A Problem in Real-Time Data Compression:  
How to Keep the Data Flowing at a Regular Rate  

by  

Sunil Ashtaputre  

Jo Perry  

and  

Carla Savage  

Center for Communications and Signal Processing  
Department of Computer Science  
North Carolina State University  

December 1985  

CCSP-TR-85/19
Abstract

We consider the following problem: an input stream of digital data enters a hardware device H at a regular rate. The function of H is to compress the input data in real time according to some appropriate scheme. By "compress", we mean that the output signal leaving H will, in some sense, contain fewer data items. Although the input signal data may enter H at a regular rate, the data compression scheme of H may be such that the output signal data leaves H at an irregular, unpredictable rate. This could prove to be awkward or inconvenient when the output signal data goes on to be processed in some succeeding stage. We show in this paper that under certain restrictions, it is possible to construct a hardware device of relatively small area to make the output of H flow at a regular rate.
1. Introduction

There are many reasons for wanting to encode or compress a digital signal. We are concerned with two reasons related to the design of special purpose hardware for signal processing. First, compression could allow more processing time per data item. Since we require our hardware to process in real time, data must be processed at the rate at which it is transmitted. If the rate is too fast, the hardware requirements may be out of the range of current technology. More likely, the hardware requirements could be met, but maybe only by complicated parallel/pipelined architectures implementing sophisticated computational procedures. Data compression could allow more processing time per data item, resulting in simpler hardware designs or rendering feasible certain computations which would have been infeasible at the transmission rate of the uncompressed signal. It should be mentioned that even though we may have more processing time per data item after compressing a signal, the computations to be performed on the compressed data may have become more complicated than the computations which were to have been performed on the original signal, so there is a tradeoff involved.

The second reason for our interest in data compression is that the hardware requirements, in area, to process a compressed signal may be less than those required for processing the original signal. For example, we have been able to show that certain computations on a sequence of raster scan images, each consisting of N pixels, require hardware of area at least proportional to N [4]. However, we can show that these same computations can be performed on a quadtree encoded version of the original image [1] with hardware of area proportional to the number of nodes in the quadtree,
which is usually smaller than N.

In this paper, we describe a solution to a problem which arises in trying to process a compressed signal with hardware. The problem is that an irregular scheme for compressing data, such as converting an image to its quadtree representation, may transform an input signal transmitted at a regular rate (that is, rhythmically) into an output signal transmitted at an irregular rate (arhythmically). This may make it difficult for subsequent processing of the output signal and may make it difficult to take advantage of the "more processing time per data item" gain which was one of our reasons for compressing the input signal in the first place. Our concern here is to find a way to make the data rhythmic again by using hardware of area small enough so as not to override the "less processing area" gain which was our second reason for compressing the input signal.

We describe our environment and assumptions informally in Section 2 and present an example. In Section 3, we define what we mean by rhythmic and arhythmic data and prove our main result, that under certain conditions, arhythmic data can be made rhythmic in hardware with small area.

3. The Problem

We consider the following problem: we have an input stream of digital data entering a hardware device H at a regular rate. H has been designed to compress the input data according to some appropriate scheme [Fig. 1]. For example, it may sample and output every other input value (in which case the output signal would approximate
the input signal). Or, it may encode the signal in a more compact format, preserving all of the information in the original signal. Whatever the hardware device, \( H \), does, we assume that the output signal leaving \( H \) will, in some sense, contain "less data" than the input signal. Although the input signal data may enter the hardware at a regular rate, the data compression scheme may be such that the output signal data leaves the hardware device at an irregular, unpredictable rate [Fig. 2(a)]. This could prove to be awkward or inconvenient when the output signal data goes on to be processed in some succeeding stage. We would like to construct a hardware device \( H' \) which would accept as input the output of \( H \) and produce as output the same output signal as \( H \), except at a regular rate [Fig. 2(b)]. The problem is described more formally in the next section, and a solution is presented.

3. A Solution

Assume that the data items enter a hardware device \( H \) in sequence, at the rate of one per time unit. In each time unit, \( H \) has the option of outputting a data item or not. Define the output signal rate to be rhythmic if there exists a linear function \( t \) from the positive integers to the positive integers such that the \( i \)-th output data item is produced at time \( t(i) \). Otherwise, the output signal rate is called arhythmic. For example, if the output is produced according to the function

\[
    t(i) = ai + b
\]

this would mean that after a delay of \( a + b \) time units, an output value is produced every a time units.
We would like to prove a result similar to one for Turing Machines in the paper [3]. The theorem would be stated as: If for some linear function \( t \) a hardware device produces the \( i \)-th output on or before time \( t(i) \), then the device can be modified to produce the \( i \)-th output at exactly time \( t(i) \). That is, if the time at which a (finite state) hardware device \( H \) produces its output values is bounded above by a linear function, then it is possible to construct a (finite state) device \( H' \) to make this output signal rhythmic. However, this is not necessarily true. The problem is that such a device as \( H' \) might require unbounded storage. For example, if we were guaranteed only that the \( i \)-th output of \( H \) leave \( H \) on or before time \( t(i) \), it may be that, in fact, \( t(k) \) values are output from \( H \) and input to \( H' \) in the \( t(k) \) consecutive time units and only \( k \) of these values can be output by \( H' \) at times \( t(1), t(2), ..., t(k) \). Then, \( H' \) must be able to store \( t(k) - k \) values until they can be output. The quantity \( t(k) - k \) grows as \( k \) grows if the coefficient of the linear term in \( t \) is greater than one. But then \( H' \) would need to store an arbitrarily large amount of data. Thus, in the case where we know only that output values are produced in time bounded by a linear function, there is no hope that in every case a finite state piece of hardware can be built to make an arhythmic output signal rhythmic.

However, most signals and compression schemes of interest satisfy more stringent conditions which will allow us to "regularize" the output signal rate. We imagine the input signal to be partitioned into blocks of size \( N \). (This partitioning could be very natural, as in the case where the input is a sequence of images, each of length \( N \). Or it could be artificially imposed.) Assume that the data compression scheme is such that an input block of size \( N \) is compressed into an output block of size \( N/p \) for some \( p > 1 \) and
that the output produced over time satisfies the following:

There exists a constant $D$ such that all $N/p$ output values in the $i$-th output block are produced between times $(i-1)N + 1 + D$ and $iN + D$, inclusive.

That is, in each time unit, although the data compression hardware device, $H$, has the option of outputting or not outputting a value, all $N/p$ values compressed from block $i$ must be output within a single $N$-time unit block. Under these conditions we can prove that the output signal rate can be made rhythmic, in the sense of the definition. Further, the area of the hardware required to make the output signal rhythmic can be kept small.

**Theorem.** Let $H$ be a hardware device which outputs blocks of size $N/p$ over consecutive time blocks of size $N$. Let $D$ be a constant such that all $N/p$ outputs in output block $i$ are produced at times $t$ satisfying

$$(i - 1)N + D + 1 \leq t \leq iN + D$$

Assume further that $p$ is an integer greater than one and $p$ divides $N$. Then there is a hardware device, $H'$, which takes as input the output sequence of $H$ and outputs this sequence at a regular rate. Further, the area of $H'$ is bounded above by

$$\frac{N}{p} - \text{floor} \left( \frac{N}{p^2} \right)$$

for some constant $c$.

**Proof.** For simplicity, assume that in a time unit if a hardware device does not output a data value, it produces a special symbol, say "b". Thus, a "b" in the input signal to $H'$ indicates a time unit during which no output value was produced by $H$. (In particular, the first $D$ values output from $H$ will be "b".) Construct $H'$ as follows. $H'$ will consist of a
queue, a mod p counter, a counter to count up to \( D + N - N/p \) and some logic. (See [2] for a hardware implementation of a queue.)

As the input signal (the output signal of \( H \)) enters \( H' \), each "b" value is ignored and each non-"b" value is added to the queue. \( H' \) produces no output values for the first \( D + N - N/p \) time units. After that, every \( p \) time units the front element is removed from the queue in \( H' \) and is produced as output. In order to show that \( H' \) works correctly, we must first show that the queue is never empty when it is time to delete an element from the front. The fact that we never delete from the queue during the first \( D + N - N/p \) time units will guarantee this. From time 1 to time \( D + N - N/p \) there is no attempt to delete the queue. We can prove by induction on \( i \) that for \( i \geq 0 \),

a. By time \( D + iN + (N - N/p) + j \) (for \( 1 \leq j \leq N/p \)), at least \( j \) values from block \( i + 1 \) have been read into the queue and only \( \text{floor}(j/p) \) values from block \( i + 1 \) have been deleted, so the queue is not empty.

b. By time \( D + (i + 1)N \), all values from block \( i + 1 \) have been read into the queue.

c. The last value from block \( i + 1 \) does not leave the queue until time \( D + (i + 1)N + (N - N/p) \).

Thus, summarizing the results of a, b, and c, the queue is never empty, for all \( i \geq 0 \), over time interval
\[ D + iN + (N - N/p) + 1 \text{ through } D + (i + 1)N + (N - N/p) \]

In addition to showing that the queue is never empty when we need a value, we must also show that there is an upper bound on the maximum number of values ever stored in the queue. During time units 1 through \( D \), no values enter the queue. During time units \( D + 1 \) through \( D + N - N/p \), a maximum of \( N/p \) values from block 1 can enter the queue, since

\[ D + N - N/p < D + N. \]

We can prove by induction on \( i \) that for \( i \geq 0 \),

a. At times \( D + iN + (N - N/p) + j \) (for \( 1 \leq j \leq N/p \)), the queue contains no values from block \( i \). At most \( N/p \) values from block \( i + 1 \) have entered the queue and \( \text{floor}(j/p) \) of these have been deleted.

b. At times \( D + (i + 1)N + j \) (for \( 1 \leq j \leq N/p \)) all \( N/p \) values from block \( i + 1 \) have entered the queue and \( \text{floor}((N/p + j)/p) \) of these have been deleted. At most \( j \) values from block \( i + 2 \) have entered the queue and none have been deleted yet.

c. At times \( D + (i + 1)N + N/p + j \) (for \( 1 \leq j \leq N - 2N/p \)), the number of values from block \( i + 1 \) remaining in the queue is \( N/p - \text{floor}((2N/p + j)/p) \). At most \( N/p \) values from block \( i + 2 \) have entered and none of these have been deleted yet.

During these time intervals a, b, and c, the queue could attain its maximum size as follows:
a. at \( j = 1 \), size \( N/p \)

b. at \( j = N/p \), size \( 2N/p - \text{floor}[(2N/p)/p] \)

c. at \( j = 1 \), size \( 2N/p - \text{floor}[(2N/p + 1)/p] \)

Each of these three quantities is bounded above by

\[
\frac{2N}{p} - \text{floor} \left( \frac{2N}{p^2} \right)
\]

which is therefore an upper bound on the size of the queue.

4. Conclusions and Extensions

The theorem of Section 3 can probably be extended to handle the cases where \( p \) is not an integer or \( p \) does not divide \( N \). It would be more interesting (and realistic) to consider the case where each block of size \( N \) is compressed into at most \( N/p \) data items rather than exactly \( N/p \) data items. It may also be of interest to consider how to handle varying block sizes.

It is fortunate for us that the area required to make the output rate regular is proportional to the size of the compressed blocks of data rather than the size of the input blocks. We have shown that problems which require hardware of area \( O(N) \) for real-time processing of images of size \( N \) could be solved by hardware of \( O(m) \) if the image could be represented by a quadtree of \( m \) nodes [4]. We can encode an image into its quadtree representation in real time with hardware of area \( O(m \log N + N) \) [1], but the output data rate is very irregular. If we required, for example, \( O(N) \) area to regularize the output data rate, our purpose in using the quadtree representation would be defeated.
The hardware device $H'$ described in Section 3 to regularize the output signal is relatively flexible. It could be made programmable to accommodate varying values of $N$, $p$, and $D$, although it is limited by the maximum queue size.

5. References


Figure 1

input signal \rightarrow H \rightarrow output signal

Data Compression Hardware

(compression)
Figure 2. "x" represents input signal value
"y" represents output signal value
"b" represents time unit during which
no output value is produced