Lossless Image Compression by Block Matching on Practical Massively Parallel Architectures

Luigi Cinque and Sergio De Agostino

Computer Science Department
Sapienza University
Via Salaria 113, 00198 Roma, Italy
{cinque, deagostino}@di.uniroma1.it

Abstract. Work-optimal $O(\log M \log n)$ time implementations of lossless image compression by block matching are shown on the PRAM EREW, where n is the size of the image and M is the maximum size of the match, which can be implemented on practical architectures such as meshes of trees, pyramids and multigrids. The work-optimal implementations on pyramids and multigrids are possible under some realistic assumptions. Decompression on these architectures is also possible with the same parallel computational complexity.

Keywords: lossless compression, sliding dictionary, bi-level image, parallel architecture.

1 Introduction

Storer suggested that fast encoders are possible for two-dimensional lossless compression by showing a square greedy matching heuristic for bi-level images, which can be implemented by a simple hashing scheme [8]. 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 [9]. Therefore, the sequential time to compress an image of size n by rectangle matching is $\Omega(n \log M)$.

The technique is a two-dimensional extension of LZ1 compression [7]. Simple and practical heuristics exist to implement LZ1 compression by means of hashing techniques [1], [10], [11]. The hashing technique used for the two-dimensional extension is even simpler.

Among the different ways of reading an image, we assume that the rectangle matching compression heuristic is scanning 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 rectangle, 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 rectangle greedy match and then replaced in the hash table by a pointer to the current position. As mentioned above, the procedure for computing the largest rectangle match with left upper corners in positions (i, j) and (k, h) takes $O(M \log M)$ time, where M is the size of the match. Obviously, this procedure can be used for computing the largest monochromatic rectangle 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 rectangle in that position. Otherwise, we compute the largest rectangle 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

Luigi Cinque, Sergio De Agostino: Lossless Image Compression by Block Matching on Practical Massively Parallel Architectures, pp. 26–34.
Proceedings of PSC 2008, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04145-1

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 and the sequential time to compress an image of size n by rectangle matching is $\Omega(n \log M)$. We want to point out that besides the proper matches we call a match every rectangle of the parsing of the image produced by the heuristic. We also call pointer the encoding of a match.

The analysis of the running time of these algorithms involve a so called *waste factor*, defined as the average number of matches covering the same pixel. In [9], it is conjectured that the waste factor is less than 2 on realistic image data. Therefore, the square greedy matching heuristic takes linear time while the rectangle greedy matching heuristic takes $O(n \log M)$ time. On the other hand, the decoding algorithms are both linear.

Work-optimal parallel coding algorithms for lossless image compression by block matching were shown on the PRAM-EREW [3], [4] and the mesh of trees [5], which is a hybrid network architecture based on arrays and trees [6], requiring $O(\log M \log n)$ time and $O(n/\log n)$ processors. The design of a parallel decoder was left as an open problem as well as the implementation on even simpler architectures as pyramids and multigrids. By slightly modifying the encoder, new parallel coding and decoding algorithms were shown in [2] still requiring $O(\log M \log n)$ time and $O(n/\log n)$ processors on the PRAM-EREW. On the mesh of trees though the decoder required $O(\log^2 n)$ time. In this paper, we show how to implement $O(\log M \log n)$ time, $O(n/\log n)$ processors coding and decoding algorithms on the PRAM EREW, mesh of trees, pyramidal, and multigrid architectures.

In section 2, we describe the PRAM EREW encoder and decoder. In section 3, we explain how the parallel encoder and decoder are implemented on the mesh of trees. In section 4, we explain how the parallel encoder and decoder are implemented on the pyramid with the same parallel complexity under some realistic assumptions. Conclusions and future work are given in section 4 where the implementations on the multigrid, which is the simplest of the architectures mentioned above, are discussed.

2 The PRAM EREW Encoder and Decoder

To achieve sublinear time we partition an m x m' image I in x x y rectangular areas, where x and y are $\Theta(\log^{1/2} n)$, and n is the size of the image. In parallel for each area, one processor applies the sequential parsing algorithm so that in $O(\log M \log n)$ time each area will be parsed in rectangles, some of which are monochromatic. Before encoding we wish to compute larger monochromatic rectangles.

2.1 Computing the Monochromatic Rectangles

Differently from [3], we compute larger monochromatic rectangles by merging adjacent monochromatic areas without considering those monochromatic rectangles properly contained in some area. In practice, these areas are very small and such limitation has no relevant effect on the compression ratio.

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 $A_{(i-1)2^{k-1}+1,j}$, $A_{(i-1)2^{k-1}+2,j}$, ..., $A_{i2^{k-1},j}$, with i odd, were merged, then they will merge with areas

$A_{i2^{k-1}+1,j}$, $A_{i2^{k-1}+2,j}$, ..., $A_{(i+1)2^{k-1},j}$, if they are monochromatic with the same color. The same is done horizontally for $A_{i,(j-1)2^{k-1}+1}$, $A_{i,(j-1)2^{k-1}+2}$, ..., $A_{i,j2^{k-1}}$, with j odd, and $A_{i,j2^{k-1}+1}$, $A_{i,j2^{k-1}+2}$, ..., $A_{i,(j+1)2^{k-1}}$. After $O(\log M)$ steps, the procedure is completed and each step takes $O(\log n)$ time and $O(n/\log n)$ processors since there is one processor for each area of logarithmic size. Therefore, the image parsing phase is realized with $O(\log M \log n)$ time and $O(n/\log n)$ processors on the PRAM EREW.

2.2 The Parallel Encoder

We derive the sequence of pointers from the image parsing computed above. In $O(\log n)$ time with $O(n/\log n)$ processors we can identify every upper left corner of a match (proper, monochromatic, or raw) by assigning a different segment of logarithmic length on a row to each processor. Each processor detects the upper left corners on its segment. Then, by parallel prefix we obtain a sequence of pointers decodable by the decompressor paired with the sequential heuristic. However, the decoding of such sequence seems hard to parallelize. In order to design a parallel decoder, it is more suitable to produce the sequence of pointers by a raster scan of each of the areas into which the image was originally partitioned, where the areas are ordered by a raster scan themselves. Then, again we can easily derive the sequence of pointers in $O(\log n)$ time with $O(n/\log n)$ processors by detecting in each of the areas the upper left corners of a match and producing the sequence of pointers by parallel prefix.

As mentioned in the introduction, 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). For the feasibility of the parallel decoder, we want to indicate the end of the encoding of the sequence of matches with the upper left corner in a specific logarithmic 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. Moreover, since some areas could be entirely covered by a monochromatic match 1111 is followed by the index associated with the next area by the raster scan. The pointer of a monochromatic match has fields for the width and the length while the pointer of a proper match also has fields for the coordinates of the left upper corner of the copy in the window. In order to save bits, the value stored in any of these fields is the binary value of the field plus 1 (so, we employ the zero value). This coding technique is more redundant than others previously designed, but its compression effectiveness is still better than the one of the square greedy matching technique.

2.3 The Parallel Decoder

The parallel decoder has three phases. Observe that at each position of the binary sequence encoding the image, we read a subsequence of bits that is either 1111 (recall that the k bits following 1111 provide the area index, where k is the number of bits used to encode it) or can be interpreted as a flag field of a pointer. Then, in the first phase we reduce the binary sequence to a doubly-linked structure and apply the well-known Euler tour technique in order to identify for each area the corresponding pointers. The reduction works as follows: link each position p of the sequence to the position next to the end of the subsequence starting in position p that either can be

interpreted as a pointer or is equal to 1111 followed by k bits. For those suffixes of the sequence which can be interpreted as pointers, their first positions are linked to a special node denoting the end of the coding. For those suffixes of the sequence which cannot be interpreted as pointers, their first positions are not linked to anything. The linked structure is a forest with one tree rooted in the special node denoting the end of the coding and the other trees rooted in the first position of a suffix of the encoding sequence not interpretable as a pointer. The first position of the binary sequence is a leaf of the tree rooted in the special node. The sequence of pointers encoding the image is given by the path from the first position to the root. In order to compute such path we need the children to be doubly-linked to the parent. Then, we need to reserve space for each node to store the links to the children. Each node has at most five children since there are only four different pointer sizes (white, black, raw, or proper match). So, for each position p of the binary sequence we set aside five locations $[p,1]\cdots[p,5]$, initially set to zero. When a link is added from position p' to p, depending on whether the subsequence starting in position p' is 1111 or can be interpreted as a pointer to a raw, white, black or proper match, the value p' is overwritten on location $[p,1]$, $[p,2]$, $[p,3]$ $[p,4]$ or $[p,5]$, respectively. Then, by means of the well-known Euler technique we can linearize the linked structure and apply list ranking to obtain the path from the first position of the sequence to the root of its tree. It is well-known that all this can be computed in $O(\log n)$ time with $O(n/\log n)$ processors on the PRAM EREW, since the row image size is greater than the size of the sequence. Then, still in $O(\log n)$ time with $O(n/\log n)$ processors we can identify the positions on the path corresponding to 1111.

In the second phase of the parallel decoder a different processor decodes the sequence of pointers corresponding to a different area. As far as the pointers to monochromatic matches are considered, each processor decompresses either a match contained in an area or the portion of the match corresponding to the left upper area. Therefore, after the second phase an area might not be decompressed. Obviously, the second phase requires $O(\log n)$ time and $O(n/\log n)$ processors on the PRAM EREW.

The third phase completes the decoding. In the previous subsection, we denoted with $A_{i,j}$ for $1 \le i \le \lceil m/x \rceil$ and $1 \le j \le \lceil m'/y \rceil$ the areas into which the image was partitioned by the encoder. At the first step of the third phase, one processor for each area $A_{2i-1,j}$ decodes $A_{2i,j}$, if $A_{2i-1,j}$ is the upper left portion of a monochromatic match and the length field of the corresponding pointer informs that $A_{2i,j}$ is part of the match. The same is done horizontally for $A_{i,2j-1}$ and $A_{i,2j}$ (using the width field of its pointer) if it is known already by the decoder that $A_{i,2j-1}$ is part of a monochromatic match. Similarly at the k-th step, one processor for each of the areas $A_{(i-1)2^{k-1}+1,j}$, $A_{(i-1)2^{k-1}+2,j}$, ..., $A_{i2^{k-1},j}$, with i odd, decodes the areas $A_{i2^{k-1}+1,j}$, $A_{i2^{k-1}+2,j}$, ..., $A_{(i+1)2^{k-1},j}$, respectively. The same is done horizontally for $A_{i,(j-1)2^{k-1}+1}$, $A_{i,(j-1)2^{k-1}+2}$, ..., $A_{i,j2^{k-1}}$, with j odd, and $A_{i,j2^{k-1}+1}$, $A_{i,j2^{k-1}+2}$, ..., $A_{i,(j+1)2^{k-1}}$. After $O(\log M)$ steps the image is entirely decompressed. Each step takes $O(\log n)$ time and $O(n/\log n)$ processors since there is one processor for each area of logarithmic size. Therefore, the decoder is realized with $O(\log M \log n)$ time and $O(n/\log n)$ processors on the PRAM EREW.

3 The Mesh of Trees Implementations

A *mesh of trees* is a network of $3N^2 - 2N$ processors with N being a power of 2, consisting of an N x N grid where a complete binary tree of processors is built on

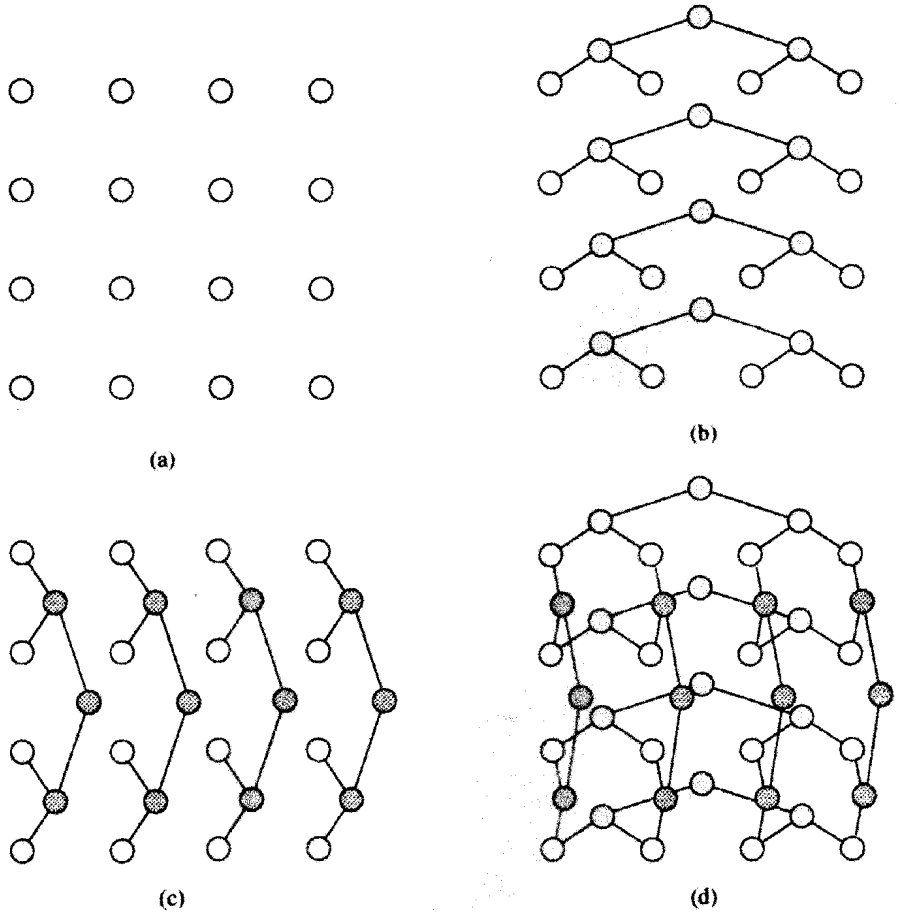

Figure 1. The construction of a mesh of trees with a 4x4 processor grid.

each row and each column as shown in Figure 1. First, we give a detailed description of the mesh of trees compression algorithm. Then, we provide the implementation of the parallel decoder.

Let max be equal to max $\{m, m'\}$. We assume m and m' have the same order of magnitude, as in practice with the height and the width of an image. Let N be the smallest power of two greater than $\lceil max/\log^{1/2} n \rceil$, where n is the size of an m x m' image. Then, the number of processors of the mesh of trees is $O(n/\log n)$ and we can store the logarithmic rectangular areas into which the parallel algorithm partitions the image into the N x N grid. Starting from the upper left corner of the grid, a different processor stores a different rectangular area and applies the sequential compression heuristic to such area. The remaining processors are inactive. From now on, we will refer only to the active processors.

3.1 Computing the Monochromatic Rectangles

After the compression heuristic has been executed on each area, it is easy to see that the PRAM EREW procedure to compute larger monochromatic rectangles can be implemented on a mesh of trees with the same number of processors without

slowing it down. In fact, if i is odd, the processors storing areas $A_{2i-1,j}$ and $A_{2i,j}$ merge them 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 $A_{(i-1)2^{k-1}+1,j}$, $A_{(i-1)2^{k-1}+2,j}$, ..., $A_{i2^{k-1},j}$, with i odd, were merged, the processor storing area $A_{i2^{k-1},j}$ will broadcast to the processors storing the areas $A_{i2^{k-1}+1,j}$, $A_{i2^{k-1}+2,j}$, ..., $A_{(i+1)2^{k-1},j}$ to merge with the above areas, if they are monochromatic with the same color. This is done in logarithmic time using the column trees. The same is done horizontally for $A_{i,(j-1)2^{k-1}+1}$, $A_{i,(j-1)2^{k-1}+2}$, ..., $A_{i,j2^{k-1}}$, with j odd, and $A_{i,j2^{k-1}+1}$, $A_{i,j2^{k-1}+2}$, ..., $A_{i,(j+1)2^{k-1}}$, using the row trees. After $O(\log M)$ steps, the procedure is completed and each step takes $O(\log n)$ time and $O(n/\log n)$ processors since there is one processor for each area of logarithmic size. Therefore, the image parsing phase is realized with $O(\log M \log n)$ time and $O(n/\log n)$ processors on the mesh of trees.

3.2 The Parallel Encoder

The sequence of pointers for each area can be trivially produced on the grid. If we assume the possibility of a parallel output, the sequences can be put together by parallel prefix. This can be realized in $O(\log n)$ time on a mesh of trees with $O(n/\log n)$ processors.

3.3 The Parallel Decoder

We know that the end of the encoding of an area is indicated by 1111 followed by the index of the area corresponding to the next encoding. Then, we can store the encodings in the positions of the grid corresponding to the locations of the areas in the image. In fact, the first phase of the PRAM EREW decoding algorithm corresponds to the input process of a distributed memory system (as the mesh of trees is) and is not part of our complexity analysis. At this point, each processor on the grid completes the second phase of the decoder described in subsection 2.3. Then, it is easy to see that the third and last phase of the PRAM decoder is implementable on a mesh of trees with the same number of processor and no slowdown. In conclusion, the decoder takes $O(\log M \log n)$ time on a mesh of trees with $O(n/\log n)$ processors.

4 The Pyramid Implementations

An N x N *pyramid* is a network consisting of $\log N + 1$ two-dimensional grids with N being a power of 2, each one of size $N/2^k$ x $N/2^k$ for $0 \leq k \leq \log N$. The grids are interconnected so that the (i, j) processor on the 2^k x 2^k grid is connected to processors $(2i-1, 2j-1)$, $(2i-1, 2j)$, $(2i, 2j-1)$ and $(2i, 2j)$ on the 2^{k+1} x 2^{k+1} grid, as shown in Figure 2 for the 4 x 4 pyramid network. As for the mesh of trees, let N be the smallest power of two greater than $\lceil max/\log^{1/2} n \rceil$, where n is the size of an m x m' image and max is equal to max $\{m, m'\}$. If we assume that m and m' have the same order of magnitude, the number of processors of the pyramid is $O(n/\log n)$ and we can store the logarithmic rectangular areas into which the parallel algorithm partitions the image into the N x N grid. Starting from the upper left corner of the grid, a different processor stores a different rectangular area and applies the sequential compression heuristic to such area while the other processors remain inactive. From now on, we will refer only to the active processors.

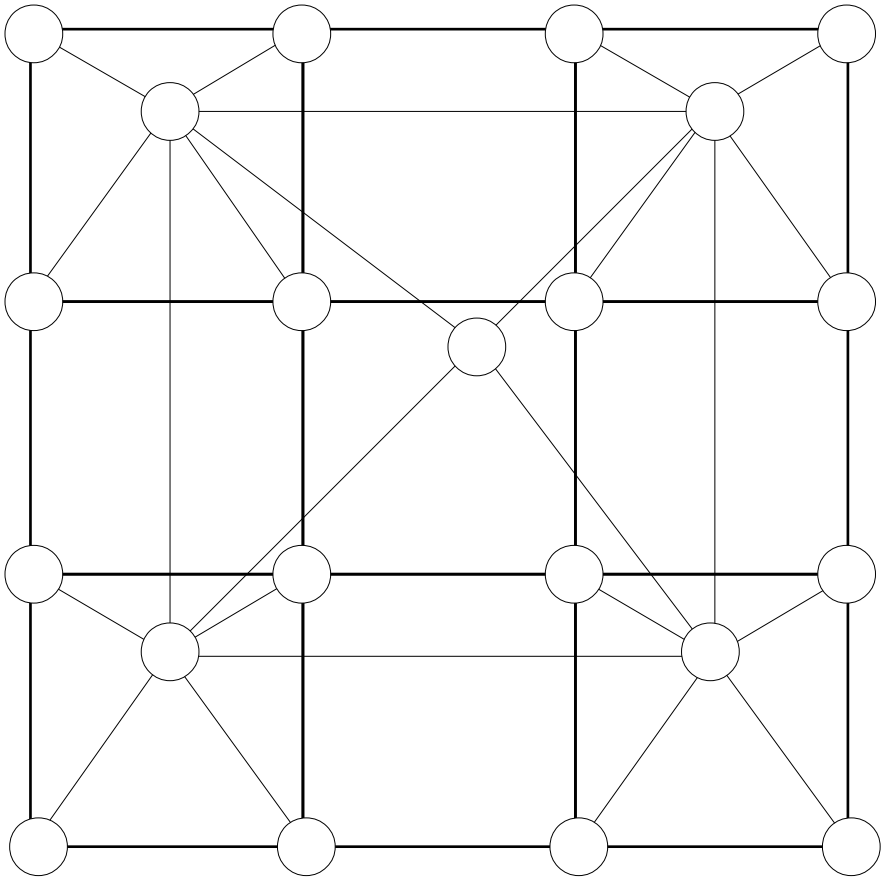

Figure 2. A 4 x 4 pyramid network.

4.1 Computing the Monochromatic Rectangles

After the compression heuristic has been executed on each area, we have to show how the PRAM EREW procedure to compute larger monochromatic rectangles can be implemented on a pyramid with the same number of processors without slowing it down. This is possible by making some realistic assumptions. Let ℓ_R and w_R be the length and the width of a monochromatic match R, respectively. We define $s_R = \max\{\ell_R, w_R\}$. We make a first assumption that the number of monochromatic matches R with $s_R \geq 2^k \lceil \log^{1/2} n \rceil$ is $O(N^2/2^{2k})$ for $1 \leq k \leq \log N - 1$. If i is odd, the processors storing areas $A_{2i-1,j}$ and $A_{2i,j}$ merge them 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 $A_{(i-1)2^{k-1}+1,j}$, $A_{(i-1)2^{k-1}+2,j}$, ..., $A_{i2^{k-1},j}$, with i odd, were merged for $w_1 \leq j \leq w_2$, the processor storing area A_{i2^{k-1},w_2} will broadcast to the processors storing the areas $A_{i2^{k-1}+1,j}$, $A_{i2^{k-1}+2,j}$, ..., $A_{(i+1)2^{k-1},j}$ to merge with the above areas for $w_1 \leq j \leq w_2$, if they are monochromatic with the same color. The same is done horizontally, that is, if $A_{i,(j-1)2^{k-1}+1}$, $A_{i,(j-1)2^{k-1}+2}$, ..., $A_{i,j2^{k-1}}$, with j odd, were merged for $\ell_1 \leq i \leq \ell_2$, the processor storing area $A_{\ell_2,j2^{k-1}}$ will broadcast to the processors storing the areas $A_{i,j2^{k-1}+1}$, $A_{i,j2^{k-1}+2}$, ..., $A_{i,(j+1)2^{k-1}}$ to merge with the above areas for $\ell_1 \leq i \leq \ell_2$, if they are monochromatic with the same color. After $O(\log M)$ steps, the procedure is completed. If the waste factor is less than 2, as conjectured in [9], we can make a second assumption that each pixel is covered by a constant small number of monochromatic matches. It follows from this second

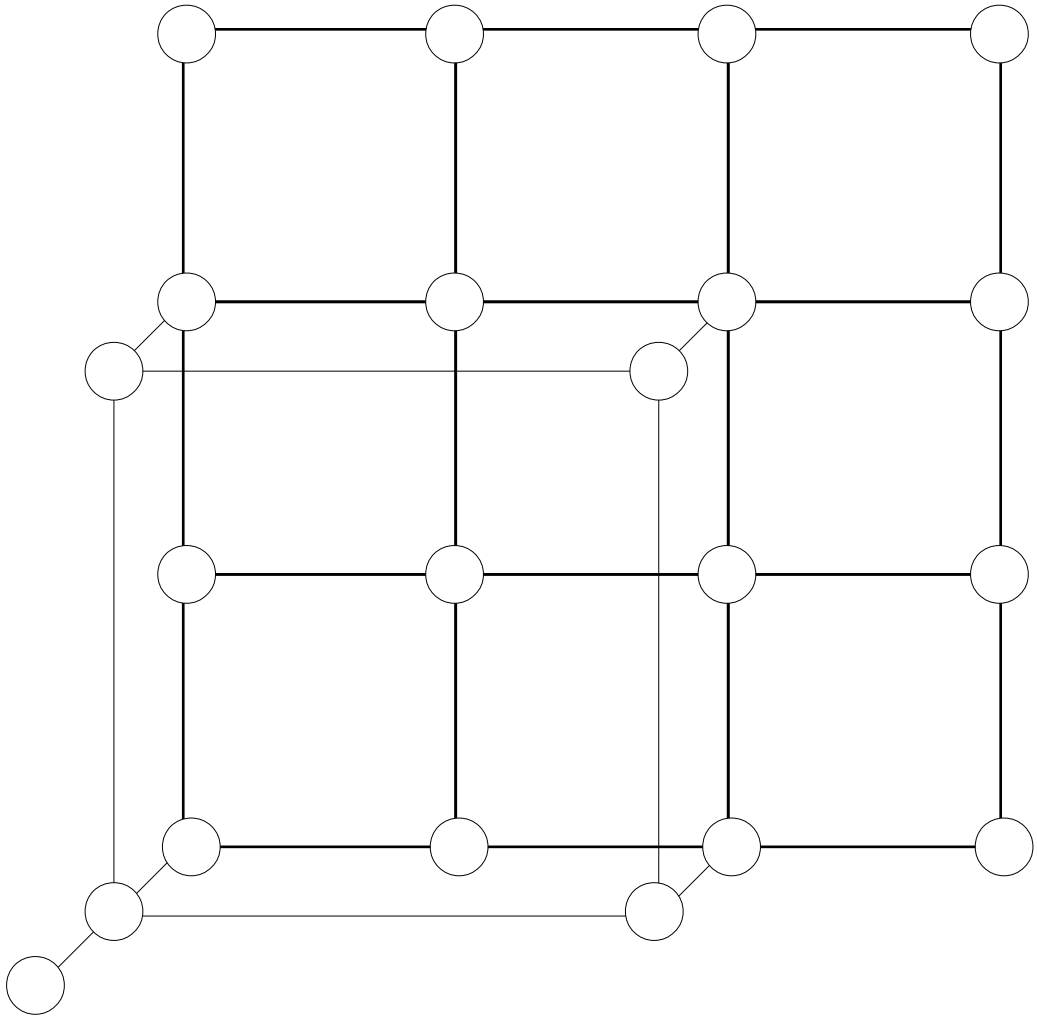

Figure 3. A 4 x 4 multigrid network.

assumption that the information about the monochromatic matches is distributed among the processors of a grid in a way very close to uniform. Then, it follows from the first assumption that the amount of information each processor of the grid at level k must broadcast is constant, for $1 \leq k \leq \log N - 1$. Therefore, each step takes $O(\log n)$ time and the image parsing phase is realized with $O(\log M \log n)$ time and $O(n/\log n)$ processors on the pyramid. Finally, we want to point out that the unique processor of the 1 x 1 grid at level $\log n$ is not involved in the computation of the image parsing and is used only for the ineherently sequential input/output operations which have, generally speaking, standard solutions for network algorithms.

4.2 The Parallel Encoder

The sequence of pointers for each area can be trivially produced on the grid at level 0. This is, obviously, realized in $O(\log n)$ time on a pyramid with $O(n/\log n)$ processors.

4.3 The Parallel Decoder

As for the mesh of trees, we can store the encodings of each area in the positions of the grid at level 0 corresponding to the locations of the areas in the image. At this

point, each processor on the grid completes the second phase of the decoder described in subsection 2.3. Then, it is easy to see that the third and last phase of the PRAM decoder is implementable on a pyramid with the same number of processor and no slowdown, if the same realistic assumptions are made. In conclusion, the decoder takes $O(\log M \log n)$ time on a pyramid with $O(n/\log n)$ processors.

5 Conclusions

Parallel coding and decoding algorithms for lossless image compression by block matching were shown requiring $O(\log M \log n)$ time and $O(n/\log n)$ processors on the PRAM-EREW, the mesh of trees and the pyramid. The parallel coding algorithms are work-optimal since the sequential time required by the coding is $\Omega(n \log M)$. On the other hand, the parallel decoding algorithms are not work-optimal since the sequential decompression time is linear. The mesh of trees and pyramid implementations of the decoder have the same performance of the PRAM EREW implementation if we do not consider the input process. The pyramid is a simpler architecture than the mesh of trees [6] and needs some realistic assumptions to give the same performance. There exist real parallel machines whose architecture is a pyramid. One of them is at the University of Pavia in Italy (PAPIA). As future work, we wish to implement our algorithm on this machine. An even simpler architecture than the pyramid is the multigrid, which is computationally equivalent up to a factor of 2 in speed to the pyramid [6]. Since the multigrid is a subgraph of the pyramid (Figure 3), we wish to implement and experiment our algorithm on this architecture as well.

References

1. R. P. Brent: *A linear algorithm for data compression.* Australian Computer Journal, 19 1987, pp. 64–68.
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, 10 2003, pp. 13–20.
4. S. DeAgostino: *A work-optimal parallel implementation of lossless image compression by block matching*, in Proceedings Prague Stringology Conference, 2002, pp. 1–8.
5. S. DeAgostino: *Lossless image compression by block matching on a mesh of trees*, in Proceedings IEEE Data Compression Conference, Poster Session, 2006, p. 443.
6. F. T. Leighton: *Introduction to Parallel Algorithms and Architectures*, Morgan-Kaufmann, 1992.
7. A. Lempel and J. Ziv: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23 1977, pp. 337–343.
8. J. A. Storer: *Lossless image compression using generalized lz1-type methods*, in Proceedings IEEE Data Compression Conference, 1996, pp. 290–299.
9. J. A. Storer and H. Helfgott: *Lossless image compression by block matching.* The Computer Journal, 40 1997, pp. 137–145.
10. J. R. Waterworth: *Data compression system.* US Patent 4 701 745, 1987.
11. D. A. Whiting, G. A. George, and G. E. Ivey: *Data compression apparatus and method.* US Patent 5016009, 1991.