Binary Image Compression via Monochromatic Pattern Substitution: Effectiveness and Scalability

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

[1] Computer Science Department, Sapienza University, 00198 Rome, Italy
[2] Computer Science Department, University of Pavia, 27100 Pavia, Italy

Abstract. We present a method for compressing binary images via monochromatic pattern substitution. Monochromatic rectangles inside the image are compressed by a variable length code. Such method has no relevant loss of compression effectiveness if the image is partitioned into up to a thousand blocks and each block is compressed independently. Therefore, it can be implemented in parallel on both small and large scale arrays of processors with distributed memory and no interconnections. We experimented the procedure with up to 32 processors of a 256 Intel Xeon 3.06 GHz processors machine (`avogadro.cilea.it`) on a test set of large topographic bi-level images. We obtained the expected speed-up of the compression and decompression times, achieving parallel running times about twenty times faster than the sequential ones. In the theoretical context of unbounded parallelism, we show experimentally that interprocessor communication is needed when we scale up the distributed system. It results that compression effectiveness has a bell-shaped behaviour which is again competitive with the sequential performance when the highest degree of parallelism is reached.

Keywords: lossless compression, binary image, distributed algorithm, scalability

1 Introduction

The best lossless image compressors are enabled by arithmetic encoders based on the model driven method [2]. The *model driven method* consists of two distinct and independent phases: *modeling* [17] and *coding* [16]. Arithmetic encoders are the best model driven compressors both for binary images (JBIG) [13] and for grey scale and color images (CALIC)[24], but they are often ruled out because they are too complex.

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. 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. LOCO-I (JPEG-LS) [22] is the current lossless standard in low-complexity applications and involves Golomb-Rice codes [12], [15] rather than the arithmetic ones. A low-complexity application compressing 8x8 blocks of a grey-scale or color image by means of a header and a fixed-length code is PALIC [6], [7] which can be implemented on an arbitrarily large scale system at no communication cost. As explained in [7], parallel implementations of LOCO-I would require more sophisticated architectures than a simple array of processors. PALIC achieves 80 percent of the compression obtained with LOCO-I and is an extremely local procedure which is able to achieve a satisfying degree of compression by working independently on different very small blocks. On the other hand, no local low-complexity binary image compressor has been designed so far. BLOCK MATCHING [18], [19] is the best low-complexity compressor for binary images and extends data compression via textual substitution to two-dimensional data by compressing sub-images rather than substrings [14], [20], achieving 80 percent of the compression obtained with JBIG. However, it does not work locally since it applies a generalized LZ1-type method with an unrestricted window.

In this paper, we present a method for compressing binary images via monochromatic pattern substitution. Monochromatic rectangles inside the image are compressed by a variable length code. Such monochromatic rectangles are detected by means of a *raster* scan (row by row). If the 4 x 4 subarray in position (i, j) of the image is monochromatic, then we compute the largest monochromatic rectangle in that position else we leave it uncompressed. The encoding scheme is to precede each item with a flag field indicating whether there is a monochromatic rectangle or raw data. The procedure for computing the largest monochromatic rectangle with left upper corner in position (i, j) takes $\mathrm{O}(M \log M)$ time, where M is the size of the rectangle. The positions covered by the detected rectangles are skipped in the linear scan of the image and the sequential time to compress an image of size n by rectangle matching is $\Omega(n \log M)$. The analysis of the running time of this algorithm involves a *waste factor*, defined as the average number of matches covering the same pixel. We experimented that the waste factor is less than 2 on realistic image data. Therefore, the heuristic takes $\mathrm{O}(n \log M)$ time. On the other hand, the decoding algorithm is linear. The compression effectiveness of this technique is about the same as the one of the rectangular block matching technique [19], which still requires $\Omega(n \log M)$ time. However, in practice it is about twice faster.

Compression via monochromatic pattern substitution by the variable length code presented here has no relevant loss of effectiveness if the image is partitioned into up to a thousand blocks and each block is compressed independently. Therefore, it can be implemented in parallel on both small and large scale arrays of processors with distributed memory and no interconnections. We experimented the procedure with up to 32 processors of a 256 Intel Xeon 3.06 GHz processors machine (`avogadro.cilea.it`) on a test set of large topographic bi-level images. We obtained the expected speed-up of the compression and decompression times, achieving parallel running times about twenty times faster than the sequential ones. Similar results were obtained in [10], [8], [9] on the same machine for the version of block matching compressing squares. However, although the square block matching technique [18] has a linear sequential time it has slower compression and decompression running times in practice. Moreover, it has a lower effectiveness and it is not scalable.

In the theoretical context of unbounded parallelism, interprocessor communication is needed when we scale up the distributed system. Parallel algorithms are shown in [1], [4], [5], [10], [9] for the rectangular block matching heuristic. In [5], a variable length coding technique similar to the one presented in this paper is described for the sequential implementation of the rectangular block matching method. The technique encodes bounded size two-dimensional patterns and has not been employed in the theoretical parallel implementations. We employed it in the experimental results of this paper on compression effectiveness since the bound to the pattern dimensions is large enough with respect to the size of realistic image data. It resulted that the compression effectiveness has a bell-shaped behaviour which is again competitive with the sequential performance when the highest degree of parallelism is reached.

Previous work on the square block matching heuristic is reported in section 2. Compression via monochromatic pattern substitution is described in section 3. In section 4, we present the parallel implementations and the experimental results on the speed-up. In section 5, we show how parallel implementations can be realized in the theoretical context of unbounded parallelism by means of interprocessor communication. Section 6 presents the experimental results relating scalability to compression effectiveness. Conclusions and future work are given in section 7.

2 Previous Work

As mentioned in the introduction, the square block matching procedure has been so far the only low-complexity lossless binary image compression technique implemented in parallel at no communication cost [10], [8], [9]. Such technique is a two-dimensional extension of Lempel-Ziv compression [14] and simple practical heuristics exist to implement Lempel-Ziv compression by means of hashing techniques [3], [11], [21], [23].

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 1. Compression times of the block matching procedure (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 2. Decompression times of the block matching procedure (cs.)

The hashing technique used for the two-dimensional extension is even simpler [18]. The square matching compression procedure scans an m x m' image by a raster scan. A 64K table with one position for each possible 4x4 subarray is the only data structure used. All-zero and all-one squares 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

```
$c = p_{r,j}$;
$r = i$;
$width = m'$;
$length = 0$;
$side1 = side2 = area = 0$;
**repeat**
    Let $p_{r,j} \cdots p_{r,j+\ell-1}$ be the longest string in $(r,j)$ with color $c$ and $\ell \leq width$;
    $length = length + 1$;
    $width = \ell$;
    $r = r + 1$;
    **if** $(length * width > area)$ {
        $area = length * width$;
        $side1 = length$;
        $side2 = width$;
    }
**until** $area \geq width * (i - k + 1)$ or $p_{r,j} <> c$
```

Figure 3. Computing the largest monochromatic rectangle match in (i, j)

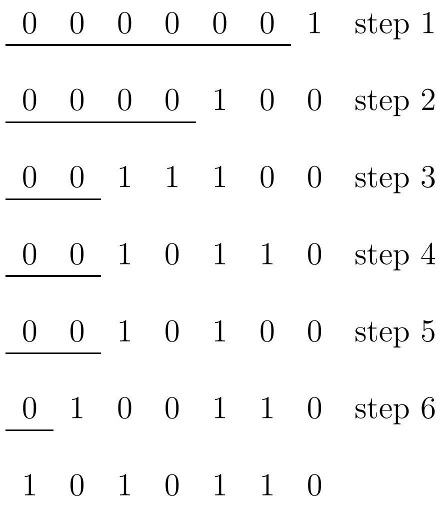

Figure 4. The largest monochromatic match in (0,0) is computed at step 5

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 $O(M)$ time, where M is the size of the match. Obviously, this procedure can be used to compute 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 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

Figure 5. Image 1

Figure 6. Image 2

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. We wish to point out that besides the proper matches we call a match every square of the parsing of the image produced by the heuristic. We also call pointer the encoding of

Figure 7. Image 3

Figure 8. Image 4

a match. Therefore, each pointer starts with a flag field indicating whether there is a monochromatic match (0 for the white ones and 10 for the black ones), a proper match (110) or a raw match (111).

Figure 9. Image 5

We were able to partition an image into up to a hundred areas and to apply the square block matching heuristic independently to each area with no relevant loss of compression effectiveness on both the CCITT bi-level image test set and to the set of five 4096 x 4096 pixels half-tone images of figures 5-9. Moreover, in order to implement decompression on an array of processors, we needed to indicate the end of the encoding of a specific area. So, we changed the encoding scheme by associating the flag field 1110 to the raw match so that 1111 could indicate the end of the sequence of pointers corresponding to a given area. Then, the flag field 1110 is followed by sixteen bits uncompressed, flag fields 0 and 10 by the size of the square side while flag field 110 is also followed by the position of the matching copy. The values following the flag fields are represented by a fixed length code.

In [10], [8], [9] we showed the compression and decompression times of the parallel procedure applied to the five images of figures 5-9, doubling up the number of processors of the avogadro.cilea.it machine from 1 to 32 (figures 1 and 2). In the next section, we present a lossless compression technique for binary images which is implementable at no communication cost on both small and large scale parallel systems, differently from the one described in this section.

3 Compression via Monochromatic Pattern Substitution

The technique scans an image row by row. If the 4 x 4 subarray in position (i, j) of the image is monochromatic, then we compute the largest monochromatic rectangle in that position. We denote with $p_{i,j}$ the pixel in position (i, j). The procedure for finding the largest rectangle with left upper corner (i, j) is described in figure 3. At the first step, the procedure computes the longest possible width for a monochromatic rectangle in (i, j) and stores the color in c. The rectangle 1 x ℓ computed at the first step is the current detected rectangle and the sizes of its sides are stored in $side1$ and $side2$. In order to check whether there is a better match than the current one, the longest sequence of consecutive pixels with color c is computed on the next row starting from column j. Its length is stored in the temporary variable $width$ and the temporary variable $length$ is increased by one. If the rectangle R whose sides have size $width$ and $length$ is greater than the current one, the current one is replaced by R. We iterate this operation on each row until the area of the current rectangle is greater or equal to the area of the longest feasible $width$-wide rectangle, since no further improvement would be possible at that point.

For example, in figure 4 we apply the procedure to find the largest monochromatic rectangle in position $(0,0)$. A monochromatic rectangle of width 6 is detected at step 1. Then, at step 2 a larger rectangle is obtained which is 2 x 4. At step 3 and step 4 the current rectangle is still 2 x 4 since the longest monochromatic sequence on row 3 and 4 comprises two pixels. At step 5, another sequence of two pixels provides a larger rectangle which is 5 x 2. At step 6, the procedure stops since the longest monochromatic sequence is just one pixel and the rectangle can cover at most 7 rows. It follows that the detected rectangle is 5 x 2 since a rectangle of width 1 cannot have a larger area. Such procedure for computing the largest monochromatic rectangle in position (i,j) takes $\mathrm{O}(M \log M)$ time, where M is the rectangle size. In fact, in the worst case a rectangle of size M could be detected on row i, a rectangle of size $M/2$ on row $i+1$, a rectangle of size $M/3$ on row $i+2$ and so on.

If the 4 x 4 subarray in position (i,j) of the image is not monochromatic, we do not expand it. The positions covered by the detected rectangles are skipped in the linear scan of the image. The encoding scheme for such rectangles uses a flag field indicating whether there is a monochromatic match (0 for the white ones and 10 for the black ones) or not (11). If the flag field is 11, it is followed by the sixteen bits of the 4 x 4 subarray (raw data). Otherwise, we bound by twelve the number of bits to encode either the width or the length of the monochromatic rectangle. We use either four or eight or twelve bits to encode one rectangle side. Therefore, nine different kinds of rectangle are defined. A monochromatic rectangle is encoded in the following way:

– the flag field indicating the color;
– three or four bits encoding one of the nine kinds of rectangle;
– bits for the length and the width.

Four bits are used to indicate when twelve bits or eight and twelve bits are needed for the length and the width. This way of encoding rectangles plays a relevant role for the compression performance. In fact, it wastes four bits when twelve bits are required for the sides but saves four to twelve bits when four or eight bits suffice.

4 The Parallel Implementations

The variable length coding technique expained in the previous section has been applied to the CCITT test set of bi-level images and has provided a compression ratio equal to 0.13 in average. The images of the CCITT test set are 1728 x 2376 pixels. If these images are partitioned into 4^k sub-images and the compression heuristic is applied independently to each sub-image, the compression effectiveness remains about the same for $1 \leq k \leq 5$ with a 1 percent loss for $k = 5$. Raw data are associated with the flag field 110, so that we can indicate with 111 the end of the encoding of a sub-image. For $k = 6$, the compression ratio is still just a few percentage points of the sequential one. This is because the sub-image is 27 x 37 pixels and it still captures the monochromatic rectangles which belong to the class encoded with four bits for each dimension. These rectangles are the most frequent and give the main contribution to the compression effectiveness. On the other hand, the compression ratio of the square block matching heuristic when applied to the CCITT test set is about 0.16 for $1 \leq k \leq 3$ and it deteriorates in a relevant way for $k \geq 4$.

The compression effectiveness of the variable-length coding technique depends on the sub-image size rather than on the whole image. In fact, if we apply the parallel procedure to the test set of larger binary images as the 4096 x 4096 pixels half-tone topographic images of figures 5-9 we obtain about the same compression effectiveness for $1 \leq k \leq 5$. The compression ratio is 0.28 with a 2 percent loss for $k = 6$. This means that actually the approach without interprocessor communication works in the context of unbounded parallelism as long as the elements of the image partition are large enough to capture the monochromatic rectangles encoded with four bits for each dimension. On the other hand, the compression ratio of the square block matching heuristic is 0.31 and depends on how the match size compares to the whole image. In fact, the compression effectiveness is again about the same for $1 \leq k \leq 3$ and deteriorates in a relevant way for $k \geq 4$.

We wish to remark at this point that parallel models have two types of complexity, the interprocessor communication and the input-output mechanism. While the input/output issue is inherent to any parallel 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 to design a practical algorithm. The simplest model for this phase is, of course, a simple array of processors with no interconnections and, therefore, no communication cost. Obviously, the compression and decompression procedures described here are suitable for such a model and implementable on both small (about 100 processors) and large (about 1000 processors) scale distibuted systems.

Image	1 proc.	2. proc.	4 proc.	8 proc.	16 proc.	32 proc.
1	41	22	12	7	4	2
2	40	23	14	7	4	2
3	42	22	15	8	4	2
4	42	25	15	7	4	2
5	41	22	13	7	4	3
Avg.	41.2	22.8	13.8	7.2	4	2.2

Figure 10. Compression times of the new procedure (cs.)

We show in figures 10 and 11 the compression and decompression times of the parallel procedure applied to the five images of figures 5-9, 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 run-

Image	1 proc.	2. proc.	4 proc.	8 proc.	16 proc.	32 proc.
1	20	11	6	4	2	1
2	20	11	7	4	2	1
3	21	11	8	4	2	1
4	21	13	8	4	2	1
5	20	11	7	4	2	1
Avg.	20.4	11.4	7.2	4	2	1

Figure 11. Decompression times of the new procedure (cs.)

ning 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 running times about twenty times faster than the sequential ones. Images 1 and 5 have the smallest compression time, while image 2 has the greatest one. Image 2 also has the greatest sequential decompression time while image 5 has the smallest one. The greatest compression time with 32 processors is given by image 5 while the decompression time with 32 processors is the same for all images.

5 A Massively Parallel Algorithm

Compression and decompression parallel implementations are presented, requiring $O(\alpha \log M)$ time with $O(n/\alpha)$ processors for any integer square value $\alpha \in \Omega(\log n)$ on a PRAM EREW. As in the previous section, we partition an m x m' image I in x x y rectangular areas, where x and y are $\Theta(\alpha^{1/2})$. In parallel for each area, one processor applies the sequential parsing algorithm so that in $O(\alpha \log M)$ time each area is parsed into rectangles, some of which are monochromatic. In the theoretical context of unbounded parallelism, α can be arbitrarely small and before encoding we wish to compute larger monochromatic rectangles. The monochromatic rectangles are computed by merging adjacent monochromatic areas without considering those monochromatic matches properly contained in some area. We denote with $A_{i,j}$ for $1 \leq i \leq \lceil m/x \rceil$ and $1 \leq j \leq \lceil m'/y \rceil$ the areas into which the image is partitioned. In parallel for $1 \leq i \leq \lceil m/x \rceil$, if i is odd, a processor merges areas $A_{2i-1,j}$ and $A_{2i,j}$ provided they are monochromatic and have the same color. The same is done horizontally for $A_{i,2j-1}$ and $A_{i,2j}$. At the k-th step, if areas $A_{(i-1)2^{k-1}+1,j}$, $A_{(i-1)2^{k-1}+2,j}$, $\cdots$ $A_{i2^{k-1},j}$, with i odd, were merged, then they will merge with areas $A_{i2^{k-1}+1,j}$, $A_{i2^{k-1}+2,j}$, $\cdots$ $A_{(i+1)2^{k-1},j}$, if they are monochromatic with the same color. The same is done horizontally for $A_{i,(j-1)2^{k-1}+1}$, $A_{i,(j-1)2^{k-1}+2}$, $\cdots$ $A_{i,j2^{k-1}}$, with j odd, and $A_{i,j2^{k-1}+1}$, $A_{i,j2^{k-1}+2}$, $\cdots$ $A_{i,(j+1)2^{k-1}}$. After $O(\log M)$ steps, the procedure is completed and each step takes $O(\alpha)$ time and $O(n/\alpha)$ processors since there is one processor for each area. Therefore, the image parsing phase is realized in $O(\alpha \log M)$ time with $O(n/\alpha)$ processors on an exclusive read, exclusive write shared memory machine. The interested reader can see how the coding and decoding phase are designed in [1], [10], since similar procedures are subroutines of the parallel block matching compression and decompression algorithms. As in the previous section, the flag field 110 corresponds the raw data so that we can indicate with 111 the end of the sequence of pointers corresponding to a given area. Since some areas could be entirely covered by a monochromatic rectangle 111 is followed by the index associated with the next area by the raster scan. Under some realistic assumptions, implementations with the same parallel complexity can be realized on a full binay tree architecture [1], [10].

6 Scalability and Compression Effectiveness

Since the images of the CCITT test set are 1728 x 2376 pixels, they can be partitioned into 4^k sub-images for $1 \leq k \leq 8$. For $k = 8$, the sub-image is 7 x 9 pixels. The extremal case is when we partition the image into 4 x 4 subarrays. As we mentioned in section 4, the compression ratio is 0.13 in average for $1 \leq k \leq 4$ and 0.14 for $k = 5$. For greater values of k, the compression effectiveness deteriorates. For $k = 6$, $k = 7$ and $k = 8$ the compression ratio is 0.17, 0.23 and 0.51 respectively. For the extremal case, the compression ratio is about 0.80. For $k \geq 6$ and the extremal case, we obviously have an improvement on the compression results if we use interprocessor communication to compute larger monochromatic rectangles with the procedure explained in the previous section. As shown in figure 12, the improvements are consistent but not satisfactory but for the extremal case. This is because we compute larger monochromatic matches by merging adjacent monochromatic elements of the image partition. This process implies that such matches are likely to be sub-arrays of larger monochromatic rectangles with the margins included in non-monochromatic elements of the partition. Yet, these rectangles are likely to be computed as matches by the sequential procedure. Therefore, the smaller the elements are the better the merging procedure works. A more sophisticated procedure would be to consider the monochromatic matches properly contained in some element but such approach is not very practical and does not provide a parallel decoder [5].

N. Pr.	c. ratio	c. ratio
$\leq 4^4$	.13	.13
4^5	.14	.14
4^6	.17	.16
4^7	.24	.19
4^8	.51	.29
$\sim 4^9$	.80	.15
comm.	without	with

Figure 12. Average compression and scalability on the CCITT images

As pointed out earlier, the compression effectiveness depends on the sub-image size rather than on the whole image. In fact, if we apply the parallel procedures with and without interprocessor communication to the test set of larger binary images of figures 5-9 we obtain the average results of figure 13. As we can see, the compression effectiveness has still a bell-shaped behaviour which mantains the sequential performance to a higher degree of parallelism with respect to the CCITT test set and is again competitive with the sequential procedure when the highest degree of parallelism is reached by the extremal case. This confirms that actually the approach without interprocessor communication works in the context of unbounded parallelism as long as the elements of the image partition are large enough to capture the monochromatic rectangles encoded with four bits for each dimension. On the other hand, interprocessor communication is effective only in the extremal case.

N. Pr.	c. ratio	c. ratio
$\leq 4^5$	.28	.28
4^6	.30	.29
4^7	.33	.33
4^8	.42	.39
4^9	.76	.60
$\sim 4^{10}$	.95	.30
comm.	without	with

Figure 13. Average compression and scalability on the 4096 x 4096 pixels images

7 Conclusions

We provided a parallel low-complexity lossless compression technique for binary images which is suitable for both small and large distributed memory systems at no communication cost. The degree of locality of the technique is high enough to garantee no loss of compression effectiveness when it is applied independently to blocks with dimensions about 50 x 50 pixels. When the blocks are smaller, interprocessor communication is needed but it is not effective except for the extremal case of blocks with dimensions 4 x 4 pixels. We presented experimental results with at most 32 processors and obtained the expected linear speed-up. As future work, we wish to experiment with more processors by implementing the procedure on a graphical processing unit.

References

1. S. D. Agostino: *Compressing bi-level images by block matching on a tree architecture*, in Proceedings of the Prague Stringology Conference, 2008, pp. 137–145.
2. T. C. Bell, J. G. Cleary, and I. W. Witten: *Text Compression*, Prentice Hall, 1990.
3. R. P. Brent: *A linear algorithm for data compression.* Australian Computer Journal, 19 1987, pp. 64–68.
4. L. Cinque and S. D. Agostino: *Lossless image compression by block matching on practical massively parallel architectures*, in Proceedings of the Prague Stringology Conference, 2008, pp. 26–34.
5. L. Cinque, S. D. Agostino, and F. Liberati: *A work-optimal parallel implementation of lossless image compression by block matching.* Nordic Journal of Computing, 10 2003, pp. 11–20.
6. L. Cinque, S. D. Agostino, F. Liberati, and B. Westgeest: *A simple lossless compression heuristic for grey scale images*, in Proceedings of the Prague Stringology Conference, 2004, pp. 48–55.
7. L. Cinque, S. D. Agostino, F. Liberati, and B. Westgeest: *A simple lossless compression heuristic for grey scale images.* International Journal of Foundations of Computer Science, 16 2009, pp. 1111–1119.
8. L. Cinque, S. D. Agostino, and L. Lombardi: *Speeding up lossless image compression: Experimental results on a parallel machine*, in Proceedings of the Prague Stringology Conference, 2008, pp. 35–45.

9. L. Cinque, S. D. Agostino, and L. Lombardi: *Lossless image compression by block matching on practical parallel architectures.* Texts in Algorithmics, 11 2009, pp. 136–151.
10. L. Cinque, S. D. Agostino, and L. Lombardi: *Scalability and communication in parallel low-complexity lossless compression.* Mathematics in Computer Science, 3 2010, pp. 391–406.
11. J. Gailly and M. Adler: *The gzip compressor.* http://www.gzip.org, 1991.
12. S. W. Golomb: *Run length encodings.* IEEE Transactions on Information Theory, 12 1966, pp. 399–401.
13. 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.
14. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23 1977, pp. 337–343.
15. R. F. Rice: *Some practical universal noiseless coding technique – part I.* Technical Report JPL-79-22 Jet Propulsion Laboratory, 1979.
16. J. Rissanen: *Generalized kraft inequality and arithmetic coding.* IBM Journal on Research and Development, 20 1976, pp. 198–203.
17. J. Rissanen and G. G. Langdon: *Universal modeling and coding.* IEEE Transactions on Information Theory, 27 1981, pp. 12–23.
18. J. A. Storer: *Lossless image compression using generalized LZ1-type methods*, in Proceedings of the 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. J. A. Storer and T. G. Szymanski: *Data compression via textual substitution.* Journal of ACM, 29 1982, pp. 928–951.
21. J. R. Waterworth: *Data compression system.* US Patent 4 701 745, 1987.
22. M. J. Weimberger, G. Seroussi, and G. Sapiro: *LOCO-I: A low-complexity, context based, lossless image compression algorithn*, in Proceedings of the IEEE Data Compression Conference, 1996, pp. 140–149.
23. D. A. Whiting, G. A. George, and G. E. Ivey: *Data compression apparatus and method.* US Patent 5016009, 1991.
24. X. Wu and M. D. Memon: *Context-based, adaptive, lossless image coding.* IEEE Transactions on Communications, 45 1997, pp. 437–444.