HIGH THROUGHPUT RECONSTRUCTION OF HUFFMAN-CODED IMAGES

Horng-Der Lin and David G. Messerschmitt

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, California 94720
(415) 642-0294

ABSTRACT

The Huffman code is an efficient noiseless coding method for images, video, and other applications. For applications in graphic computers, high-definition televisions (HDTV), and optical fiber networks, Huffman-coded images need to be reconstructed at a high throughput rate. Due to the recursion within the reconstruction algorithm, the achievable throughput rate in a given IC technology is limited by the iteration bound. We propose methods of relaxing the iteration bound of the reconstruction process, thus increasing the concurrency in the decoding process. These methods are simple, effective, and flexible. Unlimited concurrency can be achieved at the expense of additional latency. The incurred overhead is very low, and the complexity increases only linearly with the achieved throughput improvement.

1. INTRODUCTION

With the advance of graphic computers, high-definition television (HDTV), and optical fiber networks, efficient methods of coding images and high-throughput real-time reconstruction of coded images become increasingly important. The Huffman code [1] is a popular source coding method that approximates entropy coding. It is a noiseless coding technique, meaning that no distortion is added to the image. The code is thus particularly attractive for high-quality images like HDTV and high-quality media like disk storage or optical fibers.

In image processing, the Huffman code assigns the lengths of its variable-length codewords according to the probabilities of the quantized sample values within the image, or approximately the frequency of the occurrences of these values. The Huffman code achieves close to the theoretical minimal average codeword length. It is often used in conjunction with other algorithms to compress image information.

In the Huffman coding process, the quantized sample values, or more generally the messages with the lowest probability are combined recursively in a tree structure. For example, suppose there are four possible quantized values with their probabilities as shown in table 1. We combine the two values C3 and C4, which are the two with the lowest probability, to form a codeword of length 9. Within a tree structure and sum up their probability at the junction, as shown in figure 1. Then the next set of lowest probability messages or junctions is then combined recursively until the tree converges to a single junction, as illustrated in figure 1. The branches of the tree are denoted arbitrarily as ones and zeros. To form a codeword for a message, the tree is traversed back to a message node while recording the sequence of path designations, one or zero for each branch of the tree. For example, assigning an upper branch to 1 and a lower branch to 0 in figure 1 leads to the codewords shown in table 1.

<table>
<thead>
<tr>
<th>Table 1. An example Huffman code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantiization Level</td>
</tr>
<tr>
<td>C1</td>
</tr>
<tr>
<td>C2</td>
</tr>
<tr>
<td>C3</td>
</tr>
<tr>
<td>C4</td>
</tr>
</tbody>
</table>

Although it improves the coding efficiency, the variable codeword size in the Huffman code also limits the decoding throughput. The decoding process needs to identify the obscured codeword boundaries, which are now recursively dependent. Therefore the decoding of each codeword depends on the previously decoded codewords, leading to a generalized iteration bound on the decoding throughput. Here the iteration bound is somewhat different from the conventional iteration bound [2], because the decoding time for each codeword is variable. However, we can expect its average throughput is bounded by \( \frac{1}{t} \), where \( t \) is the average time for decoding a codeword. The practical significance of this iteration bound is that it limits the available throughput that can be achieved in a given IC technology.

In this paper, we propose three methods to relax the iteration bound. These methods generate concurrency without incurring significant overhead, thus effectively improving the throughput by orders of magnitude in a given implementation technology. Following our previous experience [3, 4], we solve the problem by considering the algorithms and the architectures jointly. The parallel decoding algorithms proposed in this paper break the dependency between subsequent image reconstructions in a flexible way. This means that the concurrency can be generated in a form suitable for parallel execution of other image processing algorithms. The advantages include the improvement in the system throughput and the simplification of the system design, such as data access.

The architectures considered in this paper are based on combinations of parallel and/or pipelined processing elements. The hardware is not strongly dependent on the parameters of the algorithms, and can be implemented in cost-effective IC technologies. All the methods can achieve, in a given technology, an arbitrarily high decoding throughput at the expense of an increased complexity proportional to the throughput.

In the next section, we describe several concurrent algorithms and compare their performance. Section 3 shows that these algorithms map into low-overhead concurrent architectures in a flexible way.

2. CONCURRENT ALGORITHMS

2.1. The Finite State Machine Technique

The first set of methods are generated from our technique [4] for synchronous finite state machines (synchronous FSM's), which are discrete-time finite state machines. The decoder for reconstructing Huffman-coded images is simply a special finite state machine that does not always generate output. For the code in our previous example, the finite state machine shown in figure 2 can identify the codeword boundaries and the associated codewords from the incoming bit stream when initialized at state S0.

![Figure 2. The Huffman coding tree.](image-url)
One of our methods, called the all-initial-state (AIS) method, is to break the coded bit stream into segments of the same size or different sizes, and decode concurrently for all segments and all possible leading states of each segment. For example, suppose a frame of coded sequence starts on our previous example code is 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1. 0. To reconstruct three times faster, we can divide the sequence into three segments: 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0. (The segments need not be of the same length.) The three segments can then be reconstructed concurrently based on figure 2 for each possible leading state, as summarized in table 2. Then these results can be combined hierarchically by matching the ending state of the preceding segment to the leading state of the current segment. For example, combining the decoded results of the first two segments in this way gives the results in table 3. The combined results in table 3 can be further combined with other results in the same way. Obviously the decoding throughput increases as the number of segments increases, and is theoretically unlimited.

The complexity of the method grows linearly with the increase in throughput and the number of states. The overhead of combining the concurrent decoding results can be made small by using longer segments. However, in many cases the actual complexity may become even lower due to special characteristics of the Huffman code used. For example, in our example the leading states for all possible leading states are exactly the same, so generally reducing the overhead in combining results.

Other methods of synthesizing concurrent FSM’s are also applicable. For decoders for the Huffman codes, or more generally FSM’s with non-linear recursive transitions, look-ahead [2] can improve the iteration bound, but with a complexity growing exponentially with the achieved throughput improvement [5]. One variation of look-ahead is discussed in [4], which linearizes the nonlinear recursion by processing the state transitions in a different domain. This linearization makes look-ahead feasible again, as the complexity now depends only linearly or logarithmically on the achieved throughput improvement as in linear adaptive filters [6]. Further variations can reduce the complexity dramatically [5]. For example, the coded bit stream within a frame can be segmented, where the leading state of each segment is hierarchically precomputed with linearized look-ahead. The segments within the frame are then processed concurrently, each decoded from its precomputed leading state. Again the iteration bound of the reconstruction process can be improved arbitrarily by increasing the number of segments within a frame. As illustrated in table 2, this approach offers a saving of 96% over the basic look-ahead method with linearized recursions while improving the throughput by 100 times.

Combining the AIS method with the linearized recursion leads to an implementation suitable for array architectures [5], which will be illustrated in section 3.

2.2. The Source Control Technique

The second set of methods is obtained by modifying the techniques for concurrent implementations of the Viterbi algorithm [3]. The source control methods in [7] generate concurrency for the decoding or reconstruction process by modifying the encoding process. For example, the block method [3] modifies the original Huffman code so that codeword boundaries appear every M bits, for example. Then the decoder divides the incoming bit stream into blocks of M bits each, and concurrently decodes these blocks. These methods essentially have zero computational overhead for the achieved throughput.

The modified Huffman encoder for the block method can be implemented in the following way: the encoder finds the codeword for the current input sample, buffers the codeword bits, and sends the bits out sequentially. A counter tracks the number of bits generated by the encoder, and resets every M bits for signaling the block boundaries. When the counter resets, the encoder checks if the codeword is complete. If so, the codeword boundary is aligned with the block boundary. If not, the encoder repeats the buffered bits of the incomplete codeword in the next block and then continues to complete the codeword. The repeated bits are clearly redundant, thus reducing the coding efficiency, but none the less essential for breaking the dependency or generating concurring for the reconstruction process.

The coding inefficiency can be made arbitrarily small by controlling the block-length parameter M without affecting the concurrency [7]. The loss in coding efficiency is defined as the ratio between the total redundant bits and the total bits, which is $R = \frac{M}{R}$, where $R$ is the average length of the redundant bits per block. Since $R$ is upper bounded by $C - 1$, where $C$ is the average codeword length, the loss $R$ decreases as the average codeword length gets smaller. This makes the Huffman code attractive for its ability of minimizing the average codeword length. As $M$ increases, the loss $R$ becomes smaller, as verified in table 4. For our previous example, the loss drops from 1.14% to 0.57% when $M$ goes from 50 to 100. The choice of the $M$ value may depend on the application. For example, in coding four quantization values with probability 0.35, 0.3, 0.2, 0.15, an even $M$ value can result in zero loss.

Other methods in [3, 7] are also readily applicable without any loss in coding efficiency. For example, the interleaved method [7] has high currency and full coding efficiency at the expense of higher encoder complexity. In this method, the samples are interleaved into $K$ independent Huffman encoders and coded independently. The coded bit streams from the encoders are then multiplexed on the bit level basis, which requires buffering for variable lengths of codewords. Then the multiplexed bit stream in fact contains $K$ independent coded bit streams. The decoder can simply apply pipeline self-interleaving [2] to decode each independent code word streams concurrently. As opposed to self-interleaving in the interleaved method, the multiplex method cross-interleaves codewords from different sources, such as from different frames or users. The sample stream from each source is coded independently, without prior interleaving, and then multiplexed on the bit level basis. The multiplexed coded bit stream still contains independent codeword streams that can be processed concurrently. The Interleaved method and the multiplex method have been collectively called the multi-streaming method in [3] for obvious reasons.

**Table 2. The results of independently decoding the three segments.**

<table>
<thead>
<tr>
<th>Leading State</th>
<th>Ending State</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S2</td>
<td>C2, C1, C1</td>
</tr>
<tr>
<td>S1</td>
<td>S2</td>
<td>C3, C1, C1</td>
</tr>
<tr>
<td>S2</td>
<td>S2</td>
<td>C4, C1, C1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Leading State</th>
<th>Ending State</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>C5, C1, C2</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>C4, C1, C1</td>
</tr>
<tr>
<td>S2</td>
<td>S0</td>
<td>C4, C2, C1</td>
</tr>
</tbody>
</table>

**Table 3. Combining the results from decoding the first two segments concurrently.**

<table>
<thead>
<tr>
<th>Leading State</th>
<th>Ending State</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>C2, C1, C1, C1, C2</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>C3, C1, C1, C2, C1</td>
</tr>
<tr>
<td>S2</td>
<td>S0</td>
<td>C4, C1, C1, C2, C1</td>
</tr>
</tbody>
</table>

173
Furthennore, but rather exploit the coding features and find the inherent

dependency is

image processing algorithms.

23. The Overlapped-Block Technique

feattures.

corcurrency.

process.

man

boundaries.

control

into the
dow.

dary

window,

the

decoding

of the methods from the

number

of

t. Image information can

free can be

assigned as

arbiuarily. Mer possibility is

generated

node of

method

assigns

the

header of

technique.

Figure 3. The overlapped-block method divides the coded bit

stream into blocks with overlapping windows of size $C_{\text{max}}$.

All these source-control methods set few limits on how the depen-
dency is to be broken, inherhlng high flexibility in generating concurrency.
Furthermore, the source control may also introduce concurrency for other
image processing algorithms.

Parallel to the comments in [7, 3] on the applicability of the source
control technique, the Huffman-coded images can be constructed con-
currently even without controlling the coding process if the source has spe-
cial coding features. In this case, we do not artificially break the depen-
dency but rather exploit the coding features and find the inherent
concurrency. For example, our example Huffman code has a feature that
zero can only appear in the coded bit stream as the last bit of a codeword.
Thus a subset of the codeword boundaries are observable by searching for
zero bits. We can segment the coded bit stream at these known codeword
boundaries, and decode these segments concurrently. More generally, some
bit patterns are only possible preceding or succeeding some states in a Huff-
man code. We seek specific bit patterns that mark the current state of the
decoder. Then we can segment the coded bit stream according to the loca-
tions of the patterns and decode from the associated states.

The "no-control" method is clearly code dependent, which limits its
applicability. The generated concurrency is also uncontrollable. How-
ever, the method can be made more accessible with minor control on the
coding process. For example, it may be possible to "create" the necessary coding
features with proper code assignments, as the branches of the Huffman cod-

ing tree can be assigned as 0 or 1 arbitrarily. Another possibility is to code

only with a fixed set of Huffman codes that have the required coding
features. Then the code identifier or the patterns to be searched can be put
into the header of the coded bit stream, for example. Of course, the no-
control method can also be used in conjunction with other methods to gen-
erate concurrency or to reduce the overhead.

2.3. The Overlapped-Block Technique

The third set of methods are the overlapped-block methods. As

shown in figure 3, the decoder divides the incoming bits into blocks of $M$
bits long, and assigns a window of $C_{\text{max}}$, bits long between adjacent blocks, where $C_{\text{max}}$ is the maximum codeword size. The
significance of this overlap is that a codeword boundary is known to fall
within it. Image information can thus be reconstructed concurrently for
these blocks as follows. For each block and each possible bit position in the
leading window of the block, assume the position as the codeword bound-
dary and decode. When the codeword boundary moves into the trailing
window, step and pair up the bit positions in the leading and trailing win-
don. These bit position pairs are then combined hierarchically to align the

codeword boundaries across all bits in the bit stream. For example, from
decoding the first block we can find the codeword boundary in the first trail-
ing window, which is also the leading window of the second block. Then
the codeword boundary in the trailing window of the second block can be
found by checking the pairing bit position. As the checking is relatively
expensive, the complexity of the method is approximately $O(C_{\text{max}})$. This
method is more effective than those from the finite state machine
(FSM) technique. To see this, we only need to observe that the complexity
of the methods from the FSM technique is at best proportional to the
number of states. When mapped to the Huffman coding tree, the number of
states equals to the number of junctions or nodes in the tree (including root),
while the maximum codeword length equals to the depth of the tree (the
number of branches in the longest path from the root). Because the depth
is never larger than the number of nodes in a tree, the bit-positioning tech-
nique is less complex than the FSM technique.

Table 4. Coding efficiency improves as $M$ increases. Two examples with different probability weighting are simulated.

<table>
<thead>
<tr>
<th>Sample Probabilities</th>
<th>$M=30$</th>
<th>$M=50$</th>
<th>$M=100$</th>
<th>$M=200$</th>
<th>$M=500$</th>
<th>$M=1000$</th>
<th>$M=2000$</th>
<th>$M=5000$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5, 0.25, 0.125, 0.125</td>
<td>1.9%</td>
<td>1.1%</td>
<td>0.57%</td>
<td>0.28%</td>
<td>0.11%</td>
<td>0.05%</td>
<td>0.003%</td>
<td>0.001%</td>
</tr>
<tr>
<td>0.5, 0.15, 0.025, 0.025</td>
<td>0.79%</td>
<td>0.48%</td>
<td>0.24%</td>
<td>0.12%</td>
<td>0.05%</td>
<td>0.024%</td>
<td>0.012%</td>
<td>0.005%</td>
</tr>
</tbody>
</table>

Variations of this method are certainly possible. One simple improve-
ment is to preprocessor the bits in the window to find the possible codeword
boundaries. This search eliminates those bit positions that are not used in
the subsequent pairing operation.

2.4. Summary

The choice of the methods depends on the applications and possibly
also on the data compression algorithms applied jointly with the Huffman
code. Our methods can concurrently reconstruct images in different pat-
ters, such as interleaved sets of scanning lines or disjoint spatial zones.
This flexibility is very important in improving the feasibility of our
methods. A comparison among our methods show that the first set of
methods is often less efficient than the others in terms of computational
complexity, while the second set of methods is the most efficient in compu-
tation but not in coding (with exceptions). However, all the methods show
that the iteration bound of reconstructing Huffman-coded images can be
related arbitrarily with a very small overhead.

The methods here can be combined arbitrarily. For example, combing
the source control technique and the bit position technique gives a
method that generates concurrency with a coding efficiency higher than that
for the source control technique, and with a complexity lower than that for
the bit positioning technique.

3. ARCHITECTURES

The concurrent algorithms do not strongly depend on their hardware
architectures. For example, the algorithms can be mapped to general com-
purers with multiprocessing, or to applications specific hardware in parallel
and/or pipelined form. Pipelined versions of the conventional decoders
require that the data within the pipeline latches are independent of one
another. A common architecture based on pipeline interleaving [2] is illus-
trated in figure 4, where independent coded bit streams are interleaved into
the pipelined decoder. The decoder is a pipelined version of the conven-
tional decoder modeled as a finite state machine. The number of interleaved
bit streams must match the pipelining level, which is four in the figure.

An alternative architecture is the hybrid pipelined architecture
described in [7]. The hybrid pipelined decoder, as shown in figure 5, is a
dynamic combination of parallel and pipelined decoders. The advantages
of this architecture are its greater generality and flexibility. The hybrid
pipelined decoder essentially serves all kinds of concurrent algorithms
under a wide range of system parameters and codes. The host computer or
the switch controller need only ensure the correct switching, which is usu-
ally scheduled dynamically. If scheduled dynamically, the hybrid pipelined
decoder can simultaneously reconstruct multiple source images under dif-
ferent concurrent algorithms. Using variable-rate buffers, high-throughput
decoding can be achieved with multiple low-throughput decoders in this
configuration [7].

Figure 4. Pipeline interleaving on a decoder for the Huffman
codes. Here four independent bit streams are interleaved into
the pipelined decoder, which is a finite state machine with four
stages of pipeline latches.
The concurrent decoder for the Huffman codes can be implemented with array architectures. The architectures and the comprising processing units can be derived following the approach in [4] for FSM's. First a trellis is formed by expanding the state transition diagram of the FSM against the time index. The operation of the FSM is then mapped onto the trellis, where the processing units effectively compute state transitions according to the input and the trellis configuration. More specifically, the processing unit computes a set of state indicators from the input and the previous state indicators [5], where an indicator flags 1 only if the associated state is the current state. The processing unit is usually implemented as a parallel combination of two-level gates. A processing unit for our example Huffman code is illustrated in figure 6, where part of the two-level gates degenerate into single-level gates. The output sample value can be generated from the internal signals of the processing unit.

The processing units can be cascaded and pipelined to form linear arrays, or further combined in parallel to form two-dimensional arrays. The input and output may need to be interleaved to match the algorithm to the architecture. The number of processing units in the array is the concurrency index of the decoder architecture, i.e., the number of jobs processed concurrently by the decoder.

Hardware concurrency does not always match the concurrency generated by the algorithm. For example, when the hardware concurrency index is smaller than that generated by the algorithm, a coded bit stream is served by multiple decoders. Conversely, when the hardware concurrency index is larger, a decoder can serve multiple coded bit streams.

Summarizing, the concurrency generated by the algorithms makes architectures simple, uniform, and flexible. Contrary to look-ahead, our algorithms require no preprocessing hardware. Instead, the main part of the required processing is identical. This makes hardware uniform and reusable in various ways. For example, decoding multiple Huffman codes, each at variable throughput rates, can be achieved by allocating and controlling resources dynamically in a hybrid-pipelined decoder or a resource pool, where the decoding resources are not necessarily identical.

4. CONCLUSIONS

Methods of concurrently reconstructing Huffman-coded images are discussed. These methods greatly reduce the computational or hardware complexity by breaking the dependency in the reconstruction process. As a result, the required hardware or computation has minimal overhead and a complexity that increases linearly with the achieved throughput improvement.

The methods of breaking the dependency provide flexibility as well as concurrency. The implementation of the Huffman decoder is not strongly dependent on the design parameters, and sometimes not even on the methods chosen. This implies higher resource utilization, as resources can be allocated and reconfigured dynamically.

Our previous results in [5] show that the iteration bound of any finite state machine can be improved by an arbitrary factor K with O(K) complexity. In this paper we showed that there may exist methods that are more effective than the general methods in [5]. Interestingly enough, all these methods reduce to only a handful of basic ideas. For example, mapping the states in the FSM techniques to bit positions leads to the overlapped-block technique. We find two empirical rules in adapting the ideas to different applications. First is that if the dependency is broken in a way matched to the nature of the recursion, the result is often more effective. For example, in generating concurrency for reconstructing Huffman-coded images, look-ahead and the finite state machine technique are less natural than the overlapped-block technique. Second is that more effective methods are often made possible by solving the problem at a higher system level. For example, considering our reconstruction problem beyond the algorithm and architecture level leads to the more effective methods in the source control technique.

We are now proceeding with the IC design of a decoder for Huffman-coded images in a low-cost low-power CMOS technology with a goal of a 200Mbps throughput.

REFERENCES