# Image Rectangular Decomposition on a Reconfigurable Mesh *

*Chin-Hsiung Wu*†‡ *Shi-Jinn Horng*‡§ *Pei-Zong Lee*§

†Department of Information Management, Chinese Naval Academy, Kaohsiung, Taiwan, R. O. C.

‡Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan, R. O. C.
E-mail:horng@mouse.ee.ntust.edu.tw

§Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan, R. O. C.

## Abstract

The major contribution of this paper is in designing an efficient parallel algorithm for rectangular partitioning on the binary image. Based on the reconfigurability and power of the reconfigurable meshes, a constant time algorithm for rectangular partitioning for an $N \times N$ binary image using $N \times N$ processors is derived. For applications of this algorithm, an approach for compressing and decompressing binary images is also proposed. In the sense of the product of time and the number of processors used, our algorithms are time and cost optimal.

## 1 Introduction

Binary image representation techniques are mainly classified into three categories, namely tree, string and set of codes [12]. The set of coding is a very compact representation scheme. In the set of code type of representation, the image is represented as a set of codes, where each code represents a corresponding region of the image. Linear quadtree [4], interpolation-based binary tree [11], rectangular coding [1], blocking coding [16], and image block representation [13] fall in this category. This type of image representation has many applications because it saves much space and basic image operations can be efficiently performed.

Region-based representation of images is an important issue in image processing, computer vision and in other fields. Previously, many methods had been proposed to represent an $N \times N$ binary image as a number of rectangular regions with black pixels (i.e. 1's) instead of representing it by a 2-D array [1, 9, 13]. We refer these methods as rectangle-based representations. These algorithms partition the black pixels of the input image into a set of non-overlapping rectangular regions. The most important characteristic of the rectangle-based representation is that a perception of image parts greater than a pixel and all the image operations on the pixels belonging to a rectangle may be substituted by a simple operation on the rectangle. Therefore, the space and time complexities of applications fully depend on the cardinality of the set of rectangular regions. In the worst case, each rectangular region consists of only one pixel. Like the chessboard image, the number of rectangular regions is $N^2/2$. In

*This work was partially supported by the National Science Council under the contract no. NSC-89-2213-E011-007. Part of this work was carried out when the second author was visiting the Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan, July - December 1999.

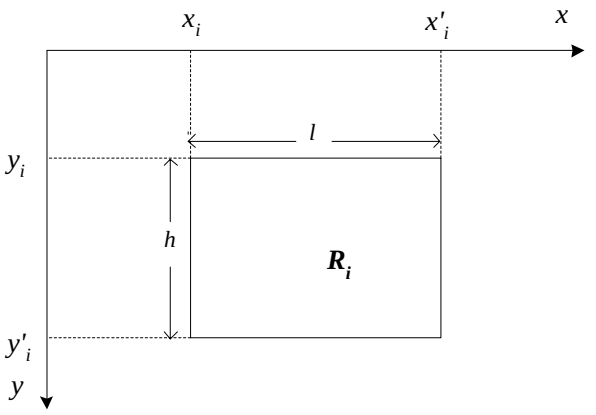


Figure 1: Rectangle $R_i$ is described by the coordinates of its two opposite corners.

most cases, however, the rectangle-based representations are superior to quadtree representation and interpolation-based binary tree representation, because the number of rectangular regions is significantly smaller than the number of nodes. Thus, using rectangle-based representation, the computational complexity of image processing can be reduced significantly.

Given an $N \times N$ binary image, the rectangle-based representation of it is a set of non-overlapping rectangles that completely cover the black pixels and it can be defined as follows. These rectangles have their edges parallel to the image axes and contain an integer number of black pixels. Let the rectangles denoted by $R_i$, $0 \leq i < r$, where $r$ is the number of rectangles. Only the two locations of any two opposite corners of each rectangle would be sufficient to represent the whole rectangle. That is, $R_i = (x_i, x'_i, y_i, y'_i)$, where $(x_i, y_i)$ and $(x'_i, y'_i)$ are the coordinates of the top left corner and the bottom right corner of rectangle $R_i$, respectively, as shown in Figure 1. For example, the image shown in Figure 2(a) can be described by non-overlapping rectangles, and rectangle $R_i$, $0 \leq i < r$, can be represented by $(x_i, x'_i, y_i, y'_i)$ or $(x_i, y_i, l, h)$, where $l_i$ and $h_i$ represent the length and height of rectangle $R_i$, respectively (see Figure 2(b)).

Most image processing problems are compu-

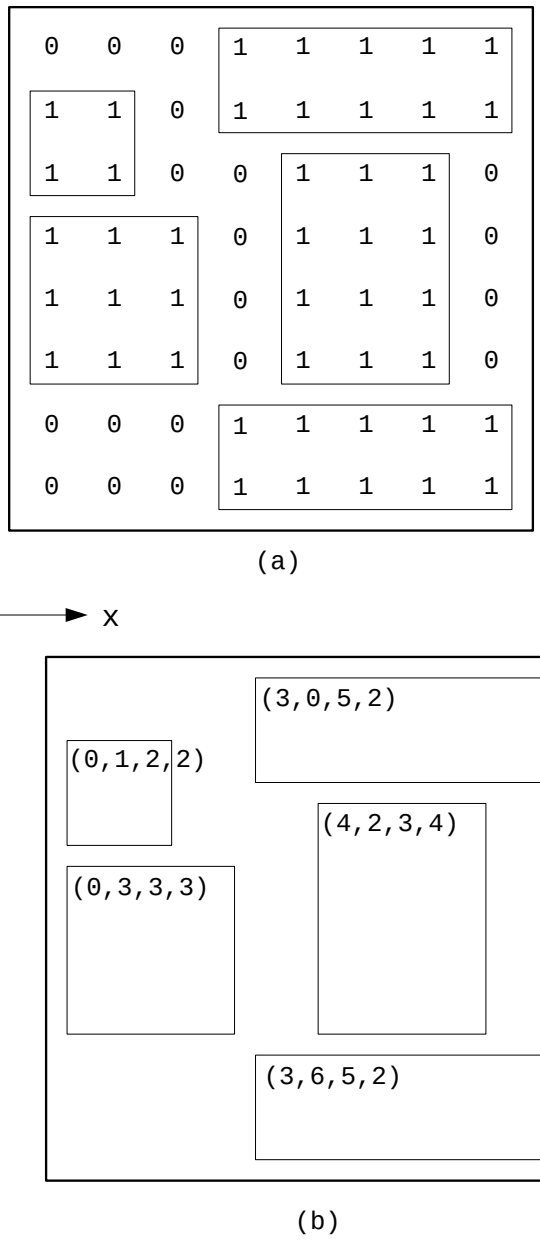


Figure 2: (a) Binary image rectangular partitioning. (b) Rectangle-based representation for the original image, where each rectangle is represented by the coordinates of the top left corner and its length and height.

tationally intensive and require parallel processing. For many applications, it is impossible to design a real-time image processing system if only a single processor is used. Instead of using a single processor, researchers try to enhance the computation power through multiprocessors. Recently, Lee and Horng *et al.* [5] derived a constant time parallel quadtree building algorithm for a given image based on a specified space-filling order.

The most important thing in rectangle-based algorithms is to extract the non-overlapping rectangles for the given image. In this paper, we will design an efficient parallel algorithm for building the rectangle-based representation of the given image. Based on the reconfigurability and power of the reconfigurable meshes, a constant time algorithm for this problem using $N \times N$ processors is derived. In the sense of the product of time and the number of processors used, our algorithm is time and cost optimal. To the best of our knowledge, there were no constant time algorithms before for this problem.

The rest of this paper is organized as follows. We give a brief introduction to the reconfigurable mesh in Section 2. Section 3 develops our parallel algorithm for rectangular partitioning. In Section 4, an example of rectangular partitioning application is addressed. We apply it to perform image compression and decompression. Finally, some concluding remarks are included in the last section.

## 2 Reconfigurable Mesh

The *reconfigurable mesh* [8] is used as the computational model throughout this paper. An $N \times N$ reconfigurable mesh consists of $N \times N$ identical processors arranged in a 2-D grid. The processor located at $(i, j)$, $0 \leq i, j < N$, is denoted by $P_{i,j}$. Each processor has four ports denoted by N, S, E and W. Any configuration of the bus system is derivable by properly establishing the local connection among ports within each processor. The interconnection among the four ports of a processor can be dynamically

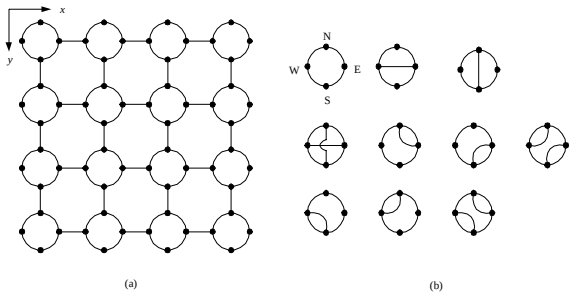


Figure 3: (a) A $4 \times 4$ reconfigurable mesh. (b) The allowed switch connection.

configured to suit computational needs during the execution of algorithms. It also allows us to reset (disconnect) the local connection. If all local connections are disconnected, the reconfigurable mesh is functionally equivalent to the mesh-connected computers.

We assume that each processor has a constant number of registers or local memory each of size $O(\log N)$ bit. We also assume that a processor can determine the first/last zero/nonzero bit in a register and perform standard arithmetic and Boolean operations in a unit of time. The reconfigurable mesh is assumed to be operated in a single instruction multiple data streams (SIMD) model. That is, for a time unit the same instruction is broadcast to all enabled processors, which execute it and wait for the next instruction. Each instruction can contain setting local connections, performing an operation mentioned above, broadcasting a value on a bus, or receiving a value from a specified bus. The regular structure of the reconfigurable mesh makes it suitable for VLSI implementation [6]. In fact, it is easy to see that the reconfigurable mesh can be used as a universal chip capable of simulating any equivalent architecture without the loss of time.

An example for a 2-D $4 \times 4$ reconfigurable mesh is shown in Figure 3(a). Some allowed local connections of a processor are shown in Figure 3(b). We assume that the setting of the local connection is destructive in the sense that setting a new pattern of connections de-

stroys the previous one. If there is no collision, it allows multiple processors to broadcast data on the buses simultaneously. According to other related models [2, 6, 8, 15], we assume that the communication time along a bus takes $O(1)$ time. The YUPPIE (Yorktown Ultra-Parallel Polymorphic Image Engine) [6], GCN (Gated-Connection Network) [14] and PPA (Polymorphic Processor Array) [7] multiprocessor systems indicate that this is a reasonable hypothesis practically. Due to these developments, the reconfigurable mesh is likely a feasible architecture for parallel processing.

The I/O loading time including download and upload is fully dependent on how complex the I/O interface between processors and peripherals will be. It is rather difficult to estimate accurately how much I/O time should be included to the time complexity of the algorithm. Therefore, we only sum up the computation time of processors and the communication time among processors as the time complexity of the proposed algorithm. This assumption has also been used by many researchers [2, 3, 6, 8, 15].

As for easily presenting our algorithms, let $var(i,\ j)$ and $ary(i,j)[k,l]$ denote the local variable $var$ (memory or register) and the local array $ary$ respectively in a processor with index $(i,\ j)$. For example, $sum(0,\ 1)$ and $rect(0,1)[k,l]$ are a local variable $sum$ and a local array $rect$ of processor $P_{0,\ 1}$.

## 3 Rectangular Partitioning

As mentioned above, there were several approaches proposed to partition the input image into a set of non-overlapping rectangles. Theoretically, an optimal partition algorithm must decompose the original image into the minimum number of non-overlapping rectangles [10]. Unfortunately, the computational complexity of optimal rectangular partitioning algorithm is significant and it is hard to implement it. Therefore, in practice, a simple and suboptimal algorithm is more valuable than an optimal one which is complex and hard to be implemented. Especially, it is important to design a fast suboptimal algorithm in the real time applications. This problem can be overcome by using parallel processing systems. The sequential algorithms proposed previously scanned the input image in a raster format and extracted the rectangles recursively. If we parallel these algorithms straightforwardly, it requires $\Omega(\log N)$ time. In this section, based on the reconfigurability and power of the reconfigurable mesh, a constant time and cost optimal algorithm for rectangular partitioning on a binary image can be derived. Given an $N \times N$ binary image $B$, assume each pixel of it can be either a black pixel or a white pixel. In the ensuing discussion, without loss of generality, it is assumed that initially the index and label of pixel $(i,j)$, $0 \leq i,j < N$, are identical and equal to $i \times N + j$. The parallel rectangular partitioning algorithm consists of three major steps.

1: For each row, connect the bus of the processor which contains black pixel relative to its previous processor (according to the column index order) having black pixel and disconnect it, otherwise. In this way, two processors with the black pixels will be connected together. After that, several sub-buses can be constructed which represent the corresponding rectangles with one unit of height on each row. Then, label each rectangle with the minimum index of the pixel to which it belongs. Using the largest column index and the smallest column index of each sub-bus, the length (or interval) of the newly extracting rectangle can be calculated.

2: The extracted rectangles then can be extended by merging the adjacent rectangles along the $y$-axis according to the following criterion: the adjacent rectangles have the same intervals, i.e. their corresponding sub-buses have the same $x$-coordinates. Then, relabel each newly extracting rectangle. The pixels with the same label will form a rectangle.

3: For each newly extracting rectangle, the pixel whose index is equal to its label is

located at the top left corner. Calculating the height of the newly extracting rectangle, the bottom right corner of each extracted rectangle can be found.

For the sake of simplicity, the input image $B$ is initially stored in the local variable $b(i,\ j)$ of processor $P_{i,\ j}$, $0 \leq i,\ j < N$. Let the data structure of a rectangle consist of four fields $x$, $y$, $l$ and $h$, respectively. That is, the coordinates of the top left corner of the rectangle are represented by $x$ and $y$, the length and height of the rectangle are represented by $l$ and $h$, respectively. The final result is stored in the local array $rect(i,\ j)[i, j, h, l]$ of processor $P_{i,\ j}$, $0 \leq i,\ j < N$. The detailed algorithm for the image rectangular partitioning (RECT) is presented as follows. From Figure 2, we know that the input image can be partitioned into five non-overlapping rectangles. Based on the image shown in Figure 2, where $N = 8$, an illustration of parallel rectangular partitioning algorithm is shown in Figure 4.

**Algorithm RECT**

INPUT: An $N \times N$ binary image. Download each binary image pixel $b(i, j)$, $0 \leq i, j < N$, to processor $P_{i,\ j}$, respectively.

OUPUT: Rectangles are stored in $rect(i,\ j)[i, j, h, l]$ of processor $P_{i,\ j}$, $0 \leq i,\ j < N$.

// Initially, assume that the label $lb(i, j)$ of pixel $b(i, j)$, $0 \leq i, j < N$, is equal to its index $id(i, j)$. //

**1:** // Partition and label the rectangles for each row. //

**1.1:** Processor $P_{i,\ j}$, $0 < i < N$, $0 \leq j < N$, establishes the local connection $WE$ if $b(i,\ j) = b(i-1, j) = 1$; disconnects the local connection, otherwise. After that, several sub-buses will be constructed on each row, and each sub-bus contains a piece of chained black pixels.

**1.2:** For each sub-bus, the leftmost processor broadcasts its label $lb$ and column index $x$ to the remaining processors; also the rightmost processor broadcasts its column index $x'$ to the remaining processors.

**1.3:** All processors on each sub-bus update its label by $lb$ received in Step 1.2 and calculate the interval $v = [x, x']$ and length $l = x' - x + 1$.

**2:** // Merge the adjacent rectangles along the $y$-axis. //

**2.1:** Processor $P_{i,\ j}$, $0 \leq i < N$, $0 < j < N$, establishes the local connection $NS$ if $b(i,\ j) = b(i, j-1) = 1$ and $v(i, j) = v(i, j-1)$; disconnects the local connection, otherwise. After that, several sub-buses will be constructed on each column. Each sub-bus contains a piece of chained black pixels with the same interval.

**2.2** For each sub-bus, the topmost processor broadcasts its label $lb$ and row index $y$ to the remaining processors; also the bottommost processor broadcasts its row index $y'$ to the remaining processors.

**2.3** All processors on each sub-bus update its label by $lb$ received in Step 2.2 and calculate the length $h = y' - y + 1$.

**3:** Processor $P_{i,\ j}$, $0 \leq i,\ j < N$, whose label and index are identical (i.e. $lb(i, j) = id(i, j)$), is the top left corner of an extracted rectangle. Set $rect(i, j) = (i, j, l, h)$.

**Theorem 1** *The reconfigurable mesh of size $N \times N$ extracts the non-overlapping rectangles of an $N \times N$ binary image in constant time.*

**Proof:** Step 1 identifies each piece of the continuous black pixels. Step 2 finds the possible block size of a rectangle for the continuous black pixels. In Step 2, continuous rectangles each with one unit of height and with the same interval are merged to form a new rectangle of

larger size along the $y$-axis. The newly extracted rectangle consists of continuous black pixels (i.e. no white pixels within them), since the intervals of the merged rectangles each with one unit of height are identical. After Step 1, the labels of continuous black pixels are updated except the leftmost corner of each piece of continuous black pixels. After Step 2, the labels of black pixels of the newly extracted rectangle are updated except the topmost piece of continuous black pixels of it. Therefore, the label of the newly extracted rectangle is equal to the label of the topmost piece of continuous black pixels of it, and this is equal to the index of the leftmost corner of the topmost piece of continuous black pixels. Hence, the top left corner of the extracted rectangle is located at processor $P(i,j)$ whose label is never updated.

It is clear that every step of the proposed algorithm takes only constant time. Thus the time complexity is $O(1)$. *Q.E.D.*

# 4 Application

In this section, we use the parallel rectangular partitioning algorithm RECT to describe an approach for compressing a given binary image into a file and recovering it using the corresponding decompression algorithm.

Let the data structure of a rectangle consist of four fields $x$, $y$, $l$ and $h$, respectively. The coordinates of the top left corner of the rectangle are represented by $x$ and $y$. The length and height of the rectangle are represented by $l$ and $h$, respectively.

**Algorithm COMPRESS:**

Using the algorithm RECT addressed in Section 3, we can create a rectangle-based representation for the input image shown in Figure 2(a) and store it into a file denoted as COMP. For example, as shown in Figure 4(d), the records stored in file COMP are (3,0,5,2), (0,1,2,2), (4,2,3,4), (0,3,3,3) and (3,6,5,2), respectively. A coordinate needs $logN$ bits. Since the length (height) is equal to $x' - x + 1$ ($y' - y + 1$), fields $l$ and $h$ require $log(N - x)$ and $log(N - y)$ bits, respectively, where $x'$ and $y'$ are the coordinates of the bottom right corner. Hence, the total number of bits required for file COMP is $r(2\log N + log(N - x) + log(N - y))$ bits, where $r$ is the number of rectangles.

**Algorithm DECOMPRESS:**
**Input:** Download the records of file COMP to the corresponding processors, respectively.
**Ouput:** The pixels of the original image in an $N \times N$ reconfigurable mesh.

**1:** // Restore the pixel color of the top left corner of each rectangle. //
Processor $P_{x,y}$, $0 \leq x, y < N$, sets $b(x,y) = 1$, if it contains record $rect(x,y)[x,y,l,h]$; $b(x,y) = 0$, otherwise.

**2:** // Establish the row buses. //
Set the local switch of processor $P_{i,j}$, $0 \leq i, j < N$, to connect processors $P_{i-1,j}$ and $P_{i+1,j}$ together, if $b(i,j) = 0$; reset it, otherwise. After that, several sub-buses can be constructed whose leftmost processor contains the top left corner of the corresponding rectangle on each row.

**3:** Processor $P_{x,y}$, $0 \leq x, y < N$, with $b(x,y) = 1$ writes $x$, $l$ and $h$ on E port. Then the data will be broadcast rightward and processor $P_{i,j}$, $0 \leq i, j < N$, reads the data bus from E port and calculates the distance $d(i,j)$ between itself and processor $P_{x,y}$. That is, $d(i,j) = i - x + 1$.

**4:** // Restore the pixel color of the top edge of each rectangle. //

Processor $P_{i,j}$, $0 \leq i, j < N$, sets $b(i,j) = 1$, if $d(i,j) \leq l$; $b(i,j) = 0$, otherwise.

**5:** // Establish the column buses. //
Set the local switch of processor $P_{i,j}$, $0 \leq i, j < N$, to connect processors $P_{i,j-1}$ and $P_{i,j+1}$ together, if $b(i,j) = 0$; reset it, otherwise. After that, similarly to Step 2, several sub-buses can be constructed on each column.

**6:** Processor $P_{x,y}$, $0 \leq x, y < N$, with $b(x,y) = 1$ writes $y$ and $h$ on S port. Then the data will be broadcast downward and processor $P_{i,j}$, $0 \leq i, j < N$, reads the data bus from S port and calculates the distance $d'(i,j)$ between itself and processor $P_{x,y}$. That is, $d'(i,j) = j - y + 1$.

**7:** // Restore the original image. //

Processor $P_{i,j}$, $0 \leq i, j < N$, sets $b(i,j) = 1$, if $d'(i,j) \leq h$; $b(i,j) = 0$, otherwise. Then, the original image is reconstructed completely.

It is clear that every step of the proposed algorithm takes only constant time. This leads to the following theorem.

**Theorem 2** *Given an $N \times N$ binary image, the image compression and decompression can be performed in constant time using rectangular partitioning technique on a reconfigurable mesh of size $N \times N$.*

## 5 Concluding Remarks

Due to its simplicity and regularity, a mesh-connected computer (MCC) is a well-known established multiprocessing system. Although the computation power is somewhat increased in the MCC, it is confronted with the tedious communication time for global communication. There are two drawbacks of the MCC: fixed architecture and long communication diameter. These two drawbacks can be overcome by equipping it with various types of bus systems. Recently, the reconfigurable networks have received much attention from researchers because they can overcome the drawbacks of the MCC.

There are many types of reconfigurable networks, the reconfigurable mesh not only solves the global communication problem but also provides the run time reconfigurability. Based on the reconfigurability and power of the reconfigurable mesh, we use $N \times N$ processors to design a constant time algorithm for constructing a rectangle-based representation for an $N \times N$ binary image. We also successfully use this algorithm to describe constant time algorithms for image compression and decompression. Compared to that of Lee and Horng *et al.* [5], our method is better since ours do not need any preprocessing time. In practice, the preprocessing time proposed in [5] requires $O(\log N)$ time.

## References


[1] M. Aoki, Rectangular region coding for binary image data compression, *Pattern Recognition*, vol. 11, pp. 297-312, 1979.

[2] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The power of reconfiguration, *Journal of Parallel and Distributed Computing*, vol. 13, pp. 139–153, 1991.

[3] G. H. Chen, B. F. Wang and H. Li, Deriving algorithms on reconfigurable networks based on function decomposition, *Theoretical Computer Science*, vol. 120, pp. 215-227, 1993.

[4] I. Gargantini, An efficient way to represent quadtrees, *Commun. ACM*, vol. 25, pp. 905-910, 1982.

[5] S. S. Lee, S. J. Horng, H. R. Tsai and S. S. Tsai, Building a quadtree and its applications on a reconfigurable mesh, *Pattern Recognition*, vol. 29, pp. 1571-1579, 1996.

[6] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Transactions on Computers*, vol. 38, pp. 1345–1351, 1989.

[7] M. Maresca, Polymorphic processor array, *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 490–506, 1993.

[8] R. Miller, V. P. Kumar, D. Reisis and Q. F. Stout, Parallel computations on reconfigurable meshes, *IEEE Transactions on Computers*, vol. 42, pp. 678–692, 1993.

[9] S. A. Mohamed and M. M. Fahmy, Binary image compression using efficient partitioning into rectangular regions, *IEEE Trans-*

*actions on Communications*, vo. 43, pp. 1888-1893, 1995.

[10] T. Ohtsuki, Minimum dissection of rectilinear regions, in *Proc. IEEE Int. Conf. Circuits and Systems*, pp. 1210-1213, 1982.

[11] M. A. Ouksel and A. Yaagoub, The interpolation-based binary tree and encoding of binary images, *CVGIP: Graphical Models and Image Processing*, vol. 54, pp. 75-81, 1992.

[12] H. Samet, *Applications of Spatial Data Structures,* Addison-Wesly, Reading, MA, 1990.

[13] I. M. Spiliotis and B. G. Mertzios, Real-time computation of statistical moments on binary images using block representation, in *Proc. 4th Int. Workshop on Time-Varying Image Processing and Moving Object Recognition*, pp. 27-34, 1993.

[14] D. B. Shu and J. G. Nash, The gated interconnection network for dynamic programming, in: Tewsburg, S K *et al.* eds., *Concurrent Computations*, Plenum Publishing, New York, 1998.

[15] B. F. Wang, G. H. Chen and F. C. Lin, Constant time sorting on a processor array with a reconfigurable bus system, *Information Processing Letters*, pp. 34, pp. 187-192, 1990.

[16] G. Zeng and N. Ahmed, A block coding technique for encoding sparse binary patterns, *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, pp. 778-780, 1989.

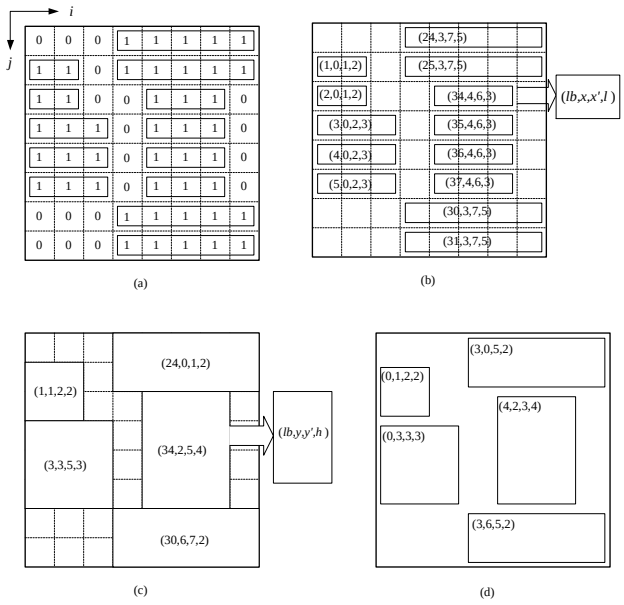


Figure 4: An illustration of algorithm RECT. (a) The row bus partitioning, after Step 1.1. (b) The value of $(lb, x, x', l)$ of each sub-bus, after Step 1.3. (c) The value of $(lb, y, y', h)$ of each sub-bus, after Step 2.3. (d) Each extracted rectangle is described by $(x, y, l, h)$ and stored in *rect* on the top left corner of the extracted rectangle, the final result.