# Speeding up Lossless Image Compression: Experimental Results on a Parallel Machine

Luigi Cinque[1], Sergio De Agostino[1], and Luca Lombardi[2]

[1] Computer Science Department
Sapienza University
Via Salaria 113, 00198 Roma, Italy
{cinque, deagostino}@di.uniroma1.it
[2] Computer Science Department
University of Pavia
Via Ferrara 1, 27100 Pavia, Italy
luca.lombardi@unipv.it

**Abstract.** Arithmetic encoders enable the best compressors both for bi-level images (JBIG) and for grey scale and color images (CALIC), but they are often ruled out because too complex. The compression gap between simpler techniques and state of the art compressors can be significant. Storer extended dictionary text compression to bi-level images to avoid arithmetic encoders (BLOCK MATCHING), achieving 70 percent of the compression of JBIG1 on the CCITT bi-level image test set. We were able to partition an image into up to a hundred areas and to apply the BLOCK MATCHING heuristic independently to each area with no loss of compression effectiveness. On the other hand, we presented in [5] a simple lossless compression heuristic for gray scale and color images (PALIC), which provides a highly parallelizable compressor and decompressor. In fact, it can be applied independently to each block of 8x8 pixels, achieving 80 percent of the compression obtained with LOCO-I (JPEG-LS), the current lossless standard in low-complexity applications. We experimented the BLOCK MATCHING and PALIC heuristics with up to 32 processors of a 256 Intel Xeon 3.06 GHz processors machine in Italy (avogadro.cilea.it) on a test set of large topographic bi-level images and color images in RGB format. We obtained the expected speed-up of the compression and decompression times, achieving parallel running times about twenty-five times faster than the sequential ones.

**Keywords:** lossless image compression, sliding dictionary, differential coding, parallelization

## 1 Introduction

Lossless image compression is often realized by extending string compression methods to two-dimensional data. Standard lossless image compression methods extend model driven text compression [1], consisting of two distinct and independent phases: *modeling* [16] and *coding* [15]. In the coding phase, arithmetic encoders enable the best model driven compressors both for bi-level images (JBIG [10]) and for grey scale and color images (CALIC [20]), but they are often ruled out because too complex. The compression gap between simpler techniques and state of the art compressors can be significant.

Storer [18] extended dictionary text compression [17] to bi-level images to avoid arithmetic encoders by means of a square greedy matching technique (BLOCK MATCHING), achieving 70 percent of the compression of JBIG1 on the CCITT bi-level image test set. The technique is a two-dimensional extension of LZ1 compression [12] and is suitable for high speed applications by means of a simple hashing scheme.

Rectangle matching improves the compression performance, but it is slower since it requires $O(M \log M)$ time for a single match, where $M$ is the size of the match [19]. Therefore, the sequential time to compress an image of size $n$ by rectangle matching is $\Omega(n \log M)$. However, rectangle matching is more suitable for polylogarithmic time work-optimal parallel implementations on the PRAM EREW [3], [6] and the mesh of trees [2], [7]. Polylogarithmic time parallel implementations were also presented for decompression on both the PRAM EREW and the mesh of trees in [2].

Parallel models have two types of complexity, the interprocessor communication and the input-output mechanism. While the input/output issue is inherent to any sublinear algorithm and has standard solutions, the communication cost of the computational phase after the distribution of the data among the processors and before the output of the final result is obviously algorithm-dependent. So, we need to limit the interprocessor communication and involve more local computation. The simplest model for this phase is, of course, a simple array of processors with no interconnections and, therefore, no communication cost. The parallel implementations mentioned above require more sophisticated architectures than a simple array of processors to be executed on a distributed memory system.

Dealing with square matches, we were able to partition an image into up to a hundred areas and to apply the BLOCK MATCHING heuristic independently to each area with no loss of compression effectiveness. With rectangles we cannot obtain the same performance since the width and the length are shortened while the corresponding pointers are more space consuming than with squares. So we would rather implement the square BLOCK MATCHING heuristic on an array of size up to a hundred processors.

The extension of Storer's method to grey scale and color images was left as an open problem, but it seems not feasible since the high cardinality of the alphabet causes an unpractical exponential blow-up of the hash table used in the implementation.

As far as the model driven method for grey scale and color image compression is concerned, the modeling phase consists of three components: the determination of the context of the next pixel, the prediction of the next pixel and a probabilistic model for the *prediction residual*, which is the value difference between the actual pixel and the predicted one. In the coding phase, the prediction residuals are encoded. A first step toward a good low complexity compression scheme was FELICS (Fast Efficient Lossless Image Compression System) [11], which involves Golomb-Rice codes [9], [14] rather than the arithmetic ones. With the same complexity level for compression (but with a 10 percent slower decompressor) LOCO-I (Low Complexity Lossless Compression for Images) [13] attains significantly better compression than FELICS, only a few percentage points of CALIC (Context-Based Adaptive Lossless Image Compression). As explained in [5], also polylogarithmic time parallel implementations of FELICS and LOCO-I would require more sophisticated architectures than a simple array of processors.

The use of prediction residuals for grey scale and color image compression relies on the fact that most of the times there are minimal variations of color in the neighborood of one pixel. Therefore, differently than for bi-level images we should be able to implement an extremely local procedure which is able to achieve a satisfying degree of compression by working independently on different very small blocks. In [5], we showed such procedure. We presented the heuristic for grey scale images, but it could also be applied to color images by working on the different components [4]. We call such procedure PALIC (Parallelizable Lossless Image Compression). In fact, the main advantage of PALIC is that it provides a highly parallelizable compressor and decompressor since it can be applied independently to each block of 8x8 pixels, achieving 80 percent of the compression obtained with LOCO-I (JPEG-LS), the current lossless standard in low-complexity applications.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 255 | 255 | 255 | 254 | 254 | 110 | 110 | 110 |
| 255 | 255 | 255 | 254 | 254 | 110 | 110 | 110 |
| 255 | 255 | 255 | 254 | 254 | 110 | 110 | 110 |
| 255 | 255 | 255 | 254 | 254 | 110 | 110 | 110 |
| 255 | 255 | 254 | 128 | 127 | 128 | 129 | 130 |
| 255 | 253 | 253 | 128 | 128 | 129 | 130 | 131 |
| 254 | 253 | 252 | 129 | 129 | 130 | 131 | 132 |
| 253 | 252 | 251 | 130 | 130 | 130 | 254 | 255 |

**Figure 1.** An 8x8 pixel block of a grey scale image.

The compressed form of each block employs a header and a fixed length code. Two different techniques might be applied to compress the block. One is the simple idea of reducing the alphabet size by looking at the values occurring in the block. The other one is to encode the difference between the pixel value and the smallest one in the block. Observe that this second technique can be interpreted in terms of the model driven method, where the block is the context, the smallest value is the prediction and the fixed length code encodes the prediction residual. More precisely, since the code is fixed length the method can be seen as a two-dimensional extension of differential coding [8]. Differential coding, often applied to multimedia data compression, transmits the difference between a given signal sample and another sample.

In this paper, we experimented the square BLOCK MATCHING and PALIC heuristics with up to 32 processors of a 256 Intel Xeon 3.06 GHz processors machine in Italy (`avogadro.cilea.it`) on a test set of large topographic bi-level images and color images in RGB format. We obtained the expected speed-up of the compression and decompression times, achieving parallel running times about twenty-five times faster than the sequential ones.

In section 2, we explain the heuristics. In section 3 we provide the experimental results on the parallel machine. Conclusions are given in section 4.

## 2 BLOCK MATCHING and PALIC

Among the different ways of reading an image, we assume the square BLOCK MATCHING heuristic scans an $m$ x $m'$ image row by row (*raster* scan). A 64K table with one position for each possible 4x4 subarray is the only data structure used. All-zero and all-one rectangles are handled differently. The encoding scheme is to precede each item with a flag field indicating whether there is a monochromatic square, a match or raw data. When there is a match, the 4x4 subarray in the current position is hashed to yield a pointer to a copy. This pointer is used for the current square greedy match and then replaced in the hash table by a pointer to the current position. The procedure for computing the largest square match with left upper corners in positions $(i, j)$ and $(k, h)$ takes $\mathrm{O}(M)$ time, where $M$ is the size of the match. Obviously, this procedure can be used for computing the largest monochromatic square in a given position $(i, j)$ as well. If the 4 x 4 subarray in position $(i, j)$ is monochromatic, then we compute the largest monochromatic square in that position. Otherwise, we compute the largest square match in the position provided by the hash table and update the table with the current position. If the subarray is not hashed to a pointer, then it is left uncompressed and added to the hash table with its current position. The positions covered by matches are skipped in the linear scan of the image. Therefore, the sequential time to compress an image of size $n$ by square matching is $\mathrm{O}(n)$. We want to point out that besides the proper matches we use to call a match every rectangle of the parsing of the image produced by the heuristic. We also call a pointer the encoding of every match. As mentioned above, the encoding scheme for the pointers uses a flag field indicating whether there is a monochromatic rectangle (0 for the white ones and 10 for the black ones), a proper match (110) or raw data (111).

As mentioned in the introduction, we were able to partition an image into up to a hundred areas and to apply the BLOCK MATCHING heuristic independently to each area with no loss of compression effectiveness on both the CCITT bi-level image test set and the bi-level version of the set of five 4096 x 4096 pixels images in Figures 2–6.

Moreover, in order to implement decompression on an array of processors, we want to indicate the end of the encoding of a specific area. Therefore, we change the encoding scheme by associating the flag field 1110 to the raw match so that we can indicate with 1111 the end of the sequence of pointers corresponding to a given area.

We explain now how to apply the PALIC heuristic independently to blocks of 8x8 pixels of a grey scale image. We still assume to read the image with a raster scan on each block. The heuristic applies at most three different ways of compressing the block and chooses the best one. The first one is the following.

The smallest pixel value is computed on the block. The header consists of three fields of 1 bit, 3 bits and 8 bits, respectively. The first bit is set to 1 to indicate that we compress a block of 64 pixels. This is because one of the three ways will partition the block in four sub-blocks of 16 pixels and compress each of these smaller areas. The 3-bits field stores the minimum number of bits required to encode in binary the distance between the smallest pixel value and every other pixel value in the block. The 8-bits field stores the smallest pixel value. If the number of bits required to encode the distance, say $k$, is at most 5, then a code of fixed length $k$ is used to encode the 64 pixels, by giving the difference between the pixel value and the smallest one in the block. To speed up the procedure, if $k$ is less or equal to 2 the other ways are not tried because we reach a satisfying compression ratio on the block. The second and third ways are the following.

The second way is to detect all the different pixel values in the 8x8 block and to create a reduced alphabet. Then, to encode each pixel in the block using a fixed length

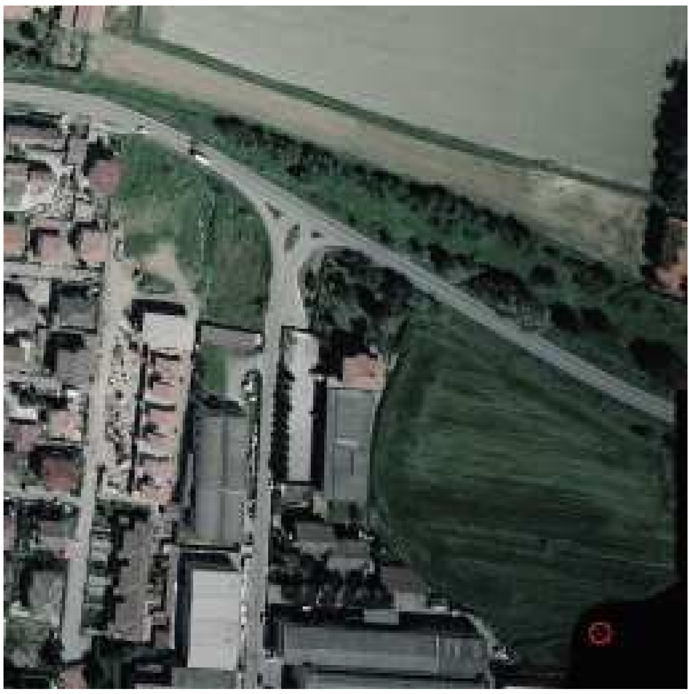

**Figure 2.** Image 1.

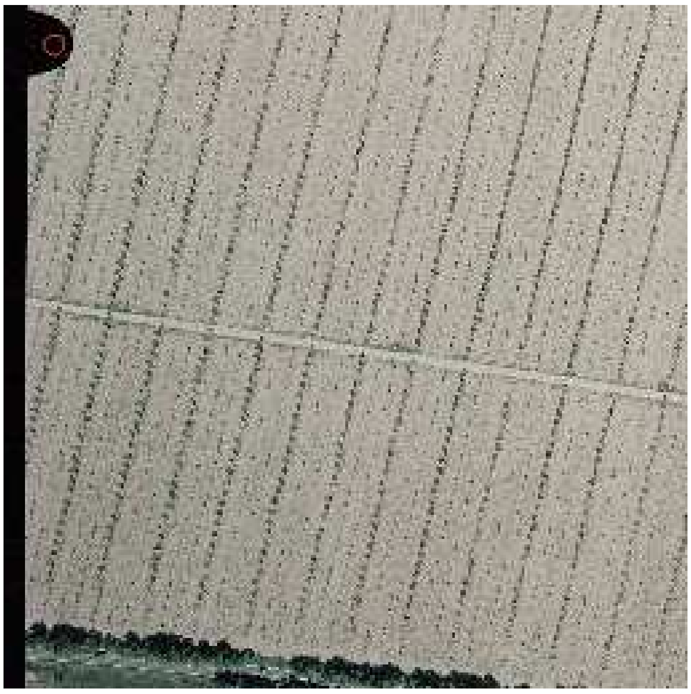

**Figure 3.** Image 2.

code for this alphabet. The employment of this technique is declared by setting the 1-bit field to 1 and the 3-bits field to 110. Then, an additional three bits field stores the reduced alphabet size $d$ with an adjusted binary code in the range $2 \leq d \leq 9$.

**Figure 4.** Image 3.

**Figure 5.** Image 4.

The last componenent of the header is the alphabet itself, a concatenation of $d$ bytes. Then, a code of fixed length $\lceil \log d \rceil$ bits is used to encode the 64 pixels.

The third way compresses the four 4x4 pixel sub-blocks. The 1-bit field is set to 0. Four fields follow the flag bit, one for each 4x4 block. The two previous techniques

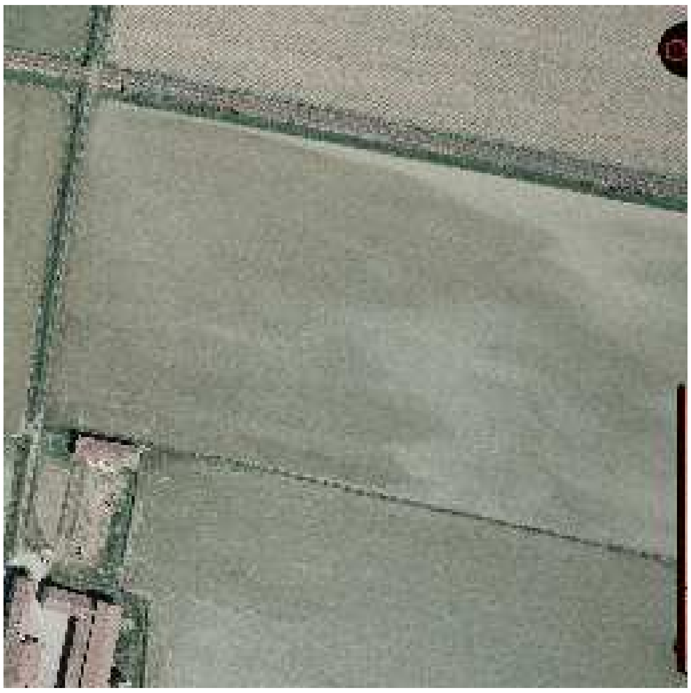

**Figure 6.** Image 5.

are applied to the blocks and the best one is chosen. If the first technique is applied to a block, the corresponding field stores values from 0 to 7 rather than from 0 to 5 as for the 8x8 block. If such value is in between 0 and 6, the field stores three bits. Otherwise, the three bits (111) are followed by three more. This is because 111 is used to denote the application of the second way to the block as well, which is less frequent to happen. In this case, the reduced alphabet size stored in these three additional bits has range from 2 to 7, it is encoded with an adjusted binary code from 000 to 101 and the alphabet follows. 110 denotes the application of the first technique with distances expressed in seven bits and 111 denotes that the block is not compressed. After the four fields, the compressed forms of the blocks follow, which are similar to the ones described for the 8x8 block. When the 8x8 block is not compressed, 111 follows the flag bit set to 1.

We now show how PALIC works on the example of Figure 1.

Since the difference between 110, the smallest pixel value, and 255 requires a code with fixed length 8 and the number of different values in the 8x8 block is 12, the way employed to compress the block is to work separately on the 4x4 sub-blocks. Each block will be encoded with a raster scan (row by row). The upper left block has 254 as its smallest pixel value and 255 is the only other value. Therefore, after setting the 1-bit field to zero the corresponding field is set to 001. The compressed form after the header is 1110111011101110. The reduced alphabet technique is more expensive since the raw pixel values must be given. On the other hand, the upper right block needs the reduced alphabet technique. In fact, one byte is required to express the difference between 110 and 254. Therefore, the corresponding field is set to 111000, which indicates that the reduced alphabet size is 2, and the sequence of two bytes 0110111011111110 follows. The compressed form after the header is 1000100010001000. The lower left block has 8 different values so we do not use the reduced alphabet technique since the alphabet size should be between 2 and 7. The smallest pixel value in the block is 128 and the largest difference is 127 with the pixel value 255. Since a code of fixed length 7 is required, the corresponding field is 111110. The compressed form after the header is (we introduce a space between pixel encodings in the text to make it more readable): 1111111 1111111 1111110 0000000 1111111 1111101 1111101 0000000 1111110 1111101 1111100 0000001 1111101 1111100 1111011 0000010. Observe that the compression of the block would have been the same if we had allowed the reduced alphabet size to grow up to 8. However, experimentally we found more advantageous to exclude this case in favor of the other technique. Our heuristic does not compress the lower right block since it has 8 different values and the difference between pixel values 127 and 255 requires 8 bits. Therefore, the corresponding field is 111111 and the uncompressed block follows.

We experimented PALIC on the kodak image test set, which is an extension of the standard jpeg image test set and reached 70 to 85 percent of LOCO-I compression ratio (78 percent in average). We also experimented it on the set of five 4096 x 4096 pixels grey scale topographic images in Figure 2-6 and the compression effectiveness was about 80 percent of LOCO-I compression effectiveness as for the kodak image set. The heuristic can be trivially extended to RGB color images by working sequentially on each of the three components of the block and the same compression effectiviness results in comparison with LOCO-I were obtained for the RGB version of the five images in Figures 2–6.

## 3 Experimental Results on a Parallel Machine

We show in Figures 7–8 the compression and decompression times of PALIC on the RGB version of the five images in Figures 2–6 doubling up the number of processors of the `avogadro.cilea.it` machine from 1 to 32. We executed the compression and decompression on each image several times. The variances of both the compression and decompression times were small and we report the greatest running times, conservatively. As it can be seen from the values on the tables, also the variance over the test set is quite small. The decompression times are faster than the compression ones and in both cases we obtain the expected speed-up, achieving parallel running times about twenty-five times faster than the sequential ones.

| Image | 1 proc. | 2 proc. | 4 proc. | 8 proc. | 16 proc. | 32 proc. |
|---:|---:|---:|---:|---:|---:|---:|
| 1 | 227 | 117 | 57 | 34 | 17 | 9 |
| 2 | 243 | 120 | 69 | 33 | 16 | 10 |
| 3 | 235 | 118 | 72 | 35 | 17 | 9 |
| 4 | 236 | 131 | 71 | 34 | 16 | 9 |
| 5 | 232 | 113 | 67 | 30 | 16 | 11 |
| Avg. | 234.6 | 119.8 | 67.2 | 33.2 | 16.4 | 9.6 |

**Figure 7.** PALIC compression times (cs.).

The images of Figures 2–4 have the greatest parallel decompression times with 32 processors. On the other hand, the image of Figure 3 has the greatest sequential compression and decompression times. The smallest compression time with 32 processors

| Image | 1 proc. | 2 proc. | 4 proc. | 8 proc. | 16 proc. | 32 proc. |
|---|---|---|---|---|---|---|
| 1 | 128 | 65 | 32 | 18 | 11 | 6 |
| 2 | 133 | 66 | 35 | 21 | 10 | 6 |
| 3 | 130 | 66 | 44 | 21 | 13 | 6 |
| 4 | 129 | 91 | 36 | 20 | 10 | 5 |
| 5 | 123 | 95 | 46 | 17 | 10 | 5 |
| Avg. | 128.6 | 76.6 | 38.6 | 19.4 | 10.8 | 5.6 |

**Figure 8.** PALIC decompression times (cs.).

is given by the image of Figure 4, together with the images of Figure 2 and Figure 5. Instead, the smallest decompression time with 32 processors is given by the images of Figures 5–6. The image of Figure 6 also has the smallest sequential decompression time and the greatest compression time with 32 processors.

| Image | 1 proc. | 2 proc. | 4 proc. | 8 proc. | 16 proc. | 32 proc. |
|---|---|---|---|---|---|---|
| 1 | 76 | 39 | 19 | 11 | 6 | 3 |
| 2 | 81 | 40 | 23 | 11 | 5 | 3 |
| 3 | 78 | 39 | 24 | 12 | 6 | 3 |
| 4 | 79 | 44 | 24 | 11 | 5 | 3 |
| 5 | 77 | 38 | 22 | 10 | 5 | 4 |
| Avg. | 78.2 | 40 | 22.4 | 11 | 5.4 | 3.2 |

**Figure 9.** BLOCK MATCHING compression times (cs.).

| Image | 1 proc. | 2 proc. | 4 proc. | 8 proc. | 16 proc. | 32 proc. |
|---|---|---|---|---|---|---|
| 1 | 43 | 22 | 11 | 6 | 4 | 2 |
| 2 | 44 | 22 | 12 | 7 | 3 | 2 |
| 3 | 43 | 22 | 15 | 7 | 4 | 2 |
| 4 | 43 | 30 | 12 | 7 | 3 | 2 |
| 5 | 41 | 32 | 15 | 6 | 3 | 2 |
| Avg. | 42.8 | 25.6 | 13 | 6.6 | 3.4 | 2 |

**Figure 10.** BLOCK MATCHING decompression times (cs.).

We obtained similar results for the BLOCK MATCHING heuristic. In Figures 9–10 we show the compression and decompression times of the square BLOCK MATCHING heuristic on the bi-level version of the five images in Figures 2–6, doubling up the number of processors of the `avogadro.cilea.it` machine from 1 to 32. This means that when $2^k$ processors are involved, for $1 \leq k \leq 5$, the image is partitioned into $2^k$ areas and the compression heuristic is applied in parallel to each area, independently.

As far as decompression is concerned, each of the $2^k$ processors decodes the pointers corresponding to a given area.

## 4 Conclusions

In this paper, we showed experimental results on the coding and decoding times of two lossless image compression methods on a real parallel machine. By doubling up the number of processors from 1 to 32, we obtained the expected speed-up on a test set of large topographic bi-level images and color images in RGB format, achieving parallel running times about twenty-five times faster than the sequential ones. The feasibility of a highly parallelizable compression method for grey scale and color images relied on the fact that most of the times there are minimal variations of color in the neighborood of one pixel. Therefore, we were able to implement an extremely local procedure which achieves a satisfying degree of compression by working independently on different very small blocks. On the other hand, we designed a non-massive approach to bi-level image compression which could be implemented on an array of processors of reasonable size, achieving a satisfying degree of compression. Such goal was realized by making each processor work on a single large block rather than on many very small blocks as when the non-massive way is applied to grey scale or color images.

## References

1. T. C. Bell, J. G. Cleary, and I. H. Witten: *Text Compression*, Prentice Hall, 1980.
2. L. Cinque and S. DeAgostino: *A parallel decoder for lossless image compression by block matching*, in Proceedings IEEE Data Compression Conference, 2007, pp. 183–192.
3. L. Cinque, S. DeAgostino, and F. Liberati: *A work-optimal parallel implementation of lossless image compression by block matching.* Nordic Journal of Computing, 2003, pp. 183–192.
4. L. Cinque, S. DeAgostino, and F. Liberati: *A simple lossless compression heuristic for rgb images*, in Proceedings IEEE Data Compression Conference, Poster Session, 2004.
5. L. Cinque, S. DeAgostino, F. Liberati, and B. Westgeest: *A simple lossless compression heuristic for grey scale images.* International Journal of Foundations of Computer Science, 16 2005, pp. 1111–1119.
6. S. DeAgostino: *A work-optimal parallel implementation of lossless image compression by block matching*, in Proceedings Prague Stringology Conference, 2002, pp. 1–8.
7. S. DeAgostino: *Lossless image compression by block matching on a mesh of trees*, in Proceedings IEEE Data Compression Conference, Poster Session, 2006, p. 443.
8. J. D. Gibson: *Adaptive prediction in speech differential encoding system.* Proceedings of the IEEE, 68 1980, pp. 488–525.
9. S. W. Golomb: *Run-length encodings.* IEEE Transactions on Information Theory, 12 1966, pp. 399–401.
10. P. G. Howard, F. Kossentini, B. Martinis, S. Forchammer, W. J. Rucklidge, and F. Ono: *The emerging jbig2 standard.* IEEE Transactions on Circuits and Systems for Video Technology, 8 1998, pp. 838–848.
11. P. G. Howard and J. S. Vitter: *Fast and efficient lossles image compression*, in Proceedings IEEE Data Compression Conference, 1993, pp. 351–360.
12. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23 1977, pp. 337–343.
13. G. S. G. M. J. Weimberger and G. Sapiro: *Loco-i: A low complexity, context based, lossless image compression algorithm*, in Proceedings IEEE Data Compression Conference, 1996, pp. 140–149.
14. R. F. Rice: *Some practical universal noiseless coding technique - part i*, Tech. Rep. JPL-79-22, Jet Propulsion Laboratory, Pasadena, California, USA, 1979.

15. J. Rissanen: *Generalized kraft inequality and arithmetic coding.* IBM Journal on Research and Development, 20 1976, pp. 198–203.
16. J. Rissanen and G. G. Langdon: *Universal modeling and coding.* IEEE Transactions on Information Theory, 27 1981, pp. 12–23.
17. J. A. Storer: *Data Compression: Methods and Theory*, Computer Science Press, 1988.
18. J. A. Storer: *Lossless image compression using generalized lz1-type methods*, in Proceedings IEEE Data Compression Conference, 1996, pp. 290–299.
19. J. A. Storer and H. Helfgott: *Lossless image compression by block matching.* The Computer Journal, 40 1997, pp. 137–145.
20. X. Wu and N. D. Memon: *Context-based, adaptive, lossless image coding.* IEEE Transactions on Communications, 45 1997.