

# VERY HIGH SPEED ENTROPY CODING

*Martin Boliek, James D. Allen, Edward L. Schwartz, Michael J. Gormish*

RICOH California Research Center  
2882 Sand Hill Road, Suite 115 Menlo Park, CA 94025  
Internet: boliek@crc.ricoh.com

## ABSTRACT

Efforts to build high-speed hardware for many different entropy coders are limited by fundamental feedback loops. Here is a method that allows for parallel compression in hardware. This parallelism results in extremely high rates, 100 million symbols/second or higher. The system is generalizable to any lossless or lossy system with deterministic decompression.

Prototype hardware that divides the data into multiple streams that feed parallel coders is presented here. The problem of efficient transmission of multiple streams of variable-length coded data is solved by a unique coded data interleave method.

## 1. INTRODUCTION

There are many applications for very high-speed entropy coders such as the lossless portion of High Definition Television, high-speed computer networks, expensive satellite networks, etc. Most efficient entropy coders are limited in speed by fundamental feedback loops. A possible solution is to divide the incoming data stream into multiple streams and feed these to parallel encoders. The output of the encoders are multiple streams of variable-length coded data. The problem is how to transmit the data on a single channel. An elegant method of interleaving these coded data streams is presented.

Before describing the interleaving method, some possible ways to parallelize the data stream are presented. From this discussion a system block diagram is given. To understand the insight for the interleaving method, a concatenated file system for transmitting the coded data is presented. This system, while achieving the speedup desired, requires large memory buffers. However, examination of this system shows the deterministic nature of the decoder and leads to a natural codeword order. This order is exploited to interleave the data at the encoder without requiring buffers at the decoder, markers in the coded data stream, or other overhead.

Two system examples are given: a run length coding system for which prototype hardware has been built and a non-adaptive Huffman coding system, such as used by JPEG and MPEG. A discussion of new problems that arise with parallel coders is offered.

## 2. PARALLEL DATA STREAMS

There are many different variable-rate high-compression entropy coding systems. Coders, such as Huffman Coding and the Q-Coder, can achieve moderate speeds in hardware implementations. However, because of various feedback loops and the sequential processing of data, these systems are unable to achieve extremely high rates needed for many applications.

### 2.1 Q-Coder, Finite State Machine, and Huffman

The Q-Coder is a binary arithmetic coder that uses shifts and additions instead of multiplications [1]. Although shifts and additions are much faster than multiplications, every bit requires one addition and possibly one shift operation for either encoding and decoding. The result of these operations is required before processing the next bit or codeword, thereby fundamentally limiting the throughput of this coder. Many attempts have been made to increase the speed of this loop [2][3][4], but no parallelization method for this inner loop has been offered.

Finite state machine coding is a new entropy coding method [5][6][7] in which the code stream is created from transition logic or a look-up table. The bit stream and the probability "class" along with the current state of the machine form the index, or address, to this table. However, this method still has a fundamental feedback loop because the new state must be known before the next bit can be processed.

Huffman coding is a fixed-length input to variable-length output system. While there are methods for performing encoding in parallel, the decoding is more difficult. A high-speed Huffman decoder may offer many bits of the code stream to a look-up table and the token is output along with the codeword length. This length information is essential for shifting the code stream to align the next codeword. A "superscaler" scheme that uses the token statistics to predict the location(s) of consecutive codewords has been offered [8]. However, like microprocessor superscaler designs, the hit rate is dependent on the data and the speedup is limited, sub-linear in hardware cost.

### 2.2 Parallel Data Streams

Since it does not seem possible to parallelize the inner loop

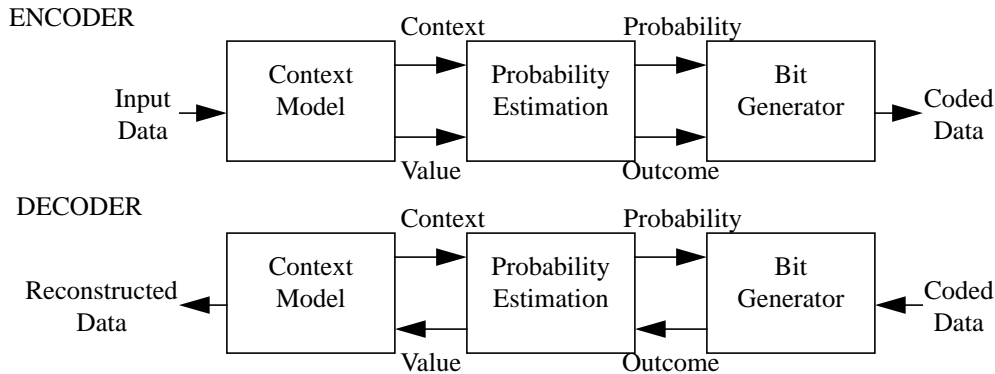


Figure 1 - Entropy Coding Model

of these systems, the obvious alternative is to replicate the entire coder and parallelize the input stream. For example, image compression systems that divide the image into blocks (tiling) can easily be imagined. Each of these blocks is fed to its own processor. The speedup of a system like this is directly proportional to the number of coders employed. However, it requires that the entire image, or data streams from different parts of the image, be available to the compressor at the same time. This might require a frame buffer for data reordering if the data is delivered in raster (or other arbitrary) order. If the system is truly real-time, a second buffer might be needed to handle the next frame while the current one is being processed (banking). Of course the same buffering is needed on the decompression side.

In some cases, it may be desirable to divide the data by context or probability class. This specialization can permit lower cost hardware in some applications. Gooch [9] patented an interesting method that separated the data into three different streams according to probability skew. Gooch uses an 8-bit template as a context model. The context bins are grouped into one of three probabilities, highly skewed, moderately skewed, and no skew (50%). A long run length code is used for the first group, a shorter run length code is used for the second class, and no coding is used for the third class. Note, however, that this system was not designed to increase the speed, although, if the prediction was fast enough, it could have that result. Also, this method suffers from the same frame buffer banking need as the previous example.

Consider why a system might need buffering. The data order into the encoder is the same as out of the decoder. However, the codeword order might be quite different because of the fractional bit characteristic of many entropy coders. An encoder may encode several bits before emitting a codeword. The bits directed to any given encoder could be separated by many bits in the original stream. These intervening bits are directed to other

encoders and may create several, perhaps hundreds or thousands, of codewords. The decoder, however, needs the codeword for the first bit first. Thus, reordering the codewords is essential for this scheme. An elegant method of interleaving these codewords is presented.

### 3. PARALLEL ENTROPY CODING

Consider the classical entropy coding model shown in Figure 1 [2]. To encode, the bits are segregated into contexts based on past data. The probability state, or class, is determined for every context independently. Given the outcome and the probability state, the bit is encoded using the most appropriate code available. The decoder has the opposite data flow but the context and probability class are determined in exactly the same order. Based on previous bits, the context is determined and passed to the probability model. The probability of that context is retrieved (and/or updated) and the proper code is determined for decoding. The input stream must then be shifted according to the length of the codeword decoded. Finally the decoded bit is ready for the determination of the next context. Note that this model is quite general. It could describe arithmetic coding, Q-coding, adaptive Huffman coding, adaptive Golomb run length coding, and many more.

#### 3.1 Adding Parallelism to the Entropy Coding Model

To parallelize the data stream, the data may be divided according to either context or probability or any other method. A parallel encoder fed by data differentiated by context model (CM) is shown in Figure 2. (Figure 2 shows three parallel coders but any number could be used.) The CM divides data stream into different contexts in the same way as a conventional CM and sends the multiple streams to the parallel hardware encoding resources. Individual contexts, or groups of contexts, are directed to different probability estimators (PEM) and bit generators (BG).

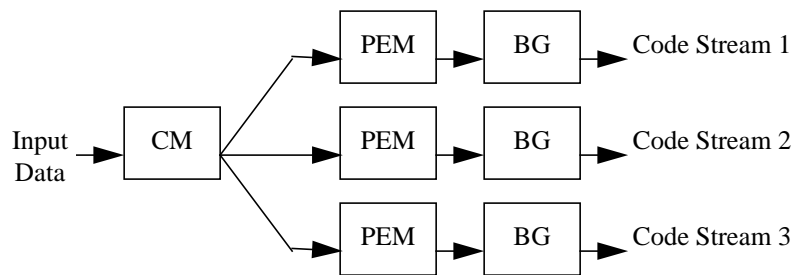


Figure 2 - Context Dependent Parallel Encoder

For decoding the input comes from several code streams, one for each PEM and BG. The CM takes the decompressed data bit from the appropriate PEM and BG, in effect reordering the data into the original order. Note that the control for this design flows in the reverse direction of the data stream. The BG and PEM decodes data as the CM needs it, staying at least one bit ahead.

This configuration is designed to couple the PEM and BG tightly. The IBM Q-Coder is a good example of this type of coder. Local feedback loops between these two are not the fundamental limit to system performance. In a different design, the PEM could differentiate the data and send it to parallel BG units. Thus, there would be only one CM and PEM and the BG is replicated. Adaptive Huffman coding and finite state machine coding could be used in this way.

### 3.2 Channel Ordering of Multiple Data Streams

There are many different design issues and problems that affect system performance. A few of these will be mentioned below. However, the major problem with the design shown in Figure 2 is the multiple code streams. Systems with parallel channels that could accommodate this design are imaginable: multiple telephone lines, multiple heads on a disk drive, etc. However, for this discussion, it is assumed that only one channel is available, or convenient. Indeed, if multiple channels were required there would probably be poor utilization of the bandwidth because of the bursty nature of the individual code streams.

In a method for using one channel similar to the Gooch method [9] the code streams are concatenated and sent contiguously to the decoder. Pointers or instream marker codes locate the beginning bit location of each stream. The complete compressed data file is available in a buffer to the decoder. As needed, the codewords are retrieved from the proper location.

Note that this method requires an entire coded frame to be stored at the decoder and, for practical purposes, at the encoder. If a real-time system, or less bursty data flow, is required then there must be two frame buffers for banking

at both the encoder and the decoder. These requirements significantly increase the cost of a hardware implementation.

### 3.3 Data Order to Codeword Order

Notice that a decoder decodes codewords in a given deterministic order. With parallel coding, the order of the requests to the code streams, while possibly complicated, is deterministic. Thus, if the codewords from parallel code streams can be interleaved in the right order at the encoder, then a single code stream will suffice. The codewords are delivered to the decoder in the same order on a just-in-time basis. At the encoder, a model of the decoder determines the codeword order and packs the codewords into a single stream. This model might be an actual decoder.

With variable-length codewords interleaving into a single stream at the encoder requires a variable bit shifter (or barrel shifter) at the decoder to align the codewords. This is the bottleneck of the system and mitigates the virtue of parallelism. One solution is to use coders with fixed-length codewords, such as Tunstall coding [11], to remove the feedback loop. This has a possible cost in coding efficiency or complexity.

The solution offered here is to group the codewords from each independent code stream into fixed-length words, called interleaved words. It is convenient to have the interleaved word length larger than the maximum codeword length so at least one codeword is contained in each interleaved word. The interleaved words can contain many codewords and parts of codewords. Figure 3 shows the interleaving of an example set of parallel code streams.

These words are interleaved according to the demand at the decoder. Each independent decoder receives an entire interleaved word. The variable-length shifting operation is now done locally at each decoder, maintaining the parallelism of the system. Note in Figure 3, that the first codeword in each interleaved word is the lowest remaining codeword.

This concept complicates the encoder, especially for real-time encoding. However, using the actual decoder to model the codeword request order accounts for all design

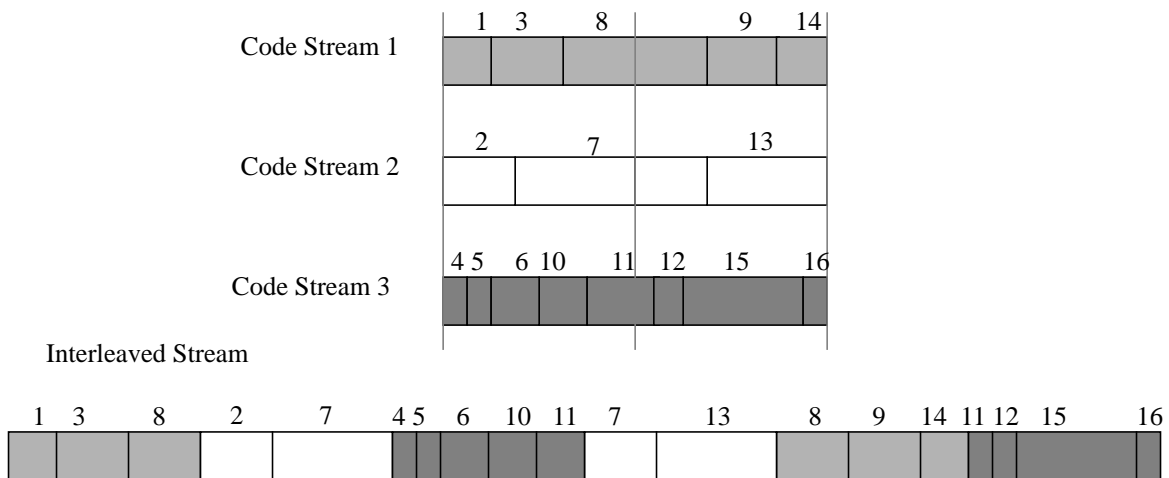


Figure 3 - Interleaved Code Streams

choices and delays. This is not a great cost for half duplex systems that must have both encoders and decoders anyway.

#### 4. R-CODERS EXAMPLE

To illustrate the power of this interleaving concept, an example is offered. Special run length codes have been developed specifically for this type of high-speed parallelism. The run length codes are based on the generalization of Golomb codes suggested by Gallager and Van Voorhis [12]. Codes are selected in the manner similar to that suggested by Langdon [13]. However, instead of each codeword leading to a new code it leads to a new state in a state machine. This single stream coder is called an R-coder.

This particular coder was chosen for its small hardware cost, coding efficiency, and speed. Since each coder is replicated many times it is important that they are as small as possible to reduce the cost of a hardware implementation. The coding efficiency of the R-Coder is comparable to the QM-Coder. Finally, the R-Coder is extremely fast in hardware. The main operation is counting and can be implemented near the speed limit of any VLSI technology.

RICOH has implemented a prototype of an interleaved decoder system with six virtual R-Coders. FPGAs (Altera FLEX8000) were used for fast prototyping. This design was intended to deliver a constant, high bit rate (16 Mbit/s) at the output of the decompressor. The encoding is performed in non-real-time on a computer.

Despite the limitations of FPGA implementation, the speed, gate count, and memory requirements compare well with the Q-Coder performance reported in [2] (5-10 Mbit/s depending on compression, 13,000 gates, 18,432 bits of memory). To give some perspective to the 16 Mbit/s rate, it

is equivalent to facsimile transmission of letter size, 400 dpi images at 64 pages per minute.

It is estimated that this circuit could run 10 to 15 times faster in a VLSI implementation.

The compression rate for the R-Coder and the QM-Coder on the 8 standard ITU-TSS test images using the same JBIG template varies by less than 2 percent.

#### 5. HUFFMAN CODE EXAMPLE

Another system example is a modification of the JPEG Huffman coding [15]. (Note that this system is also appropriate for MPEG Huffman coding.) In this case the selection of hardware resources is performed with a sequencer state machine. The symbols are delivered to adjacent coders in order. For example, if there are five coders then each coder receives every fifth symbol. Since the codewords are variable-length, concatenating into fixed-length words and reordering for channel interleaving is performed as before. The codewords are delivered to the decoders on demand and the symbols are decoded in the same sequential order in which they were encoded.

The speedup of this system is directly proportional to the number of coders. The above system would be nearly five times faster than a single coder. Thus, for a coder that decodes at 20 million symbols per second, the five coder system would run at 100 million symbols per second. Note that this is around the worst case token rate for High Definition Television.

This system also has the advantage that the buffering needed by the encoder is low and deterministic (not data dependent). Because Huffman coding creates a codeword for every token and the tokens are delivered equally between the coders, the worst case buffer length can be determined, and is typically around 100 bytes per parallel coder.

## 6. NEW PROBLEMS OF PARALLEL CODING

One consideration of this parallel system is the distribution of data between the coders. If a particular window of the data stream leads to many bits coming from the same coder, the system throughput is reduced to the throughput of that single coder during that time. In order to utilize the parallel coders efficiently the selection of hardware resources should divide the data as equally as possible, locally as well as globally.

Related, in part, to the above problem is the design of the context model. Now that the code stream can be decoded at extremely high rates, the context model could be the new bottleneck. Also, the context models might need rules to prevent consecutive access to the same context bin to help alleviate the distribution problem.

As mentioned above, the interleaving of several code streams places the complexity of the system on the encoder. The individual code streams must be buffered while the codewords are interleaved into the data stream. This might require a good deal of storage, especially if the data for one context bin is far more skewed than for others, or is rarely called. In these two cases, the first bit to be coded in the codeword is many bits away from the last and several, may be hundreds or thousands of other codewords are created by the intervening data. These codewords must be stored.

## 7. CONCLUSIONS AND FUTURE EXTENSIONS

A new method of interleaving coded streams that allows for parallel entropy coding has been introduced. An example prototype proves that this approach is viable and results in extremely fast, cost efficient, and coding efficient VLSI implementations. An example of how this technology could be used to accelerate the Huffman coding shows a near linear relation between speedup and additional hardware cost. Finally a few of problems that are new to coding technology that arise from parallel coding have been discussed.

This technology could be extended to systems that are lossy. The only requirement is for a deterministic decoder that can be modeled at the encoder. Also, the parallel coders need not be the same type of coder. Thought can also be given to mixing any type of data that needs to be sequenced. One example is the interleaving of video and sound data in low bit rate video compression. This data needs to be synchronized, however the video data is quite bursty (interframes as opposed to predicted frames) while the sound data might be more constant. This interleaving method would ensure synchronization without marker bits in the data stream.

This technology enables not only a whole new approach to high-speed entropy coding but also new ways

of handling parallel streams.

## BIBLIOGRAPHY

- [1] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon, Jr., R.B. Arps, "An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder," *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 717-26, Nov 88.
- [2] R.B. Arps, T.K. Truong, D.J. Lu, R.C. Pasco, T.D. Friedman, "A Multi-purpose VLSI Chip for Adaptive Data Compression of Bilevel Images," *IBM J. Res. Develop.*, Vol. 32, No. 6, pp. 775-95, Nov 88.
- [3] G. Feygin, P.G. Gulak, P. Chow, "Minimizing Error and VLSI Complexity in the Multiplication Free Approximation of Arithmetic Coding," *Proc. Data Compression Conference*, 30 Mar 93, Snowbird, UT, pp. 118-27.
- [4] H. Printz, P. Stubbley, "Multialphabet Arithmetic Coding at 16 MBytes/sec," *Proc. Data Compression Conference*, 30 Mar 93, Snowbird, UT, pp. 128-37.
- [5] J.D. Allen, "Method and Apparatus for Entropy Coding," US Patent 5,272,478, 21 Dec 93.
- [6] M.J. Gormish, "Source Coding with Channel, Distortion, and Complexity Constraints", Ph.D. thesis, Stanford University, March 1994.
- [7] P.G. Howard, J.S. Vitter, "Practical Implementations of Arithmetic Coding," *Image and Text Compression* (J.A. Storer, editor), Kluwer Academic Publishers, Boston, MA, 92, pp. 85-112.
- [8] M.P. Boliek, J.D. Allen, E.L. Schwartz, "High Speed Superscaler Huffman Decoder Hardware Design," *IS&T/SPIE Symposium on Electronic Imaging*, Vol. 2186-06, 9 Feb 94, and US Patent 5,325,092, 28 Jun 94.
- [9] R.P. Gooch, "Method and Apparatus for Adaptive Facsimile Compression Using a Two Dimensional Maximum Likelihood Predictor," US Patent 4,325,085, 13 Apr 82.
- [10] T.C. Bell, J.C. Cleary, I.H. Witten, *Text Compression*, Prentice Hall, Englewood Cliffs, 90.
- [11] B.P. Tunstall, "Synthesis of Noiseless Compression Codes," Ph.D. Thesis, Georgia Institute of Technology, 68.
- [12] R.G. Gallager, D.C. Van Voorhis, "Optimal Source Codes for Geometrically Distributed Integer Alphabets," *IEEE Transactions on Information Theory*, Vol. IT-21, No. 2, Mar 75, pp. 228-30.
- [13] G.G. Langdon, Jr., "An Adaptive Run-length Coding Algorithm," *IBM Technical Disclosure Bulletin*, Vol. 26, No. 7B, Dec 83.
- [14] JBIG, "Coded Representation of Picture and Audio Information - Progressive Bi-level Image Compression Standard," ISO/IEC JTC1/SC29/WG9.
- [15] W.B. Pennebaker, J.L. Mitchell, *JPEG Still Image Compression Standard*, Van Nostrand Reinhold, New York, 93.
- [16] J.D. Allen, M.P. Boliek, E.L. Schwartz, "Method and Apparatus for Parallel Decoding and Encoding of Data," US Patent Pending (allowed), filed 10 Feb 93.