AN UNRESTRICTEDLY PARALLEL SCHEME FOR ULTRA-HIGH-RATE REPROGRAMMABLE HUFFMAN CODING

Robert A. Freking and Keshab K. Parhi

University of Minnesota, Dept. of ECE
200 Union St. SE, Minneapolis, MN 55455-0154
{rfreking, parhi}@ece.umn.edu

ABSTRACT

This paper proposes a comprehensive method for overcoming the inherently serial nature of variable-length near-entropy coding to obtain unrestrictedly parallel realizations of Huffman compression. A *codestream rearrangement* technique together with a *symbol-stream order-recovery* procedure form a concurrent approach capable of *exceeding* all previously attainable coderate figures. Furthermore, the method is noteworthy for achieving 100% hardware utilization with no coderate overhead while maintaining data output in a traditional streamed format. To further this endeavor, bit-serial encoder and decoder designs that possess compelling speed and area advantages are developed for service as parallel processing elements. However, both are suitable in more general contexts as well. The decoder, in particular, is optimally fast. The encoder and decoder designs are programmable, thus suggesting the appropriateness of the composite approach for a general-purpose ultra-high-speed codec. Benefits for low-power and variable-rate applications are briefly discussed.

1. INTRODUCTION

In the years since D. A. Huffman's acclaimed work on compression, the coding concept bearing his name has garnered increasing prominence among practitioners of data processing despite subsequent advances in the state of the art. Recent adoption in such influential standards as CCITT, JPEG and MPEG bode well for the continuation of this trend. Yet, this universal embrace belies serious encumbrances. Chief among them, variability in the length of the encoded outputs has thwarted application of traditional circuit acceleration techniques. Regular codeword boundaries are the mainstay of parallelism in particular. Absent this condition, multiple offset codewords cannot be located without first ascertaining the span of all preceding codewords. Because determination of codeword lengths is tantamount to decoding, serial codestream processing is mandated. This processing schedule is so strongly inherent in the Huffman algorithm that the great bulk of the literature focuses not on inter-codeword parallelism, but on parallelizing within the confines of the single-word decoding operation.

A variety of *ad hoc* approaches have been proposed to overcome the complications of unevenly-delimited input data streams at the intra-codeword level. Since maximum codeword lengths appreciably exceed the number of bits required by straight symbol enumeration, a brute-force solution which stores symbols indexed by full-length, arbitrarily-padded codewords affronts all design sensibilities by squandering area resources. Hence, clever configuration of the LUT dominates published results. In [1], the LUT is realized efficiently by means of a PLA. However, this style of architecture does not accommodate codebook programmability, which is a critical capability for maximizing compression ratios. A similar caveat applies to the reverse-binary-tree approach of [2], which requires rather specialized structural elements and interconnections. More conventional constructs are employed by methods which decompose the LUT into a series of sub-tables. Provided each sub-table is nearly or entirely homogeneous in code length, exponential storage redundancy is avoided. By dealing exclusively with a specialized subset of Huffman codes, viz., the Huffman-Shannon-Fano code, a numerical sequence property may be exploited to simplify sub-table addressing. In [3] sub-tables are formed according to consecutive bit patterns, while in [4] codeword-length equality is the basis of sub-table formation. Querying all sub-tables ranges in parallel permits one-cycle decoding.

Inter-codeword parallelism has been explored only in narrow contexts. In [5] it is shown that for diminutive alphabets a finite-state-machine specification can be manipulated to achieve concurrency in a non-programmable solution. A truly parallel scheme was suggested in [6]. However, that offering supported only non-standard parallel random-access output rather than the *de facto* format, streamed output.

The limitations of the foregoing methods underscore the logical inconsistency of parsing a serial data stream for which an *a priori*, data-independent demarcation rule has not been established. A recourse is provided by enforcing boundary regularity through code transformations. Although a codestream may be constructed accordingly by detaining data as necessary to achieve alignment at fixed intervals, as proposed on a block-level basis in [7], such overhead stands in contradiction to the objective of compression.

In this paper, a *codestream rearrangement transformation* is described which facilitates parallel scalability from the intra-word through inter-word level. Unlike previous methods, the degree of parallelism that may be attained is independent of algorithmic considerations. Additionally, architectures are introduced which enhance the utility of this scheme. For encoding, a tight programmable storage scheme is proposed with applications to both codebit-serial and one-cycle encoding. For decoding, a simplified realization of the tree representation of a codebit-serial decoder architecture is developed for high-speed operation. The composite effect of this approach is an unrestrictedly scalable compression scheme in which the pace of operation of individual processing elements (PEs) is slow enough that economical memory structures may be employed. In essence, this technique releases ultra-high-

This work was supported by the Army Research Office under grant number DAAG55-98-1-0203.

rate signal processing design from a codec bound, permitting optimization effort to be directed toward the more significant aspects of data presentation. With 100% PE utilization and no coderate overhead, this method may provide the only viable means of achieving a specific coderate target.

This paper is organized as follows. In section 2, the chosen codestream rearrangement is briefly explained. Section 3 illustrates the concept with an architectural template that includes the key supporting circuitry structures. Section 4 details a specific encoder implementation which offers advantages in both bit-serial and bit-parallel contexts. Section 5 offers an improved variation on decoder architectures with tree-based codebook representations. The design features a minimum critical-path delay. Finally, Section 6 concludes with assessments of the applicability of this promising scheme in key design contexts.

2. CODESTREAM REARRANGEMENT

The central impediment to inter-codeword parallelism in the Huffman compression algorithm is the inherent *a posteriori* discovery of codeword demarcations. To concurrently partition and route codewords to individual PEs, an unambiguous data-independent boundary heuristic must be known to both encoder and decoder. An aptly chosen rearrangement of the codestream can impose sufficient order to accomplish this goal. The specific technique employed in this work is m-fold interleaving, where m symbolizes the desired degree of parallelism. With this approach, re-ordered codewords are time-division multiplexed (TDM) at the encoder by allotting an invariant time-slot offset. In bit-serial fashion, each codeword is completely output via its assigned slot. Immediately thereafter a new codeword is allocated to the same slot so that PEs are 100% utilized. Thus, allocation proceeds according to PE availability, with symbols assigned in the order of arrival on the input symbol stream. This interleaved rearrangement process is depicted in Fig. 1. Note that longer codewords will occupy a slot proportionally longer than shorter codewords. In fact, if the shortest codeword is assumed to be of length 1, one PE may process up to b short codewords while another processes a single b-bit codeword. As is the practice, a fixed maximum codeword length, designated as $b_{\max}$, will henceforth be assumed.

3. PARALLEL SUPPORT STRUCTURE

From a high-level perspective, the design of an encoder that realizes this operation is particularly straightforward. The serial-to-parallel converter works as follows; using PEs which implement the bit-serial encoding process described in section 4, a symbol is drawn from the input symbol stream in FIFO order and delivery to the encoder PE of current index is attempted. If that processor is preoccupied, the processor of the succeeding index, modulo m, is queried for availability. This step is repeated until a free PE is located. The symbol is then deposited and the process repeats.

PEs are clocked m times slower than the serial-to-parallel-converter circuitry. Thus, one new symbol could potentially be required by each PE in a PE clock cycle. As is often the case, associated codeword lengths exceeding unity dictate less frequent symbol acceptance. By the very nature of variable-length coding, instantaneous symbol-stream to codestream size ratios fluctuate. Managing symbol-stream flow at both the encoder input and decoder output is, therefore, an issue common to all Huffman schemes and will not be addressed herein.

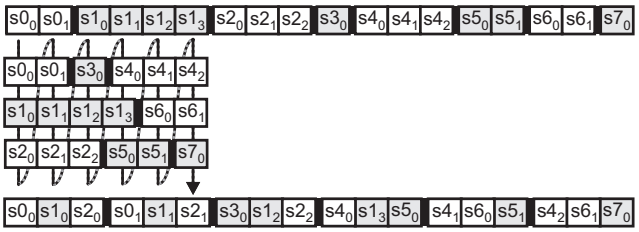

Figure 1: Stream rearrangement: entire codewords are allocated to the (prioritized) first-available PE channels. These are scanned in order bit-by-bit resulting in an output with embedded concurrency.

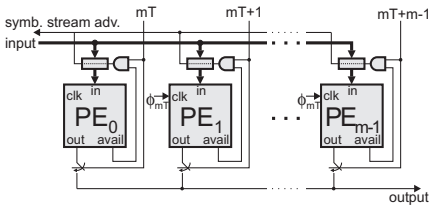

Figure 2: Basic support circuitry for the parallel encoder.

Deferring the internal workings of the PE for the present exposition, it suffices to recognize that one codebit per PE clock period per PE is produced. These are straightaway assembled into a codestream in the TDM manner, with the codebit of the PE indexed by $i \in (0, m-1)$ inserted at time period $mT + i$, where mT is the period of the PE clock. The described supporting circuitry for the encoder is diagrammed in Fig. 2.

3.1. Symbol-Stream Order Recovery

In the inverse process, the transmitted codestream is disassembled at the decoder, with the leading bit of an m-bit input block correlated to the PE of lowest index. At this point, each PE, to the extent possible, decodes the newly assigned bit as the initiation or continuation of a codeword. If the bit completes a codeword, the associated symbol is output to a local queue buffer. A prerequisite for the recovery of the original symbol-stream ordering is the postponement of retrieval of a subsequent symbol before a preceding symbol is decoded. Hence, the queue buffer is required for this function. Since, as observed previously, any one PE may process up to $b_{\max}$ 1-bit codewords before another finishes computation on a single $b_{\max}$-bit codeword, the queue must retain $b_{\max}$ decoded symbols of wordlength $\log_2 n$, where n is the cardinality of the symbol alphabet. To avoid the critical-path delay of a counter addressing structure, two cyclic shift registers of width $b_{\max}$, are employed to circulate tokens corresponding to the current input and output positions. Access to the registers comprising the queue is then regulated by entrance and exit transmission features, which are under the direction of the token registers.

The queue structure must be supplemented with a linear-shift notification register of width $b_{\max} + 1$ in order to reconstruct the original order. The arrival of the first bit of a new codeword is positively flagged in this register at the most significant end, whereas an intermediate bit is negatively flagged. Each cycle results in translation toward the externally-monitored least significant end.

Note that most-significant-bit (MSB) to least-significant-bit (LSB) traversal of the register is accomplished in precisely the same time required to decode the longest codeword. Thus, the symbol corresponding to a codeword flag received at the LSB position is guaranteed to have been previously advanced into the queue.

Supervisor logic operating at a clock rate m times faster than the PE clock rate scans PEs in sequence, examining the LSB position of each notification register for confirmation of symbol availability. If the indicator bit is affirmative, a symbol is dequeued and transferred to the output stream. As alluded to earlier, symbols are ejected onto the output stream intermittently in the characteristic manner of variable-length coding. The essential circuitry framework of the decoder is illustrated in Fig. 3.

It should be emphasized that the preceding circuit overviews are merely templates that may be tailored with the PE of choice. The succeeding sections will feature coder designs that are principally optimized for speed, with only secondary regard for area and power dissipation. Were either of the latter parameters deemed of higher precedence, different PE designs might be warranted.

4. ENCODER PROCESSOR DESIGN

Achieving substantial scalability requires that the replicated PEs possess exceptional efficiency attributes. To this end, a direct encoder design is proposed which has a fixed, low memory bound. Yet, it avoids many of the critical drawbacks of other methods by providing programmability while emitting the codeword in MSB-first order without post-encoding reversal. Furthermore, a critical-path delay of just over one memory-access period is achieved.

Despite its efficiency, the design is surprisingly straightforward. A random-access memory of size n and word length $b_{\max}+1$ is employed. The extra bit of word length permits a punctuation bit with a value of one to be appended to each codeword. The codeword so modified is stored MSB-justified, zero-padding as necessary, in the position indexed by the corresponding symbol. Encoding is accomplished by accessing the codeword using the symbol as an address into the resulting LUT. The retrieved codeword is immediately transferred into a shift register of the same word length. The MSB is output in every cycle with the shift register translating its contents one bit per cycle toward the MSB position. By injecting zero bits at the LSB position, codeword length may be ascertained implicitly by testing the $b_{\max}-1$ least-significant positions for nullity. When this condition is detected the process repeats with a new codeword.

Operating speed is clearly optimized in this design. The critical-path delay is expressed as $\tau_{lut}+\tau_{zero}$, where τ_{lut} is the delay of the LUT operation and τ_{zero} is the delay of the null-test operator. Furthermore, memory requirements are modest. In the literature, $b_{\max}$ is commonly selected as $2\log_2 n+\varepsilon$, where ε is 0 or small, e.g., for a 256-symbol alphabet, a maximum codeword length of between 16 and 20 bits is typical. A memory of size $n\times(2\log_2 n+1+\varepsilon)$ is thus required. Although, in terms of total bit expenditure better bounds have been demonstrated, a performance penalty always applies. This design is uniquely optimal in the sense that critical-path delay is reduced to near minimum for a programmable structure, yet memory sizes do not significantly exceed those of storage schemes which represent the related tree with pointers, such as in [8]. Moreover, possessing only n locations, internal address decoding in the memory unit is more economical with this direct approach than with the tree-based representations. Fig. 4 delineates the described encoding arrangement.

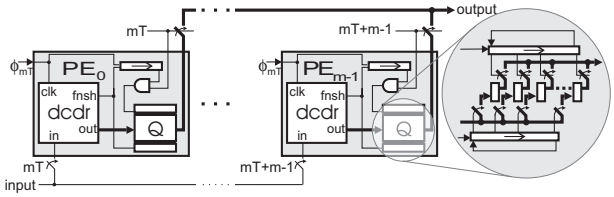

Figure 3: Support circuitry for the parallel decoder. Magnified inset: detail of the queue structure.

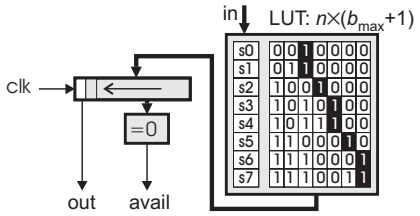

Figure 4: Schematic of encoder PE architecture. A conceptualization of the data storage technique is shown in the LUT.

5. DECODER PROCESSOR DESIGN

For bit-serial decoding, pointer-based representations of the Huffman tree lead to terse memory requirements. It has been demonstrated in [8] that it is feasible, through rather circuitous operations and assumptions, to use a memory with one word per existing tree node, where each word is just one bit wider than the bit width required to contain an arbitrary symbol. Since the quantity of nodes is invariant for a fixed-size alphabet, programmability is assured. In this section, a streamlined design with precisely the same memory requirements as the method mentioned above is presented, presupposing no properties of the codetree beyond those intrinsic to the original Huffman algorithm.

It is easily verified that a binary tree with n leaves is composed of exactly $2n-1$ nodes, regardless of configuration, provided every internal node possesses both children. If such a tree, excluding the root node, is linearized by enumerating nodes in breadth-first, left-to-right order, all left children will assume an even-numbered identity. Accordingly, if a memory indexed by these numerical identities stores a pointer to the left child of every node, the zero-valued LSB need not be saved. Thus, the width of a pointer is coincidentally truncated to the same bit width as required to store a symbol, i.e., $\log_2 n$ bits. A single memory unit may contain both data types if some means of distinction is provided. Appending one diacritical bit per memory location serves this purpose. In this arrangement, a symbol value is recorded in each memory location corresponding to a leaf. To be explicit, a memory of realistic size $2n\times(\log_2 n+1)$ implements the bit-serial LUT. Note that, unlike other methods which require the numeric-sequence property of the Huffman-Shannon-Fano code to obtain this level of economy, this decoder design is burdened by no such restriction. It should be understood that this specification is not minimum; it can be shown that the memory span between node and left child is at most $n-1$ locations; further, this extreme has the potential to occur only at the penultimate level of the tree. Consequently, if

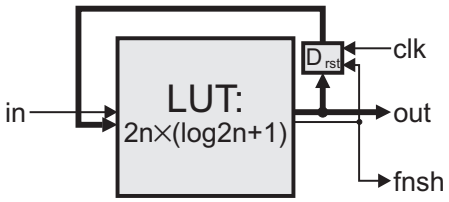

Figure 5: Diagram of the decoder PE architecture. Minimal critical-path delay and overall simplicity of design are evident.

offsets were favored over absolute addressing, an extra bit could easily be trimmed from the pointer representation. Less direct addressing schemes have been devised, resulting in ever more petite memory profiles. Notwithstanding, even the maximum span observation will be rejected in this solution to avoid the delay of an offset addition.

The architectural composition of this design assumes an unusually trivial form. Given the described order of enumeration, right-child nodes immediately succeed left-child nodes in memory. Therefore, the current codebit value fortuitously augments the left-child pointer in the absent LSB position to form the address of the next node. Starting with an address of zero, the entire codeword may be completely decoded without arithmetic operations or comparisons by recursively retrieving the content indexed by the address amalgam until the diacritical bit signals a symbol. The decoded symbol is dispatched as output to flow-regulation circuitry – in this case, the local queue structure – and the process repeats with a new codeword. The structure of the decoder core is visualized in Fig. 5. Clearly, at one LUT access, the critical-path delay is minimal for a programmable design.

6. CONCLUSIONS

Four facets of ultra-fast Huffman codec design have been explored: the decoder PE, the encoder PE, the encasing parallel control and support logic, and the codestream rearrangement scheme. The latter overcomes the problem of irregularly delimited codeword boundaries through the mechanism of m-fold interleaving. The resulting symbol dispersion is reversed with the order-recovery process implemented by way of the decoder support circuitry, thus enabling a traditional streamed output format. The optional encoder and decoder designs promote even greater performance yet boast relatively low area requirements. Because these processors function in a parallel context, intricate hardware constructs, such as barrel shifters, are entirely avoided. While tailored specifically for the parallel approach that is the subject of this paper, both designs are auspicious for stand-alone applications as well.

Given the rate-enhancing capabilities inherent in the codestream-rearrangement technique itself, the rationale behind the strategy of targeting the PE clock rate as the primary objective and consumed area as only a secondary objective may not be apparent. In fact, any LUT storage approach that compacts memory beyond one location per item must employ some style of arithmetic or other whole-word operation to decompact. Assuming modern memory implementations, such operations have the potential to roughly halve the clock rate. Thus, approximately twice as many PEs would be required to maintain throughput. Since no approach has been presented which reduces memory to significantly less than half of the sizes demonstrated herein, it is apparent even at a superficial level that overall area expenditures will be comparable. In practice, when the area of additional arithmetic logic and replicated PE support-and-control hardware overhead is accounted for, a design with a smaller LUT is seen to be more costly. Furthermore, the simplicity of the proposed designs avoids other practical difficulties, such as irregular layouts and congested wiring. Curbing critical-path delays is advantageous from the converse perspective as well; inexpensive standard memory/logic structures or less sophisticated fabrication processes may be exploited for high-rate applications.

Because the codebook is reprogrammable, the approach expounded in this paper is suitable for a general purpose codec. Furthermore, static or on-the-fly reconfiguration for various rates is possible by disabling PEs, i.e., support circuitry could be instructed to scan abbreviated subsets of PEs and, if desired, idled PEs could be powered down. Such quality-of-service regulation techniques have been shown to yield power dissipation benefits in unrelated applications. However this approach, even without such modifications, offers innate power dissipation benefits due to its scalability. A well-known design principle asserts that increasing parallelism while maintaining a specified throughput rate by adjusting supply voltage permits considerable power savings to be achieved. This scheme, being scalable without overhead, is able to extract full advantage from supply-voltage manipulation.

Though the price of a parallel approach is not insignificant in terms of consumed area, the role of nearly unlimited concurrency in achieving heretofore unapproachable coderates is overriding. Because the codestream-rearrangement technique is optimal on so many fronts – in particular, 100% PE utilization, no coderate overhead, linear speedup and a streamed output format – the technique is applicable in the broadest range of settings.

7. REFERENCES

[1] S. M. Lei and M. T. Sun, "An entropy coding system for digital HDTV applications," *IEEE Trans. Circuits Systs. Vid. Tech.*, vol. 1, no. 1, pp. 147-55, 1991.

[2] A Mukherjee, N. Ranganathan and M. Bassiouni, "Efficient VLSI designs for data transformation of tree-based codes," *IEEE Trans. Circuits Systs.*, vol. 38, no. 3, pp. 306-14, 1991.

[3] H. Park, J. C. Son and S. R. Cho, "Area efficient fast Huffman decoder for multimedia applications," in *Proc. ICASSP 1995*, vol 5., pp. 3279-81, Detroit, May, 1995.

[4] B. W. Y. Wei and T. H. Meng, "A parallel decoder of programmable Huffman codes," *IEEE Trans. Circuits Systs. Vid. Tech.*, vol. 5, no. 2, pp. 175-8, 1995.

[5] K. K. Parhi, "High-speed VLSI architectures for Huffman and Viterbi decoders," *IEEE Trans. Circuits Systs. II*, vol. 39, no. 6, 1992.

[6] P. G. Howard and J. S. Vitter, "Parallel lossless image compression using Huffman and arithmetic coding," in *Proc. Data Comp. Conf.*, pp. 299-308, Snowbird, Utah, Mar., 1992.

[7] H. D. Lin and D. G. Messerschmitt, "Designing a high-throughput VLC decoder part II – parallel decoding methods," *IEEE Trans. Circuits Systs. Vid. Tech.*, pp. 197-206, 1992.

[8] H. Park and V. K. Prasanna, "Area efficient VLSI architectures for Huffman coding," *IEEE Trans. Circuits Systs. II*, vol. 40, no. 9, pp. 568-75, 1993.