Binary Image Compression via Monochromatic Pattern Substitution: A Sequential Speed-Up

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

[1] Computer Science Department, Sapienza University, 00198 Rome, Italy
deagostino@di.uniroma1.it
[2] Computer Science Department, University of Pavia, 27100 Pavia, Italy

Abstract. A method for compressing binary images is monochromatic pattern substitution. Monochromatic rectangles inside the image are detected and 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 show in this paper that such method has a speed-up if applied sequentially to the partitioned image. Experimental results show that the speed-up happens if the image is partitioned into up to 256 blocks and sequentially each block is compressed independently. It follows that the sequential speed-up can also be applied to a parallel implementation on a small scale system.

Keywords: lossless compression, binary image, sequential algorithm, parallel computing, distributed system

1 Introduction

A low-complexity binary image compressor has been designed in [3], which employs monochromatic pattern substitution and is implementable on small and large scale parallel systems. When it comes to parallel implementations, we wish to remark 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. Compression via monochromatic pattern substitution 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.
Another low-complexity compressor for binary images is BLOCK MATCHING [6], [7], which extends data compression via textual substitution to two-dimensional data by compressing sub-images rather than substrings [5], [8]. However, it does not work locally since it applies a generalized LZ1-type method with an unrestricted window and it is not scalable [2], [4].
In this paper, we show that monochromatic pattern substitution has a speed-up if

Luigi Cinque, Sergio De Agostino, Luca Lombardi: Binary Image Compression via Monochromatic Pattern Substitution: A Sequential Speed-Up, pp. 220–225.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2

Figure 2. Sequential decompression times on the CCITT images (ms.)

2 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 4×4 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 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 4×4 subarray (raw data).

Figure 5. Sequential compression times on the 4096×4096 pixels images (ms.)

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.
The procedure for computing the largest monochromatic rectangle with left upper corner in position (i, j) takes $O(M \log M)$ time, where M is the size of the rectangle [3]. The positions covered by the detected rectangles are skipped in the linear

Figure 8. Parallel decompression times on the 4096×4096 pixels images (ms.)

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 detected monochromatic rectangles 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.

3 The Sequential Speed-Up

The variable length coding technique explained in the previous section has been applied to the CCITT test set of bi-level images. The images of the CCITT test set are 1728×2376 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 $1 \leq k \leq 4$. Though the waste factor decreases with the increasing of k. As mentioned in the previous section, the waste factor is less than 2 on realistic image data (as the CCITT test set) for $k = 0$ and decreases to about 1 when $k = 4$. It follows that if we refine the partition by splitting the blocks horizontally and vertically, after four refinements no further relevant speed-up is obtained (we consider a speed-up relevant if it has the order of magnitude of one centisecond). The experimental results in figure 1 show the speed-up obtained if the

image is partitioned into up to 256 blocks and sequentially each block is compressed independently. Obviously, there is a similar speed-up for the decompressor as shown in figure 2. Decompression is about twice faster than compression. Compression and decompression running times were obtained using a single core of a quad core with a CPU Intel Core 2 Quad Q9300 – 2.5 GHz with 3.25 GB of RAM.
The sequential speed-up can also be applied to a parallel implementation on a small scale system since the experimental results show that the speed-up happens for an image partitioned into less than 256 blocks. However, in order to decompress in parallel raw data are associated with the flag field 110 so that we can indicate with 111 the end of the encoding of a block. The parallel compression and decompression running times on the CCITT image test set are given in figures 3 and 4 using the four cores of the quadcore machine. Obviously, a similar experiment could be run using two refinemets or one refinement of the partition on a system with 16 or 64 cores respectively. Experimental results with 16 processors on test set of larger topographic images are presented in the next section.

4 Speeding-Up Parallel Computation

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 consider a test set of larger binary images as the five 4096×4096 pixels half-tone topographic images shown in [3] and these images are partitioned into 4^k sub-images, again we obtain about the same compression effectiveness for $1 \leq k \leq 4$ and a sequential speed-up with the increasing of k. On the other hand, no further speed-up is obtained for $k = 5$, that is, the waste factor seems to be determined by the number of refinements independently from the image size on realistic data. We give in figures 5 and 6 the compression and decompression running times with one processor of a 256 Intel Xeon 3.06 GHz CPU's machine (`avogadro.cilea.it`) on the five images before and after the partition into 256 blocks. The compression and decompression running times with 16 processors before and after the partition into 256 blocks are given in figures 7 and 8. This means that each processor works on a sub-image partitioned into 16 blocks when the number of blocks is 256. Running times are given as milliseconds, which are the time units used for the quadcore experiments, but the centisecond is the actual time unit employed with the avogadro machine and the running time is provided as an integer number.

5 Conclusions

In this paper, we showed that the most efficient way to apply monochromatic pattern substitution to binary image compression with a sequential algorithm is to compress independently the 256 blocks of a partitioned input image. Since a speed-up is obtained as well with partitions of lower cardinality, this can be used to improve the performance of parallel compression on a small scale distributed system. We presented experimental results with four and sixteen processors. As future work, we wish to experiment with more processors by implementing the procedure on a graphical processing unit [1].

References

1. L. Bianchi, R. Gatti, and L. Lombardi: *The future of parallel computing: Gpu vs cell - general purpose planning against fast graphical computation architecture, which is the best solution for general purpose computation*, in Proceedings GRAPP, 2008, pp. 419–425.
2. L. Cinque, S. D. Agostino, and L. Lombardi: *Speeding-up lossless image compression: Experimental results on a paralle machine*, in Proceedings Prague Stringology Conference, 2008, pp. 35–45.
3. L. Cinque, S. D. Agostino, and L. Lombardi: *Binary image compression via monochromatic pattern substitution: Scalability and effectiveness*, in Proceedings Prague Stringology Conference, 2010, pp. 103–115.
4. 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.
5. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23 1977, pp. 337–343.
6. J. A. Storer: *Lossless image compression using generalized lz1-type methods*, in Proceedings IEEE Data Compression Conference, 1996, pp. 290–299.
7. J. A. Storer and H. Helfgott: *Lossless image compression by block matching.* The Computer Journal, 40 1997, pp. 137–145.
8. J. A. Storer and T. G. Szymanski: *Data compression via textual substitution.* Journal of ACM, 29 1982, pp. 928–951.