FAST LOSSLESS IMAGE COMPRESSION WITH 2D GOLOMB PARAMETER ADAPTATION BASED ON JPEG-LS

*Z. Wang, M. Klaiber, Y. Gera, S. Simon**

Dept. of Parallel Systems, Univ. of Stuttgart
70569 Stuttgart, Germany
e-mail: wangze@ipvs.uni-stuttgart.de

Th. Richter

RUS Computing Center, Univ. of Stuttgart
70550 Stuttgart, Germany
e-mail: richter@rus.uni-stuttgart.de

ABSTRACT

A Fast and Lossless Image Compression (FLIC) algorithm based on the median edge predictor and Golomb coder of JPEG-LS is presented. FLIC eliminates the gradient-based context model from the JPEG-LS standard, the most expensive parts with respect to computational complexity and memory space requirements. To avoid a large context memory, Golomb parameter is selected based on the coding states and the prediction residuals of up to two immediate neighbors, one in each dimension. The FLIC algorithm has low memory footprint and dissolves the data dependencies in JPEG-LS to facilitate parallelization. Experimental results show that the FLIC algorithm achieves a throughput speedup factor of 3.7 over JPEG-LS with less than 4% compression performance penalty. Lossless compression performance results further show that FLIC outperforms other state-of-the-art standards including JPEG 2000 and JPEG XR.

Index Terms— Lossless image compression, low complexity coding, adaptive coding, parallelization, JPEG-LS

1. INTRODUCTION

Lossless image compression has been a challenging research topic in the past decades. In the late 90s CALIC [1] and LOCO-I [2] were introduced, which still count as the most competitive algorithms today in terms of compression performance and relatively low computational complexity. Because of the better overall efficiency, LOCO-I was later standardized as JPEG-LS [3]. Since 2000, although algorithms with improved compression performance have been proposed [4–7], less and less improvement in compression have been achieved at significantly higher complexity than JPEG-LS.

On the other hand, both the computing architecture and the application domain have evolved considerably since the 90s. Today various parallel computing architectures like multi-core CPUs, GPUs and FPGAs are widely used. Data dependencies in an algorithm limits the potential for exploiting these parallel architectures, which could be crucial for image compression tasks where the amount of input data is large. Unfortunately, for algorithms like JPEG-LS, the principle of context modeling introduces data dependency. Without special optimizations, this limits JPEG-LS to sequential processing and puts upper bounds on the achievable throughput. Meanwhile, memory-and-power-constrained embedded image processing applications are emerging, e.g. image capturing systems on mobile devices and cars. The large number of conditioning contexts in JPEG-LS (up to 367) requires a considerable amount of memory, a disadvantage for on-chip hardware implementations.

Relatively few works have considered the balance among compression performance, complexity, memory efficiency and parallelization. Two of the earlier proposals for low complexity lossless image compression are FELICS [8] and SFALIC [9]. FELICS uses a simple predictor based on two neighboring pixels and a Golomb coder, and has a similar complexity to JPEG-LS but about 15% lower compression ratio. SFALIC uses a set of 9 predictors and a Golomb coder and achieves about 3.6 times the throughput of JPEG-LS at 10-12% higher bit rate for 8-bit images. In FELICS, contexts are formulated as the prediction residual of the previous pixel combined with all possible states of the Golomb coder, while in SFALIC quantized prediction residuals are used to reduce the number of contexts. For 8-bit images, FELICS requires $2^8 \times (8-1) = 1792$ contexts while SFALIC requires $8^2 \times (8-1) = 448$ contexts. Apart from the large requirement for context memories, their sequential Golomb parameter adaptation [8] creates an issue for parallelization.

This work achieves low computational complexity by eliminating the context model from JPEG-LS. The result is a very fast lossless image compression (FLIC) codec both memory efficient and parallelizable, with closely comparable compression performance to JPEG-LS. The rest of this paper is organized as follows. Section 2 briefly reviews JPEG-LS and analyzes its memory requirement and data dependency. The proposed algorithm is first presented in its 1-dimensional adaptation form and then extended to the 2-dimensional adaptation form in Section 3. Section 4 introduces a de-emphasis method for the limited-length Golomb code. Experimental results on the throughput compared with JPEG-LS and compression performance compared with JPEG-LS, FELICS, JPEG 2000 and JPEG XR are presented Section 5. Finally Section 6 concludes this paper.

2. REVIEW OF JPEG-LS: MEMORY REQUIREMENT AND DATA DEPENDENCY

The JPEG-LS standard is based on the concept of spatial prediction followed by residual coding. First, a median edge-detecting predictor [2] is used to predict the value x of the

*The authors would like to thank the German Research Foundation (DFG) for the financial support. This work has been carried out within the research project Si 586 7/1 which belongs to the priority program DFG-SPP 1423 "Prozess-Spray".

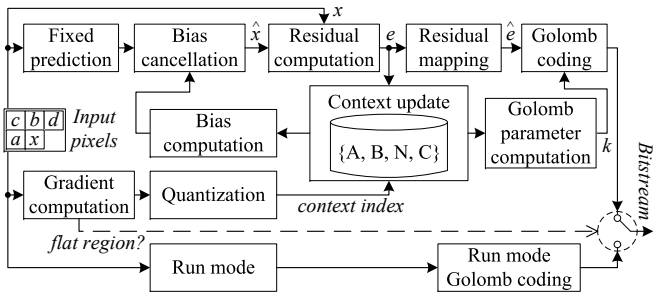

Fig. 1. Functional blocks of JPEG-LS.

current pixel depending on the known neighboring values a, b and c (cf. Fig. 1). Since the predictor requires pixels form the upper row, a memory efficient implementation strategy is to use a line buffer of size $\boldsymbol{w}+1$ for images with $\boldsymbol{w}$ pixels in each row. The required memory for prediction is $M = bpp \cdot (\boldsymbol{w}+1)$ bits, where bpp is the input sample precision.

The prediction residual is refined by a context-dependent bias cancellation term. Context is formulated from the *gradients* of surrounding pixels as $\{d-b,\ b-c,\ c-a\}$. To avoid context dilution and reduce the required memory, the gradients are quantized and mapped to a set of up to 365 contexts. Finally, the residual e between the actual value x and the bias-compensated prediction $\hat{x}$ is calculated, mapped to non-negative value $\hat{e}$ and encoded by a Golomb coder. Additional steps are performed to update the four state variables of the current context with the residual. These variables are used to adapt the Golomb parameter and the bias cancellation term to the residual distribution under the current context. As estimated in [10], for 8-bit images the four state variables require at least 34 bits of memory, and the entire context model requires $34 \cdot 365 = 12{,}410$ bits. JPEG-LS defines an alternative run-length encoding mode dedicated to flat regions, as shown in the bottom part of Fig. 1. The run mode delivers excellent performance for synthetic images but is less significant for natural images, where flat regions rarely occur.

While the context model enables higher-order entropy coding, it causes data dependencies among pixels under the same context, i.e. the encoding of the current pixel depends on the results of previous encoding steps on same-context pixels, as illustrated in Fig. 2. Such pixels must be encoded sequentially. Unfortunately, the probability for two neighboring pixels to be in the same context is as high as 83% [11]. Although parallelization schemes such as delayed context update [12] and context classification with pixel reordering [13] have been proposed, the added computational complexity and memory requirement for synchronization still create bottlenecks for high speed implementations.

3. PROPOSED FLIC ALGORITHM

One rarely answered question is whether the computational efforts, memory requirements and data dependencies implied by context modeling are the necessary price for a competitive compression performance. In this section, we first eliminate context modeling from JPEG-LS based on insights into the complexity of context modeling and its effects on the residual distribution. We then propose a fast and memory efficient alternative to the Golomb parameter adaptation rule in JPEG-LS. Next the proposed adaptation is extended to the 2-dimensional case for an improved performance. Finally the advantages of FLIC with respect to parallelization is discussed with three exemplary parallelization schemes.

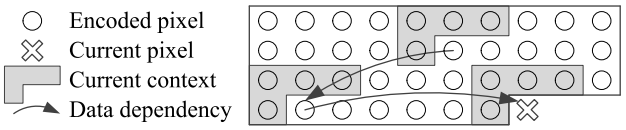

Fig. 2. Data dependencies in JPEG-LS.

3.1. Elimination of context model in JPEG-LS

Functional blocks related to the context model are the most complex algorithmic parts of JPEG-LS, including gradient computation, quantization, context update, bias computation and Golomb parameter computation. Many of them require conditional branches costly for both software and hardware. For gradient quantization, small pre-computed lookup tables can be used in a hardware implementation or an optimized software implementation, whereas for blocks like bias computation and Golomb parameter computation, either much less room for optimization exists or large multi-dimensional lookup tables are required, increasing cache misses on CPUs and on-chip memory requirements on FPGAs. With these insights, we decided to omit the gradient-based context model and evaluate the effect on the residual statistics.

Due to the lack of context based bias cancellation, the distribution of the mapped residual value $\hat{e}$, measured on the ITU test image set, shows a slightly worse characteristic identifiable by the broadened distribution depicted in Fig. 3. It can be seen that the distribution remains very close to that in the original JPEG-LS (one-sided geometric distribution, OSGD). Because no more context information is maintained, the memory requirement is reduced to $M = bpp \cdot (\boldsymbol{w}+1)$ plus some internal variables, and is hence bounded by the image dimensions. In addition the data dependency throughout the image is no longer present. This property is of particular importance for data decomposition, and therefore for parallelization schemes. The basic structure of the FLIC encoding algorithm is now simplified to fixed predictor, residual computation, residual mapping and entropy coding. Since without bias cancellation the residual still follows an TSGD (Laplacian) distribution with only slightly worse characteristics, we chose a Rice variant of Golomb coder which is known to be optimal for such sources, and design an algorithm to select its coding parameter quickly.

3.2. Fast memory efficient Golomb parameter adaptation

A Golomb coder with parameter m represents its input symbol $\hat{e}$ by first encoding the value of $\lfloor \hat{e}/m \rfloor$ by a unary code followed by a binary representation of the remainder $\hat{e} \mod m$. This strategy becomes of course extremely simple if $m = 2^k$, i.e. m is a power of two, since the division can be implemented by shifting. The length of the code then is then

$$L(\hat{e}) = \left\lfloor \frac{\hat{e}}{2^k} \right\rfloor + 1 + k \ .$$

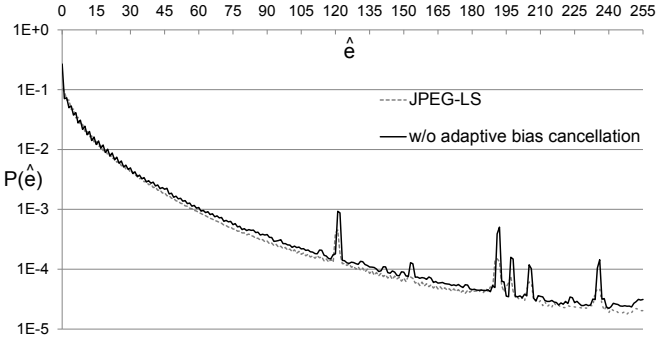

Fig. 3. Statistics of the mapped prediction residual for the original JPEG-LS and the proposed scheme.

For geometric sources with a non-negative alphabet, the appropriate value of k is known to be

$$k = \max\left\{0, \left\lceil \log_2 \frac{\mathbb{E}(\hat{e})}{2} \right\rceil\right\} , \qquad (1)$$

where $\mathbb{E}$ denotes the expectation value, see [14] for details. Because the residual distribution is approximately Laplacian, the original JPEG-LS now estimates the mean of the absolute value of residual e ($|e| \approx \frac{\hat{e}}{2}$) from previous pixels and uses backwards adaptation on the encoder and the decoder to select k. Unfortunately, this strategy adapts only very slowly on the image statistics and is only optimal under the assumption of an ideal Laplacian residual distribution, whereas the actual residual distribution is better described by the more general family of generalized Gaussian sources. Thus, an alternative approach of determining k following the idea of [15] is employed here: Let k_n be the parameter for the next coding step, k_{n-1} that of the previous coding step and $\hat{e}_{n-1}$ the mapped residual of the previous step. Then the adaptation process reads as follows:

$$k_n \leftarrow \max\left\{0,\ k_{n-1} + \left\lceil \log_2\left(\left\lfloor \frac{\hat{e}_{n-1}}{2^{k_{n-1}}} \right\rfloor + 1\right)\right\rceil - 1\right\} , \quad (2)$$

i.e. the adaptation tries to reduce the number of binary bits if the unary code part $\lfloor \hat{e}_{n-1}/2^{k_{n-1}} \rfloor$ has length zero because the previous symbol was rather small. If, however, the unary part has a length longer than zero, the value of k is gradually increased to adapt to a higher error variance. This is slightly different from the adaptation mechanism described in [15] which employs a "fractional" adaptation rule using two parameters, k_R similar to our k which is controlled through an additional state variable $k_{RP} = L \cdot k_R$, and k_{RP} is adjusted to the statistics similar to the adaptation rule (2) preceding. However, in trying to further to reduce the complexity of the coding process, we decided to use only a single state variable. Hence, the value of k_n is all that needs to be stored for the next coding step.

The adaptation rule (2) can be easily implemented by a lookup table, which then requires $(k_{\max} + 1) \times (q_{\max} + 1)$ entries only, where $q_{\max}$ is the maximum possible value for the quotient $\lfloor \hat{e}/2^k \rfloor$. The table size is necessarily limited, i.e $k_{\max} < \infty$ because even in the worst case $\hat{e}_{n-1} = 2^{bpp} - 1$ the iteration

$$k_0 \leftarrow 1, \quad k_n \leftarrow \max\left\{0, \left\lceil \log_2\left(\left\lfloor \frac{2^{bpp}-1}{2^{k_{n-1}}} \right\rfloor + 1\right)\right\rceil - 1\right\}$$

stays always bounded. A table driven approach is less practical for the original algorithm in the JPEG-LS standard [3] because the required table would be much larger and hence would impact e.g. the cache locality of the software or memory efficiency of the hardware.

3.3. Two-dimensional Golomb parameter adaptation

Rule (2) is a one-dimensional (1D) adaptation based on the *left* neighbor. A similar strategy is to adapt the Golomb parameter from the *top* neighbor. Since natural images are locally stationary in both dimensions, adapting only from the top neighbor and adapting only from the left neighbor will produce nearly the same compression performance. However, by combining both adaptations as

$$k_n = \left\lceil \left(\frac{k_a + k_b}{2}\right)\right\rceil , \qquad (3)$$

where k_a and k_b are the adapted Golomb parameter values given by rule (2) of the left and top neighbor respectively, an improved coding efficiency of about 5% over 1D adaptation has been obtained. We refer to the adaptation rule (3) as 2D Golomb parameter adaptation.

Apparently, the 2D adaptation strategy requires buffering the adapted Golomb parameters for the past $\boldsymbol{w}$ pixels, where $\boldsymbol{w}$ is the image width. Hence the memory requirement is bounded by the image dimensions. For input $bpp = 8$, a range of $[0, 7]$ is sufficient for the Golomb parameter. Thus 3 additional bits are required for each buffered value. Based on the discussion in Section 2, the required memory of the FLIC encoder with 2D adaptation (FLIC-2D) is smaller than that of the JPEG-LS encoder if $\boldsymbol{w} < 4136$.

3.4. Potential parallelization schemes

Apart from the low computational complexity, the FLIC algorithm has advantages for parallelization such as localized data dependency, fast adaptation and bounded memory requirement. Localized data dependency can be seen from rules (2) and (3), i.e. the dependency now exists strictly between *neighboring* pixels. In FLIC, because the coding parameter adapts relatively fast to the source statistics, adaptation can restart at the beginning of each scan line. This allows a line-based parallelization scheme for an FLIC-1D encoder as illustrated in Fig. 4 (a). Assuming 8-bit images and a parallelization degree of $\boldsymbol{n}$, the memory required for Golomb parameter adaptation is $3\boldsymbol{n}$ bits. For an FLIC-2D encoder, the local data dependency can be met by a "skewed" line-based parallelization scheme as shown in Fig. 4 (b). Such a scheme was first proposed for a parallel variant of JPEG-LS in [12], where a shared context set is employed. The original delay-and-merge strategy used to manage conflicting context updates from different parallel encoding units is computationally expensive. By contrast, the parallel FLIC-2D encoder can easily perform updates to

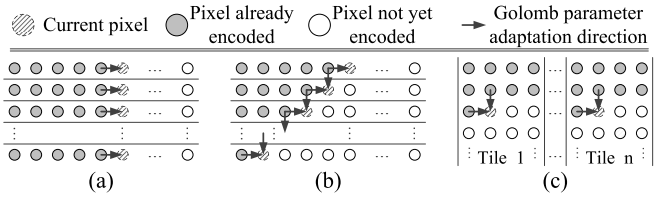

Fig. 4. Parallelization schemes. (a) Line based parallelization for FLIC-1D. (b) Line based parallelization for FLIC-2D. (c) Tiling based parallelization for FLIC-2D.

the $\boldsymbol{n}$ Golomb parameter variables at the $\boldsymbol{n}$ "current" pixel locations simultaneously, with a fixed memory requirement for 2D Golomb parameter adaptation of $3\boldsymbol{w}$ bits. FLIC's bounded memory requirement is also beneficial when used in a tiling based parallelization scheme as illustrated in Fig. 4 (c). Different tiles are encoded independently. If JPEG-LS is used as the tile encoder, the required memory for context modeling will grow linearly with $\boldsymbol{n}$, the number of parallel tile encoders. If FLIC-2D is used, however, the required memory for Golomb parameter adaptation is always $3\boldsymbol{w}$ bits.

4. DE-EMPHASIS OF LIMITED-LENGTH GOLOMB CODE

To limit the buffer space required for local expansion and to support accelerated code word readout with dedicated hardware, the maximum length of the Golomb code should be limited. A simple limitation was introduced in [2] and became part of the official JPEG-LS standard. If the quotient part of the Golomb code reaches the predefined threshold $Q_{\max}$, a so called "escape sequence" is encoded: unary representation of $Q_{\max}$ followed by a direct encoding of $\hat{e}-1$.

The same method is also applicable for the proposed FLIC algorithm. However, since the decoder no longer needs to perform context update based on the residual signal e, it is more beneficial to encode the original value of the pixel x (instead of $\hat{e}$) in the escape sequence. Doing so will save the decoding steps for prediction, inverse residual mapping and calculation of the original pixel value. The overall escape sequence length is hence $Q_{\max}+1+bpp$, where bpp is the input sample precision. For example with $bpp=8$, to limit the code word length to 25 bits, a threshold of $Q_{\max}=16$ would be chosen.

However, compared to an ideal Laplacian distribution of e, an overproportional amount of escape sequences are often generated in practical applications, and thus it is reasonable to refine the strategy to encode these escape sequences. All symbols with a quotient $\lfloor \hat{e}/2^k \rfloor \in [Q_{\max}/2, Q_{\max})$ are encoded with one additional bit in the unary part, hence enlarging their codeword by one bit, and $Q_{\max}/2$ bits in the unary codeword part are, instead, used as an escape sequence if $\lfloor \hat{e}/2^k \rfloor \geq Q_{\max}$. One can derive that this step increases the average codelength of an ideal Laplacian distribution $p(\hat{e}) = (1-\rho)\rho^{\hat{e}}$ by an amount of

$$\Delta\mathbb{E}(L) = \theta^{q/2}\left(1-\theta^{q/2}\right) - \frac{q}{2}\theta^{q}, \text{ where } q = Q_{\max},\ \theta = \rho^{2^k}. \tag{4}$$

Here the first term is due to the additional bit required to encode the symbols between $[Q_{\max}/2, Q_{\max})$, the second term is the reduced length because of the shorter escape sequence. Depending on ρ and q, this expression may be positive or negative; the loss is minor for small ρ, but becomes negative, i.e. we get a coding gain, for $\rho \to 1$. From Fig. 3, one can estimate $\rho \approx 0.9$ and, for $k=2$, $\theta \approx 0.6$ which would be in the region where the coding loss is positive.

However, our experiments showed that the de-emphasis lowers the overall length and improves the overall performance by about 1%. This is no surprise considering the shape of the distribution, namely that the tails of the distribution are overly heavy and thus the source does not quite fit to the Laplacian model implied to derive eqn. (4), and the tail distribution is outweighing the loss.

Table 1. Throughput comparison [Mega pixel per second, Mpps] between FLIC and JPEG-LS.

	JPEG-LS	FLIC-2D	Speedup factor
Natural images	17.4	71.3	4.08
Other images	25.2	81.8	3.24
Average	20.5	75.8	3.71

5. EXPERIMENTAL RESULTS

The proposed FLIC encoder and decoder have been implemented in C++ and their correctness have been verified by comparing the decoded image with the original image. Test environment for the throughput and compression performance results is a computer with 2.67 GHz Core i7 920 CPU and 6 GB DDR3 RAM running a 64-bit operating system. Twenty images from the ITU-T test image set have been used, including thirteen natural images and seven non-natural images (compound, medical, etc.). For a more fair comparison with JPEG-LS, an ad-hoc run mode is implemented in the software. All images are in 8-bit grayscale.

For each test image we neglected I/O time as well as transient (warm-up) phase and the average decoder throughput is measured based on ten stable runs. Sequential decoding without any parallelization have been considered. Table 1 shows the throughput of the FLIC-2D algorithm compared with that of JPEG-LS, measured separately for the two categories of images in Table 2. It can be seen that FLIC achieves average speedup factors of 4.08 and 3.24 over JPEG-LS for natural images and synthetic images respectively. The overall average speedup factor is 3.71 for all test images. Table 2 shows the lossless compression performance of JPEG-LS (JLS), FELICS, JPEG 2000 (J2K), JPEG XR (JXR), FLIC-1D (1D adaptation) and FLIC-2D (2D adaptation). We make the following observations: for natural images, FLIC-1D performs almost the same as JPEG XR, and is slightly outperformed by FELICS. FLIC-2D performs only second to JPEG LS, and is closely followed by JPEG 2000. For low entropy non-natural images, because of the run mode, both FLIC-1D and FLIC-2D perform better than the other algorithms except JPEG-LS. Overall, it can be seen that with 1-dimensional Golomb parameter adaptation, the proposed FLIC has a compression performance penalty of 8.1% compared with JPEG-LS. As shown in the last column of Table 2, with 2-dimensional Golomb parameter adaptation, the performance gap between FLIC and JPEG-LS is reduced to less than 4%. Moreover,

Table 2. Lossless compression performance [Bits per pixel, Bpp] of various lossless image codecs.

	Image	JLS	FELICS [16]	J2K [17]	JXR [18]	FLIC -1D	FLIC -2D
n a t u r a l	AERIAL2	5.29	5.50	5.44	5.48	5.74	5.50
	BAND1	3.56	3.87	3.74	4.07	3.84	3.71
	BIKE	4.36	4.62	4.53	4.82	4.77	4.53
	BIKE3	4.31	4.64	4.65	5.02	4.73	4.48
	CAFE	5.09	5.52	5.35	5.62	5.42	5.17
	CATS	2.57	3.28	2.53	3.12	2.76	2.65
	FINGER	5.66	6.11	5.66	5.66	6.02	5.82
	GOLD	4.48	4.75	4.60	4.74	4.86	4.62
	HOTEL	4.38	4.71	4.59	4.79	4.80	4.59
	TOOLS	5.31	5.65	5.45	5.67	5.63	5.37
	TXTUR1	6.45	6.32	6.83	6.85	6.90	6.65
	TXTUR2	5.36	5.54	5.63	5.71	5.82	5.51
	WOMAN	4.45	4.79	4.51	4.70	4.82	4.59
	Average	4.71	5.02	4.89	5.10	5.09	4.86
o t h e r s	CHART	2.84	3.52	3.08	3.81	3.07	2.96
	CMPND1	1.24	2.37	2.13	3.57	1.30	1.26
	CMPND2	1.36	2.43	2.15	3.44	1.47	1.41
	CT	3.11	3.83	3.30	3.92	3.74	3.47
	PC	1.65	2.86	3.63	6.29	1.72	1.51
	TARGET	2.19	4.21	2.13	3.58	2.29	2.31
	US	2.63	3.28	3.04	4.00	2.77	2.65
	Average	2.14	3.21	2.78	4.09	2.34	2.23
Total average		3.81	4.39	4.15	4.74	4.12	3.94
Gap [%]		0.0	15.1	8.8	24.4	8.1	3.3

FLIC-2D outperforms FELICS, JPEG 2000 and JPEG XR in lossless compression efficiency by about 10%, 5% and 20% respectively.

6. CONCLUSION

A fast and lossless image compression (FLIC) algorithm based on JPEG-LS has been proposed. By eliminating the context model from JPEG-LS, greatly reduced computational complexity has been achieved. At a very low cost of compression efficiency, FLIC not only can reduce the computational complexity and memory space requirements to a fraction of the currently available state-of-the-art algorithms, but also eliminates context-driven data dependencies throughout the image, alleviating the constraints on extensive parallelization. Software implementation of the FLIC codec has shown 3-4 times higher pixel throughput compared with JPEG-LS. Lossless compression performance results have shown that using the FLIC encoder with 1-dimensional Golomb parameter adaptation, a highly memory-efficient and fast adaptive encoding can be achieved with about 8% performance penalty compared with JPEG-LS, while with 2-dimensional Golomb parameter adaptation and a moderately increased memory requirement, the FLIC encoder shows a highly competitive compression performance, with a gap of less than 4% compared with JPEG-LS and clear gains over the other state-of-the-art standards including JPEG 2000 and JPEG XR.

7. REFERENCES

[1] X. Wu and N. Memon, "Context-based, adaptive, lossless image coding," *Communications, IEEE Transactions on*, vol. 45, no. 4, pp. 437–444, Apr. 1997.

[2] M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *Image Processing, IEEE Transactions on*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[3] ISO/IEC 14495-1 | ITU-T T.87, *Information technology – Lossless and near-lossless compression of continuous-tone still images: Baseline*, Dec. 1999.

[4] X. Li and M. Orchard, "Edge-directed prediction for lossless compression of natural images," *Image Processing, IEEE Transactions on*, vol. 10, no. 6, pp. 813–817, Jun. 2001.

[5] I. Matsuda, H. Mori, and S. Itoh, "Lossless coding of still images using minimum-rate predictors," in *Image Processing, 2000. Proceedings. 2000 International Conference on*, vol. 1, 2000, pp. 132–135.

[6] S. Takamura, M. Matsumura, and Y. Yashima, "A study on an evolutionary pixel predictor and its properties," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, Nov. 2009, pp. 1921–1924.

[7] X. Wu, G. Zhai, X. Yang, and W. Zhang, "Adaptive sequential prediction of multidimensional signals with applications to lossless image coding," *Image Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 36–42, Jan. 2011.

[8] P. Howard and J. Vitter, "Fast and efficient lossless image compression," in *Data Compression Conference, 1993. DCC '93.*, Snowbird, Utah, Mar. 1993, pp. 351–360.

[9] R. Starosolski, "Simple fast and adaptive lossless image compression algorithm," *Softw. Pract. Exper.*, vol. 37, pp. 65–91, Jan. 2007.

[10] Z. Wang, A. Michael, S. Wahl, P. Werner, and S. Simon, "A memory efficient parallel lossless image compression engine for high performance embedded systems," in *Image and Signal Processing and Analysis (ISPA), 2011 7th International Symposium on*, Sept. 2011, pp. 390–395.

[11] M. Ferretti and M. Boffadossi, "A parallel pipelined implementation of LOCO-I for JPEG-LS," in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 1, Aug. 2004, pp. 769–772.

[12] S. Wahl, Z. Wang, C. Qiu, M. Wroblewski, L. Rockstroh, and S. Simon, "Memory-efficient parallelization of JPEG-LS with relaxed context update," in *Picture Coding Symposium (PCS), 2010*, Dec. 2010, pp. 142–145.

[13] S. Wahl, H. Tantawy, Z. Wang, P. Werner, and S. Simon, "Exploitation of context classification for parallel pixel coding in JPEG-LS," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*, Sep. 2011, pp. 2001–2004.

[14] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Springer, 2002.

[15] H. Malvar, "Adaptive run-length/Golomb-Rice encoding of quantized generalized gaussian sources with unknown statistics," in *Data Compression Conference, 2006. DCC 2006. Proceedings*, Mar. 2006, pp. 23–32.

[16] A. Moffat *et al.* (1999, Aug.) MG software (Version 1.2.1). [Online]. Available: http://ww2.cs.mu.oz.au/mg/

[17] ISO/IEC 15444-5 | ITU-T T.804, "Information technology – JPEG 2000 image coding system: Reference software," Aug. 2002.

[18] ISO/IEC 29199-5 | ITU-T T.835, "Information technology – JPEG XR image coding system – Reference software," ISO/IEC JTC1/SC29/WG1 document N5973.