth-Morris-Pratt [*KnMoPr77*], "Fast Pattern Matching in Strings ,"[9]

Boyer-Moore [*BoyMoo77*], "A Fast String Searching Algorithm ,"[10][11]

[9] [*KnMoPr77*] Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt, "Fast Pattern Matching in Strings ," *SIAM J. Computing* , Volume 6, No. 2, PP. 323 - 350, June 1977. IDRC, Improved Data Recording Capability, 0.8u, 2.5MB/sec, 1990

[10] [*BoyMoo77*] Robert S. Boyer and K Strother Moore, "A Fast String Searching Algorithm ," *Communication ACM* , Volume 20, No. 10, PP. 762 - 772, October 1977.

[11] [*Smit82*] G. de Smit, "A Comparison of Three String Matching Algorithms ," *Software - Practice and Experience* , QA76.5.A1S653, volume 12, no. 1, PP. 57 - 66, January 1982.

LZ1 and Hardware-Oriented Pattern Matching Aspects

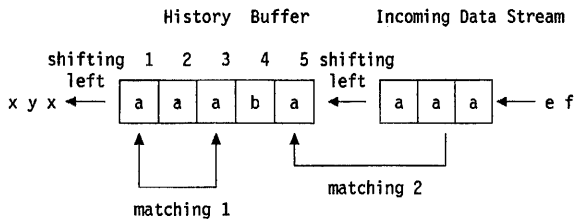


Figure 8. Pattern Matching of String..

Differences in Pattern Matching Paradigm

- P is open-ended
- Hardware accounting: cycle vs. step
- Boundary condition

Prefix-Extension Pattern Matching for Parallel Hardware

```
    loop until done {
0       read input (byte) and make copy, global match search and write
          this byte into CAM pointed by a write select register.
1       check (condition)
2       condition: (string count is 0 and no match)
          ship raw byte out
3       condition: (string count is 0 and at least one match found)
          copy match to PS, increment string count, shift (rotate) PS ;
          /* could be the 1st char of a new match string */
4       condition: (string count >= 1 and ANY_PS is 1 )
          increment string count, shift (rotate) PS /* match gets longer */
5       condition: (string count >=2 and ANY_PS is 0)  /* current match ends */
          release COMPRESSED_TOKEN;
6           if (no match)
              ship raw byte, reset string count to 0,
7           if (match)    /* possible back-to-back match case */
              copy match to PS, increment string count, shift (rotate) PS
8       condition: string count ==1 and no ANY_PS
9           ship previous raw byte out
10          if (no match)
              ship present raw byte out,
11          else
              copy match to PS, increment string count, shift (rotate) PS ;
              /* could be the 1st char of a new match string */
12      condition end, return to 0;
    }
```

Figure 9. LZ1 Concurrent only Finite State Machine Control

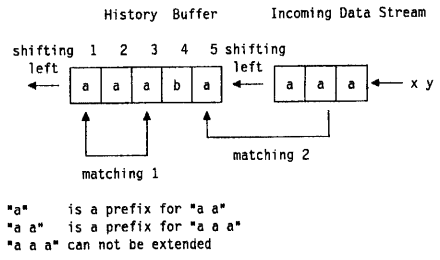


Figure 10. Prefix Extension.

- Prefix Extension plus Broadcast (CAM) Figure 10: P = "aaaxy..": "a", a proper prefix, extendable to "aa"; "aa", a proper prefix, extendable to "aaa"; "aaa", not be extendable, "aaa" is the longest matching.

Prefix-Extension Pattern Matching for Parallel Hardware (cont.)

CDF Micro-Coded Control Flow

1. Run prefix extension starting on every history buffer location.
2. Use BROADCAST, repeat prefix extension if at least one prefix can be extended.

- Moving pointers instead of data
- CAM, Content Addressable Memory, for exhaustive search
- 4X IDRC Speed

Compression Throughput	1 Byte per major cycle
Major Cycle	Variable 3-5 minor cycles
Minor Cycle	25 ns CMOS 4L

Look-Ahead Pattern Matching for Parallel Hardware

Desired Properties: Continuous Pipelined Flow

Intra-Parallel not Inter-Parallel — To preserving correlation: better compression and less system overhead

Pipelining Without Stalls — Avoid Data (no DDG loop), Control, and Structure Hazards

Simplified Data Dependance — No state machine control; Major cycle = Minor Cycle; 16X IDRC.

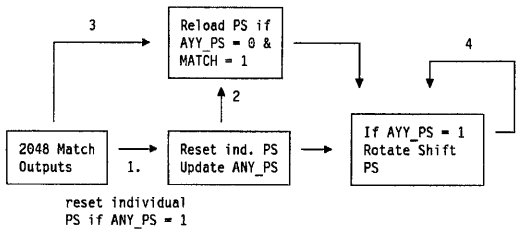


Figure 11. Data Dependance Graph of CDF Critical Data Flow. DDG is Three-Colorable, 4-5 minor cycles/major cycle data flow.

IBMLZ1 Algorithm Coding Example

History buffer size = 15

```
History Buffer   Input Char
000000000000000  t
00000000000000t  h
0000000000000th  e
000000000000the  r
00000000000ther  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,
```

IBMLZ1 Coding Efficiency Analysis

```
/*********************************************************/
/**   LZ1_FA:  Fast LZ1 with Compression Performance    */
/**     Analysis.          Joe-Ming Cheng  5-12-91      */
/*********************************************************/
.. msg_ ..          The time is : Sat May 25 08:08:40 1991

.. MLen  MCount  Coded_Bit
    0     738     6642
    1    7623    68607

.. Matched symbols are:   4514

    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
Coded bit length:    10854 average bit used:  2.40452
IBMLZ1 coding inefficiency:    43.1683 %
```


New Observations

Observation 1 Limited match length to 192 has less than .5 % compression loss.

- Reduced latency
- Preserved pipeline flow (PCDF).

Observation 2 Match Length distribution: One-Sided Exponentail till 26 - 28, then appeared to be random.

- Simpler Code Book
- Conducive to Faster Decompression/Compression

Effects of Compression Errors

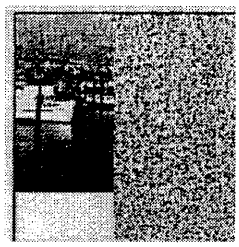


Figure 12. Decompressed Boat 4

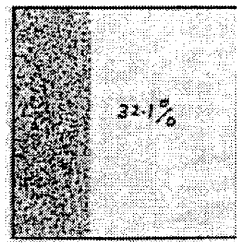


Figure 13. Compressed Boat 4

High Reliability

Compression Functional Checkings

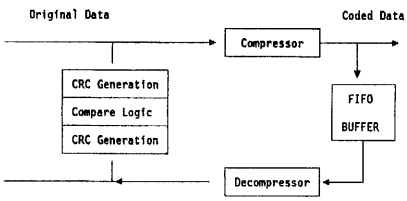


Figure 14. Compression and Decompression Pair for Encoding Reliability.

- Undetected Error: 1 in 10^{30}
  1. Integrated Source and ECC coding
  2. Hardware Duplication
  3. Compression and Decompression Coupling Figure 14
- Compression Paths Checking
- Used two-dimensional Checkings.

High Reliability (cont.)

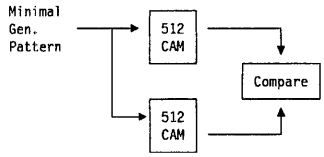


Figure 15. CAM Self-Checking for Reliability.

Compression CAM Checking

- Compression Degradation Check
- 768 Vector Self-Scrubbing

Chip and Logic Checkings

- Scan-Path
- Build-In Self Test
- IEEE Boundary Scan

IBMLZ1 Compression Ratios

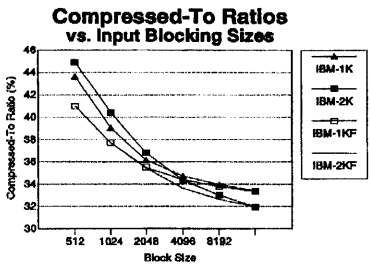


Figure 16. Compressed-To Ratios over 44 Test Cases.

- Suite of 44 Test Cases
  1. 38 Host applications
  2. 6 PC and workstation applications.
- Fast Attack not implemented at present

IBMLZ1 Compression Chips

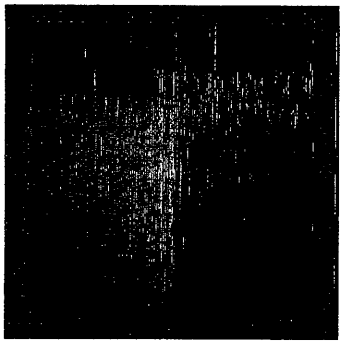


Figure 17. IBMLZ1 COMP Chip Photo
COMP2-40

IBMLZ1 Algorithm
Simplex Compression and Decompression
Fixed 40MB/sec Compress/Decompress rate
1 undetected error in 10 EXP 30
Full Test Access Port (TAP) function

153,279 cells used on 182K image (10.45mm x 10.15mm)
44.5k logic, 2k CLOCK, 9k TAP;
CAM 64,000; GRAM 21,550 (1Kx9) and 12,448 (256x18)
176 pins: OCR 69; OCD 70; VDD/GND 24;
Programmable VDD/GND 13;
Power: 250mW measured (1.19 W estimated)
4,500 Test Cases: Base Patterns, 27 Regular,
55 Look Ahead, 3 complete files; 165 Test Types;
and 50 delay runs and 27 TAP delay mode test cases.

Concluding Remarks

- Equal or slightly better compression (five algorithms)
- Considerable less hardware complexities in certain areas: Code Book (12.5%), No State Machine control
- Fast Hardware Pattern Matching, 1Byte/Clock, 2Bytes/Clock, ..
- Highly Reliable: Six levels of checkings
- Algorithm Based on New Observations: lower latency, lower complexity, faster decoding
- Algorithm Accepted as QIC 154 Standard

J. M. Cheng
IBM Storage Systems Division

L. M. Duyanovich
IBM Storage Systems Division

Fast and Highly Reliable IBMLZ1 Compression Chip and Algorithm for Storage

Data compression allows more efficient use of storage media and communication bandwidth. Standard compression offerings for tape storage have been well-established since the late 1980s. The compression technology lowers the cost of storage without changes to any 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 technology.

The IBMLZ1 compression algorithm was designed not only for robust and highly efficient compression, but also for extremely high reliability. As compression removes redundancy in the source, the compressed data becomes extremely vulnerable to data corruption. Key design objectives for the IBMLZ1 development were: efficient hardware execution and efficient use of silicon technology; and minimum system integration overhead. Through new observations of pattern matching match-length distribution and use of graph vertex coloring for evaluating data flows, the IBMLZ1 compression algorithm and technology achieved the above objectives.

Compression Objectives for Storage Controller

The addition of compression capability to the DASD subsystem facilitates more efficient use of subsystem resources: cache, data path bandwidth, and disk capacity in a *transparent* manner to the system storing the data. Compression allows existing platforms and applications to benefit from a lower storage cost and potentially higher performance without any 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 speed (18 MB/sec for ESCON [1]), and the ability to *pipeline* data through the compressor without significant *store-and-forward* penalties.

In addition to the obvious benefits in disk capacity (IBMLZ1 typically achieved better than a 3:1 savings in DASD capacity for high-end systems), there are additional benefits to the subsystem. If the data is compressed as it enters the subsystem, see Fig. 1, the cache resource has effectively been tripled. Customers can benefit either 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 the reduced utilization of internal buses and DASD paths as data flows 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 18 MB/sec 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.

---

1 ESCON, Enterprise Systems Connection I/O Architecture, defines full duplex architecture between channels and control units, the maximum peak data rate is 19.62 MB/sec.

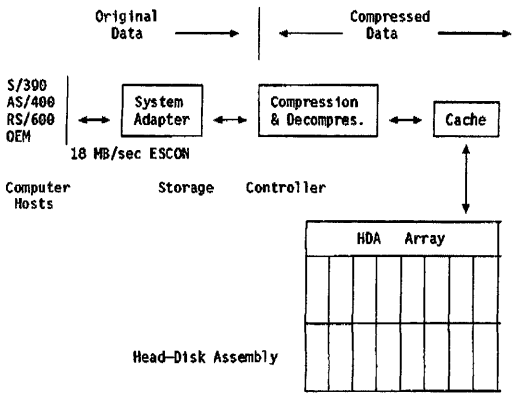


Figure 1. Storage Controller with Compression Capability.

The development of a new compression algorithm and technology was necessary for the direct *pipeline* support of 18 MB/sec ESCON, up to 40 MB/sec for Tape, and for the emerging higher speed communication protocols. The core of the 3480/3490 IDRC [2] compression algorithm is the FileCOMP file compression model and the BAC [3] compression chip as engine [*LCAM*92]. The FileCOMP and the BAC were developed between 1980 and 1983. The largest gate-array sizes for CMOS technology available in 1981 was 3K to 4K. That 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.

Compression Algorithm and System Overview

A compression system, Fig. 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 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, transform coding, and so on. 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 *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 unit and Coding unit will 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.

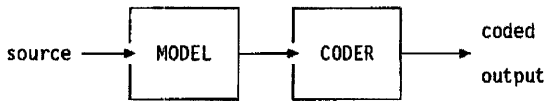


Figure 2. Compression System with a Model unit and a Coding unit.

Two common choices for the Coder unit are Huffman coding [*Huffman*52] and Arithmetic coding [*Rissa*76] [*Pasco*76] [*Lang*84*a*]. 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 [*Gallag*78], and can be uniquely decoded. The worst case redundancy, defined as expected code length less the binary *Shannon's Entropy*[4] [*Hamming*80], occurs when the most probable *symbol* probability is greater than 50%. The case occurs quite often in compressing black-and-white image. The 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* [4] or information, and thus achieves very low redundancy. The arithmetic coding method can be thought of as a generalization of the Huffman coding without the need for prefix codes, or integer-length *code words* . The Huffman code, however, remains as the most popular encoding method for its simplicity and its effectiveness in general.

Arithmetic coding encourages [*RiLa*79] [*RiLa*81] [*WiNeCl*87] a clear separation between the MODEL

---

[2] IDRC, Improved Data Recording Capability [*PaMcLLa*93]. IDRC is a data compression and compaction standard.

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

[4] Entropy, defines the average amount of information, Summation over i $p_i \times \log_2 \frac{1}{p_i}$ (for binary system). $\log_2 (\frac{1}{p_i})$ 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 symbol '0' and '1' appear 93.75% and 6.25% of the time respectively, the optimal code length for '0' and '1' are .093 bit and 4 bits respectively.

and the CODING unit, 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 [*CheLan*92].

Shannon conceived the notion of arithmetic coding, and in [*Elias*63] the method was made symbolwise recursive. The developments of the full arithmetic coding technique 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 [Mor*78] [*LCAM*92] [*Fla*85] and probability estimation [*HLMT*82]; multiplication and division *operator strength reduction to fixed augend* based addition and subtraction; and a *simplified common adaptive mechanism* shared among large possible contexts for adaptation resulted more than 50 to 1 hardware reduction and software speed-up and as well! The BAC chip, developed in 1981-1982 [*LCAM*92], and the black-white image compression system (ICOM) [*LaRi*81], both benefited from the drastic complexity reduction. Further major computational reduction which effected both hardware and software was the Q-Coder algorithm [*PeMi*88]. The ABIC [*ATLPF*88] 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 simulated annealing method for the automated optimization of the arithmetic coder probability-estimation table [*CheLan*93].

LZ1 and LZ2 Compression Algorithm Review

The LZ1 and the LZ2 compression algorithms are commonly regarded as Lempel and Ziv's compression algorithm 1 [*ZL*77] and algorithm 2 [*ZL*78]. The LZ1 and the LZ2, in their original form, expressed the notion of 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 LZ1 data structures and operations of the compression algorithm are:

1. Construct a history structure of the past stream or of the recent past stream;
2. Use pattern matching to find the maximal incoming sub-stream residing in the history structure;
3. Substitute the incoming sub-stream with a pointer into the history structure and length (or their equivalents).
4. If the coded description has shorter length than the original sub-string, use the coded form.

Conceptually, the history structure of the LZ1 algorithm is a fix-sized sliding window. One can consider it as a shift register of fixed length, which contains the recent past symbols. Fig. 3 shows maximal (length) matches of the incoming sub-string found in two locations. Match number 1 starts at location 1; and 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" also found in the history buffer, the alternative coding form is to use three bits denoting the starting address, another 3 bits for maximal matching length, and 1 bit to tell 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 field, and 3-bit offset). 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.

The LZ2, Lempel and Ziv's compression algorithm 2, [*ZL*78] [*MiWa*84] normally has a tree-structured history structure for improved search for matches in software.

---

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

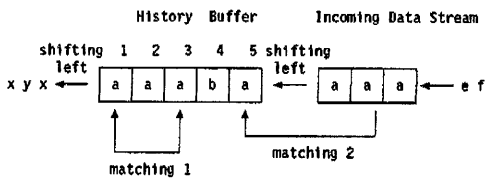


Figure 3. Pattern Matching of String. Two matchings of the maximal incoming substring "aaa" are indicated in the above five-element shift register History buffer.

In 1976, Jackson and Rackl of IBM Fishkill described "Data Expansion Apparatus," where in a long data stream, the repeated sections of data can be saved by the substitution of *tag, address, length, and the number of repetitions* for storage space saving. The idea is essentially the LZ1. It was filed for patent in 1976, and obtained the US Patent in 1977 [*JacRac*76]. In the patent description, some flavors of the LZ2 were also described.

Hardware Flow Speed-Up

Fast LZ1 hardware organization is normally based on two themes: the use of *prefix-extension* [*CoLeRi*90*S*], and the use of very powerful *parallel combinatorial operators*. For instance, the longest match in Fig. 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 a smaller job of searching for a one byte, two byte, 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, or, "aaa" can not be further extended. *Prefix-extension* then concludes that the prefix "aaa" itself is the longest match. For the powerful *parallel combinatorial operators* aspect, arrays of massive parallel comparators are used for the high-speed LZ1 hardware organization. The prefix of one byte or multiple bytes are simultaneously compared against all storage locations in the history buffer. The above organization resembles Content Addressable Memory's function, hence is often referred to as CAM.

```
History Buffer  Input Char

000000000000000 t
0000000000000t h   note: t is in match point 0
0000000000000th e   note: t is still in match point 0
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,
```

```
History Buffer  Input Char
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.

It is interesting to note that the use of prefix-extension and the use of parallel compare operators resembles the algorithmic notion of Radix sort [*CoLeRi*90*R*]. As in any two-input comparison based sorting algorithm the running time is bounded lower by the information theoretic $n \log n$, 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 runtime 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 bring the CPB, machine cycles needed for compression/decompressing a byte, down to 1 and below.

IBMLZ1 Algorithm and Technology Development

In addition to its objectives of robust and efficient compression, the IBMLZ1 compression algorithm was designed for reliability as well. The objectives, Fig. 5, for the IBMLZ1 compression algorithm established at the onset were:

1. Extremely High Reliability: one undetected error in 10^{30}. As compression removes *redundancy* in the source, the compressed data becomes extremely vulnerable to data corruption. A single bit error in the coded stream could essentially result in a total decoding failure from that bit location and onward.
2. Hardware Execution Efficiency: the hardware architecture should use as few machine cycles to compress or decompress a byte as possible. The architecture should maintain low complexity and use the silicon technology effectively. In addition, it is desired to have a moderate complexity growth as the CPB[6] gets small.
3. Robust Compression: achieve good coding efficiency for broad applications.
4. Minimal System Integration Overhead: the maximum benefit from compression is achieved when the compression can be performed without performance loss. It requires a technology which is capable of running at channel speed (18 MB/sec for ESCON [1]), and the ability to *pipeline* data through the compressor without significant *store-and-forward* penalties.

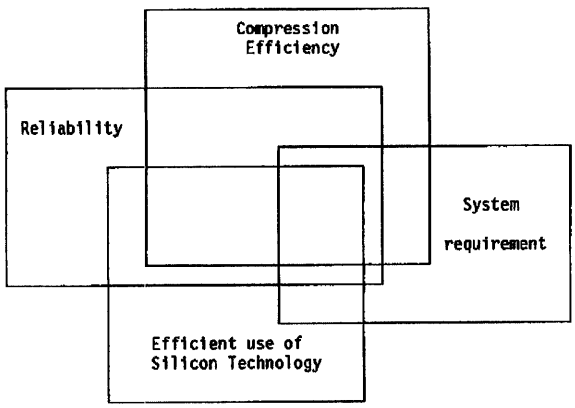


Figure 5. IBMLZ1 Compression Algorithm Development Objectives.

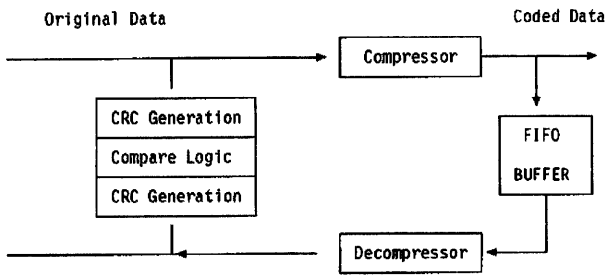


Figure 6. Compression and Decompression Pair for Encoding Reliability.

Extremely High Reliability

The 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. Fig. 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 the checking power by a factor of close to 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 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 the maximum latency behind. This implies that the FIFO buffer between the Compressor and Decompressor

6 CPB: number of machine Cycles needed for compressing or decompressing a (Per) Byte. CPB resembles the use of CPI, Cycles Per Instruction, for RISC and CISC computer architectural effectiveness measurement.

must be at least the latency times the *data expansion*[7] factor in size. And that constraint compels the design of IBMLZ1 to keep the latency small. The smaller compression latency also improves the storage control system response time.

CAM Scrubbing Operation: As pointed out earlier, the CAM's operation is based on using very powerful *parallel combinatorial operators*, and the search of the longest match is done by *prefix-extension*. What if the *prefix-extension* hardware fails? The incoming stream would then be coded as raw byte in every case, since no match longer than 1 byte can be found. The coded output stream will then be expanded by 12.5% due to the 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 can be correctly decoded. This is the class of performance degradation error that the CAM Scrubbing is intended to prevent.

During the 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[6] is defined as the number of machine cycles needed to compress or decompress a byte. CPB is a measure of the 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 MByte/sec in the compression mode, and 40 MByte/sec 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 improving the original data flow were not fruitful until the data flow was mapped to a data dependency graph. The data flow appeared to have a clique (all connected sub-graph) of degree 3 and a self-loop on the exiting vertex of that clique. The clique corresponded to three irreducible computation cycles. The self-loop represented the variable 2 additional cycles. The first data flow then was dropped.

A new data flow was sought, a flow which is of two-colorable (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*. Encouragingly, the new more powerful *parallel combinatorial operators* developed [*CheChe*92], also removed the need for inter-stage control logic all together! Control dependency, in a sense, is a function of distant past data dependency. In the new data flow, each execution stage depends only on the data from the previous stage, the need for inter-stage control is hence removed. Our second data flow achieved rather impressive *symmetrical CPB=1 for compression and decompression* (most LZ1 and LZ2 type compressors today have CPB ranged from 2 to 5) *without the need for inter-stage control.*

286LZ1, IBMLZ1, and ALDC Code Development

The IBMLZ1 compression coding algorithm (format) is a sub-set of the 286LZ1 [*CKCG*95]. The IBMLZ1 is used in IBM's high-performance DASD controller family, tape drive family, and the AIX file system with compression. The ALDC (Adaptive Lossless Data Compression) compression algorithm is a smaller sub-set of the 286LZ1. The ALDC algorithm has been approved as Quater-Inch Cartridge Drive Standard, QIC-154.

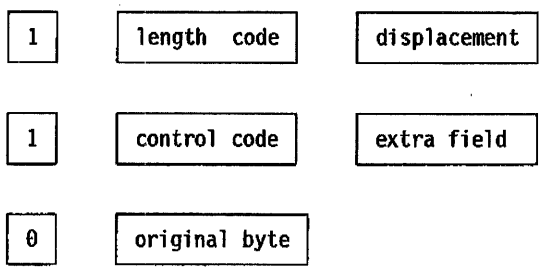


Figure 7. IBMLZ1 Coding Format.

The generic IBMLZ1 compression *code word* format is depicted in Fig. 7. The flag, the first bit, is chosen

[7] Data Expansion: Data expansion occurs when the size of the output stream generated by the compression algorithm is larger than the size of input. The maximum compression achievable is bounded below by the entropy which is data dependant. 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%.

similar to Jackson's [*JacRac*76]. When the flag bit is a '0', the next field is the 8-bit original byte. The existence of this flag bit accounts for the worst case 12.5% expansion[7] for IBMLZ1. When the flag bit is a '1', the next field is a variable-length length or 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 sent away before the end of sending the compressed stream. Or, it can be designated for future decoder redirection. The displacement field is not encoded. Study [*Karnin*91] shown encoding the displacement has limited benefit to 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 IBMLZ1 has 286 *code words* (Fig. 9), grouped in five buckets. The bucket scheme appeared in the SUNSET [*Lang*84*b*] gray-scale compression algorithm, JPEG, and other coding schemes. For the first four buckets, the number of *code words* in each bucket grows in the power of 2 on a lop-sided tree. Those 30 *code words* in the first four buckets approximate the observed exponential or Laplacian distribution.

Fig. 8 shows analytical results of a simpler distribution from one test case. There were 738 count of zero match found in the CAM; 7623 count of match length=1; 3000 count of match length=2; 798 count of match lengh=3; and so on. The exponential distribution (Laplacian or Fractal) of code length is clearly shown.

```
/*********************************************************/
/**  Fast IBMLZ1 with Compression Performance Analysis  */
/*********************************************************/

      Match     Match     Coded
      Length    Count      Bits
         0        738      6642
         1       7623     68607

     __ Matched symbols are:   4514

Match     Match     Coded    Probability
Length    Count      Bits              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 8. Example: LZ1_FAST Coding Efficiency Analysis.

Match Length Statistics Empirical Study

Forty-four test cases of expected data were chosen as the IBMLZ1 algorithm development test suite. They encompassed data bases, programs, object code, system code, and documents in two languages from major applications on VM, MVS, RS6000, and PC. K_code [*Karnin*91], was used as basis for the experimental length/control code study:

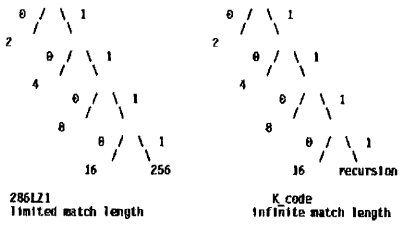


Figure 9. 286LZ1 and K_code for I Length code assignment.

< K_code > :=

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 for Length and Control Field.

< 286LZ1 > :=

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

Figure 11. 286LZ1 Length and Control Field. Five code buckets are used: the prefixes are "0", "10", "110", "1110", and "1111"; the number of *code words* are 2, 4, 8, 16, and 256 in the five buckets respectively.

Key New Observations: The empirical experiment of analyzing the match-length distribution over the 44-case suite and also over large volumes of data revealed two important observation.

1. If maximal match length is limited to 192, the increase in total compressed bytes of the test suite is less than .5% when compared to the maximal match length = 2048 case.
2. For maximal match length = 286, the match length distribution is Laplacian till match length about 27, then appear to be flat.

The 192 match-length limit observation was rather encouraging! 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 to 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 is 271 (Fig. 11).

The second observation was extremely crucial to the algorithmic decision! Since the distribution is Laplacian, it indicated the more complex adaptive arithmetic coding, in this case, could do only slightly better than the Huffman coding. The more complex adaptive coding scheme is hence ruled-out.

The second part of the second observation led us to significant coding simplification and fast hardware operation! The observation suggested that all K_code of length greater than 27 could be lumped into a single bucket. The resulting 286LZ1 has 286 *code words* in five buckets. 270 of them are used for the length description from 2 to 271; 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, the parser is on the speed critical path for decoding, we can compute all five possible length variations and archive multi-byte decompression in a machine cycle!

There were questions raised regarding the theoretic reasoning for the second observation above. More analyses can be made to classify the distribution. A somewhat relevant study is in Cleary-Witten's paper on Partial String Matching
[*CleWit*84]. The experimental results showed the the optimal model order for compressing text, program, numeric data, binary code, grey-scale image, and so on.

IBMLZ1 Compression Results

Fig. 13 shows the compress-to ratios of IBMLZ1 variants. The IBM-1K and IBM-2K are the results of using 1K and 2K history buffer respectively. The IBM-1KF and IBM-2KF are the results of the fast-attack [*CKCG*95] versions that aimed at improving the initial coding efficiency when the history buffer is partially filled. The block size denoting the size limit where the compression is restarted. The blocking

effect appears on channel to storage controller, where 4K is the most typical block size, the next common block size is 2K. CAM of 1K history buffer is about 60K equivalent-cell area, Fig. 13 suggested a 1K CAM instead of 2K will yield good compression and will keep the circuit size down.

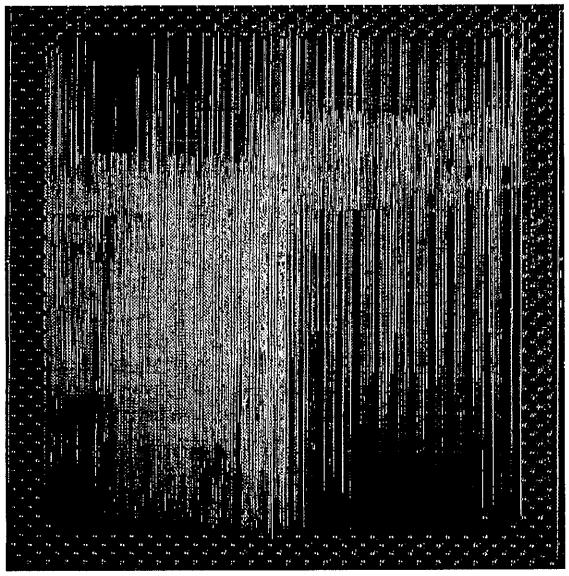


Figure 12. IBMLZ1 COMP Chip Photo for Disk Array Controller.

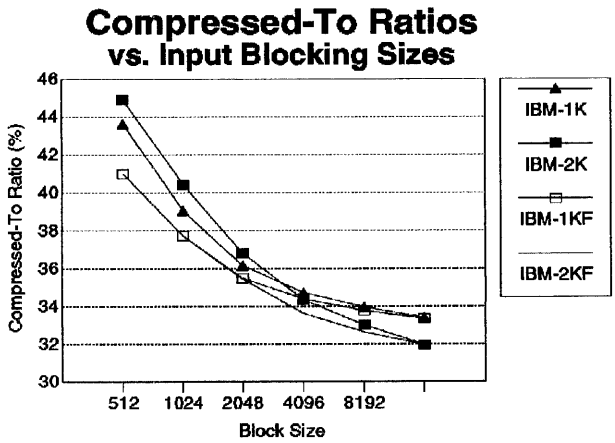


Figure 13. Compressed-To Ratios over 44 Test Cases.

IBMLZ1 Compression Technology for Storage Controller

A family of compression chips based on IBMLZ1 compression algorithm were developed. A 0.8-micron 75,000 gate-array achieved 40 MB/sec throughput [*Latt*93]; a 0.5-micron version is expected to reach 50 MB/sec throughput. Fig. 12 depicts a comprehensive compression sub-system chip with the compression macro imbedded. The chip is operating at 40 MB/sec and is *pipelined* between the ESCON and the Cache unit for the array storage controller. As mentioned earlier, the *pipelined* operation yields the maximal system benefit.

The 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. As 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 to not attempt updates in place, but to collect changed data in a log, and write it 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 re-use.

Conclusion and Future Development

The IBMLZ1 algorithm and technology was 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, scrubbing operation, and extensive build-in checkings. Extremely low CPB=1 [6] compression and decompression have been achieved. The extremely high compressing/decompressing throughput of 30 MB/sec – 50 MB/sec allows transparent mode of operation and hence achieved minimal system overhead. The IBMLZ1 algorithm compresses well over the VM, MVS, RS6000, and PC test cases. Future tasks are: developing format compatible lower CPB and low-overhead data integrity checking architectures.

Acknowledgements

The results reported in this paper could not have been achieved without the outstanding vision and leadership of Ted Lattrell (IBM Micro Electronics); outstanding technical leaderships of David J. Craft and Ehud Karnin; extensive coding performance analysis and verification by Larry Garibay and Mayank Patel; high-performance physical design by Kenneth Gray and Michael Digby; distinguish project leadership of Ted 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. Special thanks to IBM Fellow Arvind Patel on in depth guidances on CRC aspects and analysis; to IBM Fellow James Brady on guidance on control system performance aspects. One of the author, Cheng, would like to express his deepest gratitude to Dr. Glen G. Langdon, Professor at UC Santa Cruz, father of BAC, FileCOMP, ICOM, SUNSET, and many other key compression algorithms and systems, for the long-term guidance on compression algorithms.

References

[***Huffman52***] David A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proceedings of IRE*, Volume 40, PP. 1098-1101, 1952.

[***Elias63***] PP. 61-62 of "Information Theory and Coding," by Abramson, this book contains the reference to arithmetic coding, McGraw-Hill, New York, 1963.

[***Rissa76***] Jorma J. Rissanen, "Generalized Kraft Inequality and Arithmetic Coding "*IBM Journal of Research and Development*, Volume 20, Number 198, 1976.

[***Pasco76***] Richard C. Pasco, "Source Coding Algorithm for Fast Data Compression," Ph.D. thesis, *Department of Electrical Engineering*, Stanford University, 1976.

[***JacRac76***] Rory D. Jackson and Willi K. Rackl, both of IBM Poughkeepsie Lab, "Data Expansion Apparatus," described essentially the LZ1; patent filed on June 30, 1976; US Patent 4,054,951, October 18, 1977.

[***ZL77***] Jacob Ziv and Abramham Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, Volume IT-23, Number 3, PP. 337-343, May 1977.

[***ZL78***] Jacob Ziv and Abramham Lempel, "Compression of Individual Sequences via Variable-Rate Coding," *IEEE Transactions on Information Theory*, Volume IT-24, Number 5, PP. 530-536, September 1978.

[***Gallag78***] Robert G. Gallager, "Variations on a Theme by Huffman," *IEEE Transactions on Information Theory*, Volume IT-24, Number 6, PP. 668-674, November, 1978.

[***Mor78***] R. Morris "Counting Large Numbers of Events in Small Registers," *Communication of ACM*, Volume 21, PP. 840-842, 1978.

[***RiLa79***] Jorma Rissanen and Glen G. Langdon, "Arithmetic Coding," *IBM Journal of Research and Development*, 23, PP. 149-162, March, 1979.

[***Hamming80***] Richard W. Hamming, "Coding and Information Theory," *ISBN*0-13-139139-9, Prentice-Hall, Englewood Cliffs, New Jersy 07632, 1980.

[***RiLa81***] Jorma Rissanen and Glen G. Langdon, "Universal Modeling and Coding," *IEEE Transactions on Information Theory*, Volume IT-27, Number 1, PP. 12-23, January 1981.

[***LaRi81***] Glen G. Langdon, and Jorma Rissanen "Compression of Black-White Images with Arithmetic Coding," *IEEE Transaction on Communication*, COM 29,6, PP. 858-867, June, 1981.

[***HLMT82***] Daniel R. Helman, Glen. G. Langdon, N. Martin, Stephen Todd, "Statistics Collection for Compression Coding with Randomizing Feature," *IBM Technical Disclosure Bulletin*, 24, 4917, 1982.

[***Lang84a***] Glen G. Langdon, "An Introduction to Arithmetic Coding," *IBM Journal of Research and Development*, Volume 28, Number 2, 135-149, March 1984.

[***MiWa84***] Victor S. Miller and Mark N. Wegman, "Variation on a Theme by Ziv and Lempel," *IBM RC 10630*, July 31, 1984.

[***Lang84b***] Glen G. Langdon, "Further Development in Lossless Gray-scale Image Compression," *IBM Conference on Pattern Recognition and Image Processing*, Session C1, IBM T. J. Watson Research Center, November 14, 1984. Also is known as the "SUNSET algorithm," reprinted as *IBM Research Report*, RJ6426, September, 1988.

[***Fla85***] Philippe Flajolet, "Approximate Counting: A detailed Analysis," BIT 25, PP. 113-134, 1985.

[***WiNeCl87***] Ian H. Witten, Radford M. Neal, and John G. Cleary, "Arithmetic Coding for Data Compression," *Communication of ACM*, Volume 30, Number 6, PP. 520-540, June, 1987.

[***PMLR88***] William B. Pennebaker, Joan L. Mitchell, Glen G. Langdon, Jr., and Ronald B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM Journal of Research and Development*, Volume 32, Number 6, PP. 717-726, November, 1988.

[***PeMi88***] William B. Pennebaker and Joan L. Mitchell, "Probability Estimation for the Q-Coder," *IBM Journal of Research and Development*, Volume 32, Number 6, PP. 737-752, November, 1988.

[***KaGiPa89***] Randy H. Katz, Garth A. Gibson, and David A. Patterson, "Disk System Architectures for High Performance Computing," *Proceedings of the IEEE*, Volume 77, Number 12, December 1989.

[***CoLeRi90S***] Thomas H. Cormen, Charles, E. Leiserson, and Ronald L. Rivest "String Matching," chapter 34 of *Introduction to Algorithms*, McGraw-Hill Book

[***CoLeRi90R***] Thomas H. Cormen, Charles, E. Leiserson, and Ronald L. Rivest "Radix Sort," of chapter 9 on Sorting in Linear Time, *Introduction to Algorithms*, McGraw-Hill Book

[***Karnin91***] Ehud D. Karnin, "Evaluation and Enhancement of LZ-1 Based Data Compression Systems," *IBM Science and Technology, Haifa Research Group*, draft, 1991.

[***LCAM92***] 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; *IBM Research Report*, RJ 8611, February 6, 1992.

[***CheLan92***] Joe-Ming Cheng and Glen G. Langdon, "Image Compression with QM-AYA Adaptive Binary Arithmetic Coder," *SPIE Volume 1771, Application of Digital Image Processing XV*, PP.413-423, 1992.

[***CheChe92***] 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 ," SA9-94-016, IBM filed US patent application.

[***PaMcLLa93***] Mayank Patel, Neil MacLean, and Glen G. Langdon Jr., "An Economical Hardware-Oriented High-Speed Data Compression Scheme ," *IBM Research Report*, RJ 9170, January 11, 1993.

[***CheLan93***] Joe-Ming Cheng and Glen G. Langdon, "Modified Metropolis Annealing Algorithm for QM-AYA Arithmetic Coder Design Optimization," *SPIE Volume 2028, Application of Digital Image Processing,* PP.2-12, 1993.

[***Latt93***] Ted Lattrell, "40 MB/sec Throughput IBM Compression Chip and Macro Announcement," *EE Times,* front page, August 16, 1993.

[***CKCG95***] 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 US patent application.