This paper appeared in Proc. of Int. Symp. on Inform. Theory and Its Applications, Victoria, Canada, 1996

A High Throughput Multiplication Free Approximation to Arithmetic Coding

Frank May, Andreas Klappenecker*,
Volker Baumgarte, Armin Nückel, Thomas Beth
Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5, 76128 Karlsruhe
email: wavelet@informatik.uni-karlsruhe.de

Abstract — **Several solutions were proposed to avoid costly multiplications in approximations to arithmetic coding. These methods rely on repeated renormalizations which turn out to be the bottleneck in VLSI implementations. We propose a new renormalization scheme that achieves significiantly higher throughput in terms of encoded symbols per clock cycle and give some details on a VLSI implementation of this scheme.**

I. Introduction

Arithmetic coding [1] is a fixed precision version of the Elias coding [2, pp. 61–62] and is widely used as a final step in complex compression systems [3, 4]. It achieves the zero-order entropy asymtotically for arbitrary probability distributions. In contrast to Huffman coding (which is in general not optimal, but allows very efficient implementations), arithmetic coding involves expensive multiplications. Several authors have proposed approximation techniques to avoid these multiplications [5, 6, 7, 8]. In VLSI implementations, these methods require multiple clock cycles for each encoded symbol (e. g. [5]). Unfortunately, the number of cycles is even higher for lower compression ratios. This may be one reason, why arithmetic coding is rarely used in hardware implementations. We propose a new renormalization scheme which allows to encode *one symbol per clock cycle.* Furthermore we show that these modifications lead to area efficient CMOS circuits. This research is part of a VLSI wavelet compression project at the authors' institution.

II. Arithmetic Coding

A sequence of symbols (s_i) over a finite alphabet $\{a_0, \ldots, a_{N-1}\}$ produced by a source with probabilities $p(a_k) > 0$ is encoded by repeated subdivision of the interval $[0, 1)$. The current interval is represented by its lower bound C_i and its width A_i. The subdivision is given by the recurrence equations:

$$\begin{array}{lll} A_0 = 1, & A_{i+1} = p(s_i) \cdot A_i, \\ C_0 = 0, & C_{i+1} = C_i + P(s_i) \cdot A_i, \end{array} \qquad (1)$$

where $P(a_k)$ denotes the sum $\sum_{l=0}^{k-1} p(a_l)$. A finite sequence $(s_0, \ldots, s_{K-1})$ can be decoded from the value C_K.

Since the probabilities have to be stored in finite precision registers, the probabilities p are approximated by dyadic rationals $\widehat{p}$ with a precision of α bits, such that

$$2^{-\alpha} \leqslant \widehat{p}(a_k) < 1 \quad \text{and} \quad \sum_{l=0}^{N-1} \widehat{p}(a_l) \leqslant 1. \qquad (2)$$

Assume that a_{N-1} is the most probable symbol. As the sum of the approximated probabilities may be less than one, we add the difference to $\widehat{p}(a_{N-1})$, that is, $\widehat{p}(a_{N-1}) := 1 - \widehat{P}(a_{N-1})$.

The values A_i and C_i should also be represented by fixed precision registers A and C, though it follows from (1) that A_i and C_i in general require an arbitrary number of bits. Note that the sequence of the interval widths (A_i) is monotonically decreasing. It follows that for small A_i the addition of $P(s_i) \cdot A_i$ does not change the leading bits of C_i. At least these constant bits can be communicated to the decoder and A and C can be renormalized. The renormalization is achieved by shifting left both registers A and C, such that A always lies in a given interval.

The addition in (1) may result in an carry-overflow in register C. The bits shifted out of C are passed through an additional shift register C' to which the carries are added. This resolves most of the overflow carries. The bits shifted out of C' are blocked in an output buffer and then communicated to the decoder. If all bits in this output buffer are '1', the addition of a carry must be propagated further. This carry is stored in an additional bit (called *stuffbit*) so that the addition can be finished by the decoder, cf. [5].

III. Renormalization

Previous arithmetic coders keep A in a tight interval, e. g. the interval $[0.5, 1)$ used in [6, 7]. Then, coding one symbol leads in general to *several shifts* for renormalization [5]. Thus, several clock cycles are required for renormalization, prohibiting a continuously processed input stream. This effect is even worse for larger alphabeths. Apart from the obvious gain in throughput, continuous processing simplifies the design of a complex compression chip, since it is possible to avoid

*This research was supported by DFG under project Be 887/6-3.

costly internal synchronization logic and allows the use of dynamic CMOS circuits in line buffers.

Assume that the arithmetic coder has a parallel output with β bits, i. e. in each clock cycle either zero or β bits are released. To achieve high throughput, we keep A in the larger interval $[2^{-\beta}, 1)$ and additionally require $\alpha \leqslant \beta - 1$. Thus, a *single shift by β or $\beta - 1$ bits* (depending on stuffbits) is sufficient for renormalization. At the same time β bits are released to the channel. In hardware, the single shift can be realized by simple multiplexers and does not require additional clock cycles. The gain is illustrated in Figure 1.

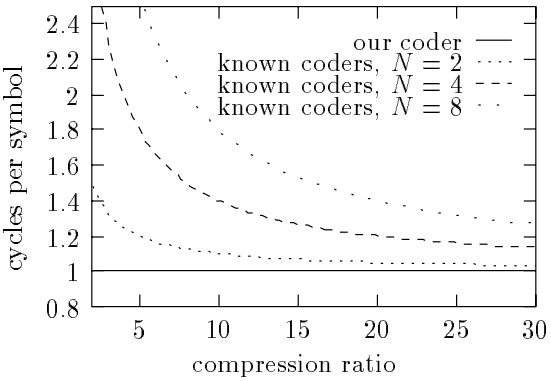

Figure 1: The diagram shows the performance of our coder compared to previous coders with different alphabet sizes N in terms of the average clock cycles needed to encode one input symbol. Note that the performance of our coder does not depend on N.

IV. Approximation

As the two multiplications in (1) are costly in terms of chip size and throughput, we apply the method proposed by Feygin et al. [7] to approximate A, thereby reducing each multiplication to one addition/subtraction and two shifts. Depending on A, there are two types of approximation:

$$\widehat{A} = 2^{-s}(2^{-1} + 2^{-(i+1)}), \quad i = 1, \ldots, \gamma - 1, \qquad (3)$$

$$\widehat{A} = 2^{-s}(1 - 2^{-i}), \qquad i = 1, \ldots, \gamma, \qquad (4)$$

where $s \in \{0, \ldots, \beta-1\}$ is the number of leading '0' bits in A and γ is the maximum number of bits used for the approximation. The exponent i is chosen to maximize $\widehat{A}$ under the constraint $\widehat{A} < A$. The approximation of $\widehat{A}$ in case (3) can be visualized as

$$\widehat{A} = 0.\underbrace{0\cdots0}_{s}1\underbrace{0\cdots01}_{i\leqslant\gamma-1}0\cdots$$

and in case (4) as

$$\widehat{A} = 0.\underbrace{0\cdots0}_{s}\underbrace{1\cdots1}_{i\leqslant\gamma}0\cdots.$$

In [7] it is shown that the worst case normalized excess length introduced by the approximation is less than 0.8 %.

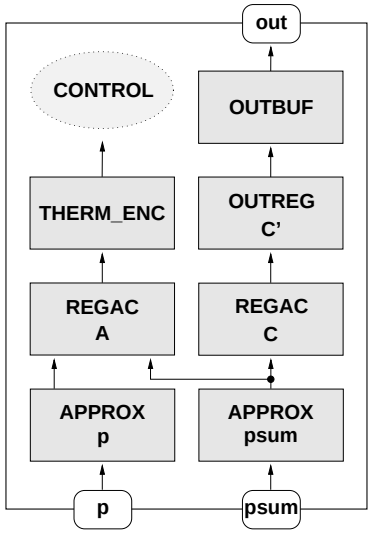

Figure 2: This block diagram sketches the architecture of the arithmetic coding chip. The main modules `APPROX` and `REGAC` are used twice, easing the layout process of the chip.

V. VLSI Implementation

Our algorithm is very suitable for VLSI implementations. A rough sketch of the architecture is given in figure 2. The inputs `p` and `psum` supply the probability $\widehat{p}(s_i)$ and the cumulative probability $\widehat{P}(s_i)$ respectively. The two `APPROX` modules compute the approximated products $\widehat{A}\cdot\widehat{p}(s_i)$ and $\widehat{A}\cdot\widehat{P}(s_i)$ which are used to update A and C in the `REGAC` modules. The `THERM_ENC` module computes the number of shifts needed by the approximation. The output of C is passed through the `OUTREG` C' and through `OUTBUF`. `OUTREG` and `OUTBUF` handle the overflow problem. Finally, β bits at a time are released via `out`.

We formulated this algorithm in the high level hardware description language ELLA. This abstract description was translated into a gate level language as input to a VLSI CAD tool. We have designed a set of full custom layouts using a 1μ dual metal CMOS process to improve the geometric and electrical efficiency. A electrical worst case simulation of the layout showed a minimum of 33 MHz clock frequency. Figure 3 shows the final layout of the `APPROX` module and Figure 4 of the `REGAC` module. Note that both modules are used twice.

Table 1 shows some simulation results of the circuit. Each line gives the zero order entropy of the encoded sequence, the resulting bit rate and the *normalized excess code length,* i. e. the difference between bitrate and entropy normalized by the entropy.

Compared to the architecture described in [7], the width of the registers A and C is increased by β bits and two additional shifters are necessary to scale the product approximations appropriately. On the other hand, we were able to simplify the control logic signif-

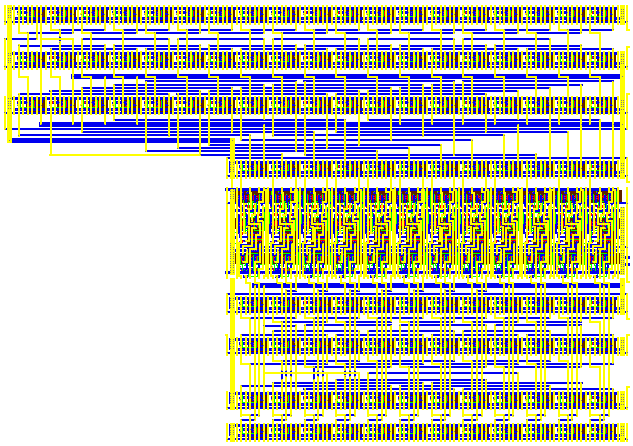

Figure 3: The final layout of the APPROX module with about 2,300 transistors per mm^2.

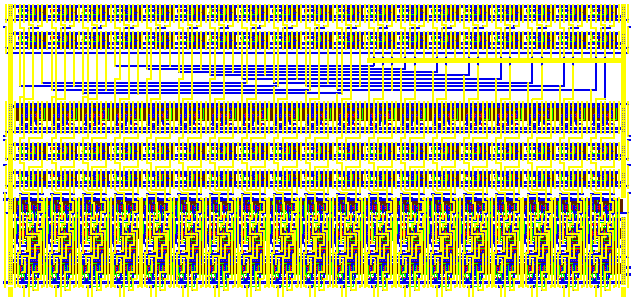

Figure 4: The final layout of the REGAC module with about 3,600 transistors per mm^2.

icantly.

Another algorithm which achieves high throughput using pipelining and fast multiplications is described in [9]. However, these multiplications are implemented with huge lookup tables which are very costly.

VI. Conclusion

Many image and video compression algorithms include the following three steps: a signal transformation, a quantizer, and a traditional entropy coder. The first two steps typically lead to highly skewed probability distributions. Thus, an arithmetic coder is a natural and efficient choice. However, most commercially available compression chip sets use a less efficient Huffman coder, since this coder achieves significantly higher throughput than previously known arithmetic coders with comparable chip size. The proposed algorithm combines both high throughput and efficient coding and is cost effective in terms of chip size.

VII. Acknowledgements

The VLSI system and layout libraries were developed under the DFG project IDEAS [10]. The support by DFG is gratefully acknowledged.

Entropy	Bit Rate	Excess Rate
0.2009	0.2014	0.24 %
0.3390	0.3397	0.19 %
0.5440	0.5449	0.15 %

Table 1: The tabular shows the results of simulation.

References

[1] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Comm. ACM*, vol. 30, pp. 520–540, June 1987.

[2] N. Abramson, *Information Theory and Coding*. McGraw-Hill, New York, 1963.

[3] W. B. Pennebaker and J. L. Mitchell, *JPEG still image compression standard*. Van Nostrand Reinhold, 1993.

[4] A. Klappenecker and F. U. May, "Evolving better wavelet compression schemes," in *Proc. of Wavelet Applications in Signal and Image Processing III, 12–14 July 1995, San Diego, California*, pp. 614–622, SPIE, 1995.

[5] R. B. Arps, T. K. Truong, D. J. Lu, R. C. Pasco, and T. D. Friedman, "A multi-purpose VLSI chip for adaptive data compression of bilevel images," *IBM J. Res. Develop.*, vol. 32, pp. 775–795, Nov. 1988.

[6] D. Chevion, E. D. Karnin, and E. Walach, "High efficiency, multiplication free approximation of arithmetic coding," in *Proceedings of the Data Compression Conference, Snowbird, Utah*, pp. 43–52, 1991.

[7] G. Feygin, P. G. Gulak, and P. Chow, "Minimizing excess code length and VLSI complexity in the multiplication free approximation of arithmetic coding," *Inform. Processing & Management*, vol. 30, no. 6, pp. 805–816, 1994.

[8] T. Y. Tong and I. F. Blake, "An improved multiplication-free multialphabet arithmetic code and the redundancy of arithmetic codes," *to appear*, 1992.

[9] H. Printz and P. Stubley, "Multialphabet arithmetic coding at 16 MBytes/sec," in *Proceedings of the Data Compression Conference, Snowbird, Utah*, pp. 128–137, 1993.

[10] T. Beth, A. Klappenecker, T. Minkwitz, and A. Nückel, "The ART behind IDEAS," in *Computer Science Today* (J. van Leeuwen, ed.), vol. 1000 of *Lecture Notes in Computer Science*, pp. 141–158, Springer Verlag, 1995.