Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Cost-Optimal Algorithm for the Channel-Assignment Problem on a Reconfigurable Parallel Processing System *

Horng-Ren Tsai† Shi-Jinn Horng† Shung-Shing Lee‡
Tzong-Wann Kao§ Chia-Ho Chen¶

†Department of Electrical Engineering, National Taiwan Institute of Technology, Taipei, Taiwan, R. O. C.
E-mail: horng@ntit64.ntit.edu.tw or horng@mouse.ee.ntit.edu.tw.
‡Department of Electronic Engineering, Fu Shin Institute of Technology and Commerce, I-Lain, Taiwan, R. O. C.
§Department of Electronic Engineering, Kuang Wu Institute of Technology and Commerce, Taipei, Taiwan, R. O. C.
¶Department of Electronic Engineering, Lee Ming Institute of Technology, Taipei, Taiwan, R. O. C.

Abstract

The computation model on which the algorithms are developed is the reconfigurable array of processors with wider bus networks (abbreviated to RAPWBN). The minor deference between the RAPWBN model with the other reconfigurable parallel processing systems is that the bus width of each bus network is extended to $N^{1/c}$-bit, where c is any constant and $c \geq 1$. Such a strategy is not only to make lots of improvement for saving the silicon area but also increase the system power enormously. In this paper, based on the wider bus network architecture, the channel-assignment problem for N pairs of components can be solved in $O(1)$ time using $2N$ processors with a $2N$-row by $2N$-column bus network, where the bus width is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$. Compared to the algorithms as proposed by Olariu et al. [13] and Lin [10], our algorithm runs in the same time complexity but reduces the number of processors to $O(N)$. Note that our algorithm is cost-optimal but others are not.

Key Words: *channel-assignment problem, minimum coloring problem, interval graph, list ranking, integer sorting, parallel algorithm, reconfigurable array of processors with wider bus networks.*

1 Introduction

Researchers have shown that the computation power of a single processor cannot be unlimitedly increased even with the advance of the hardware technique. Instead of designing the super processor, it is the best way to develop the parallel processing system to increase the power of a computer system. Because of its simplicity and regularity in architecture, the mesh-connected computer is one of famous parallel processing systems. With the advance of VLSI technique, it is quite suitable to be implemented by interconnection networks [2, 5]. Unfortunately, both fixed architecture and locality communication mechanism are two inherent drawbacks of the mesh-connected computer. Researchers overcome these two drawbacks by equipping it with a reconfigurable bus system.

A reconfigurable parallel processing system can be defined to be a set of processors connected to a reconfigurable bus system whose configuration can be dynamically established at run time. There are lots of varieties of this kind of machine including the reconfigurable meshes [12], the polymorphic torus architecture [9, 11], the processor array with a reconfigurable bus system [18] and the reconfigurable array of processors [6, 7]. Due to the reconfigurability of the bus system, many problems can be solved in constant time on such a machine. Based upon the proposed models, a VLSI chip, called YUPPIE (Yorktown Ultra Parallel Polymorphic Image Engine) [9, 11], has been implemented to prove these models being realistic.

The more processors are used in the system, the better executing time of an algorithm can be probably reached. In fact, the running time of an algorithm can be also improved by using the wider bus system instead of using the more processors. According to the experimental results, Li and Maresca [9, 11] have shown that by adding 20% silicon area over each processor, each processor has the ability to control the local switch between the processor and the system buses at the instruction level. This implies that it would be more efficient to save silicon area by increasing the bus capacity rather than by increasing the processor complexity. Based on such a fact, three improved models have been proposed. There are the

*The Work was supported by National Science Council under the contract no. NSC-85-2221-E011-004.

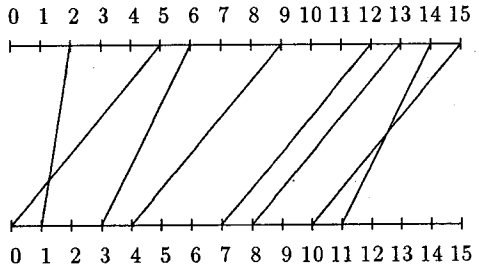

Figure 1: 8 pairs of components placed on the two sided circuit board.

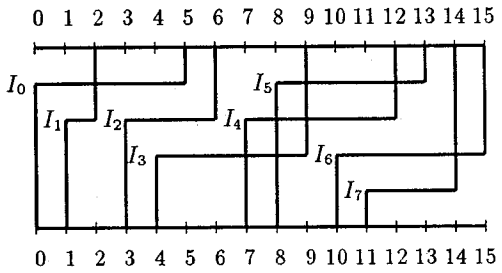

Figure 2: A channel-assignment corresponding to Figure 1.

reconfigurable multiple bus machine (abbreviated to RMBM) [17], the distributed memory bus computer (abbreviated to DMBC) [14] and the reconfigurable array of processors with wider bus networks (abbreviated to RAPWBN) [6, 8]. The minor difference between the RAPWBN model with the other two models is that the bus width of each bus network of the former is extended to $N^{1/c}$-bit, where c is any constant and $c \geq 1$.

The channel-assignment problem is one of the fundamental problems and has many practical applications in computer-aided design. The problem definition is defined in the following. Given a two-sided printed circuit board, there are horizontal lines called *channels* on one side and vertical lines on the other. There are N pairs of components, where each pair of components is to be placed on a specific vertical

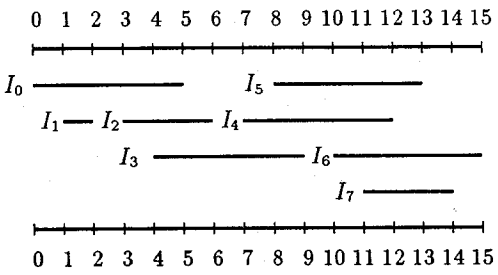

Figure 3: The corresponding intervals for the 8 pairs of components of Figure 1.

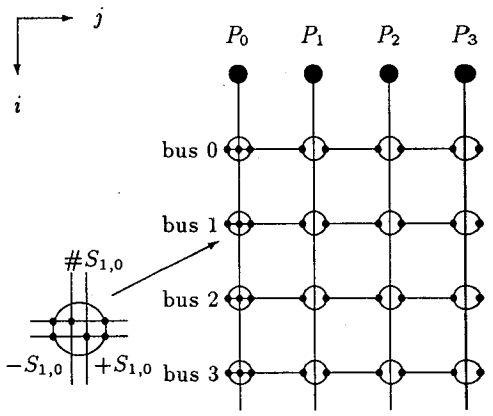

Figure 4: A linear RAPWBN of size 4 with a 4 by 4 bus network, where each bus network is 2 bits.

line, and each pair of components is to be connected by the horizontal line segment. Two pairs of components can be shared on the same channel if their connections do not conflict with each other. The channel-assignment problem is asked to minimize the total number of channels. An example for 8 pairs of components is shown in Figure 1. Figure 2 shows the channel-assignment of these 8 pairs of components corresponding to Figure 1. This problem also refers to the minimum coloring problem on interval graphs as follows. Let these N pairs of components be corresponding to N intervals on a real line, where both the left and the right endpoints of each interval are corresponding to the positions of a pair of components. Then, the minimum coloring problem is to assign a color to each interval such that the overlapping intervals have distinct colors. For example, Figure 3 shows the relationship between intervals and components corresponding to Figure 1.

The channel-assignment problem (or the minimum coloring problem on interval graphs) has been studied extensively by many researchers. Gupta et al. [4] gave an $O(N \log N)$ time sequential algorithm to solve this problem. The time complexity of their algorithm can be reduced to $O(N)$ if the intervals are sorted. Clearly, both sorted and unsorted cases are cost optimal. Dekel and Sahni [1] gave an $O(\log N)$ time parallel algorithm for this problem on the EREW PRAM model using $O(N^2)$ processors. With the different approach, Savage and Wloka [15], Yu et al. [19] and Yu and Yang [20] gave an $O(\log N)$ time parallel algorithm for this problem on the EREW PRAM model using $O(N)$ processors, respectively. The number of processors of their algorithms can be reduced to $O(N/\log N)$ if the intervals are sorted. Clearly, both sorted and unsorted cases are cost optimal. On the reconfigurable parallel processing system, Olariu et al. [13] gave an $O(1)$ time parallel algorithm for this problem on the reconfigurable meshes using N^2 processors. Lin [10] also gave an $O(1)$ time parallel algorithm for this problem on the processor arrays with reconfigurable bus systems using $O(N^2)$ processors.

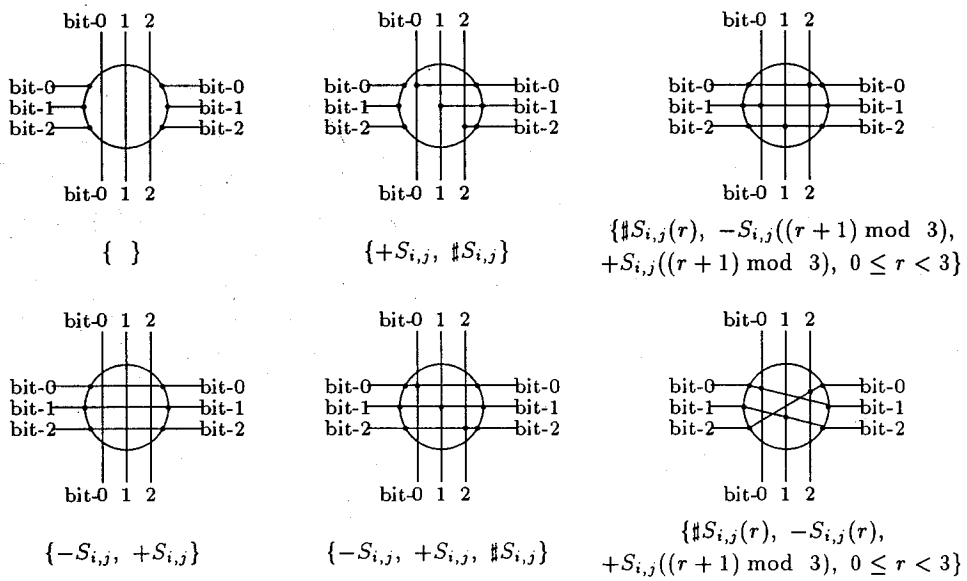

Figure 5: Six local switch configurations of an RAPWBN, where the bus width is 3 bits ($w = 3$).

In this paper we are interested in using the reconfigurable array of processors with wider bus networks [6, 8] to solve the channel-assignment problem. We review some basic operations including the list ranking algorithm which was proposed by Kao and Horng [6], and then derive an $O(1)$ time integer sorting algorithm. Based on these proposed basic operations, a constant time optimal algorithm for the channel-assignment problem is developed. This is the best result and first reported in the literature.

2 The Computation Model

A linear RAPWBN of size N contains N processors embedded in an M-row by N-column bus network. Each processor is identified by a unique index denoted as P_j, $0 \le j < N$, and the bus width of each bus network is usually assumed to be $N^{1/c}$-bit, where N is the number of processors and c is any constant for $c \ge 1$. For convenience, we assume $N^{1/c} = m$, where m is an integer. The M-row by N-column bus network has $2MN$ ports denoted by $-S_{i,j}, +S_{i,j}$ and each port has m-bit bus connection switches denoted by $-S_{i,j}(k), +S_{i,j}(k)$ for $0 \le i < M, 0 \le j < N$ and $0 \le k < m$. The i^{th}-row bus, $0 \le i < M$, connects the j^{th}-column port switch $+S_{i,j}$ to the $(j+1)^{th}$-column port switch $-S_{i,j+1}$ for $0 \le j < N-1$. Each processor P_j also has a column bus with M ports denoted by ♯$S_{i,j}$ and each port has m-bit bus connection switches denoted by ♯$S_{i,j}(k)$ for $0 \le i < M$ and $0 \le k < m$. The m-bit column bus of a processor can be connected to any row bus by setting the port connection switches ♯$S_{i,j}(k)$ to $-S_{i,j}(k)$ and/or $+S_{i,j}(k)$ for $0 \le i < M$, $0 \le j < N$ and $0 \le k < m$.

Any configuration of the bus system is derivable by properly establishing the local connection among the data bus of each port within each processor. To represent the local connection within each processor, we use the notations $\{g_0\}, \{g_1\}, \ldots, \{g_t\}$, where g_i, $0 \le i < t$, denotes a group of buses that are connected together. By setting the port connection $\{$♯$S_{i,0}(k)$, $-S_{i,0}(k)$, $+S_{i,0}(k)$, $0 \le i < 4$, $0 \le k < 2\}$, we show an example in Figure 4 for a linear RAPWBN of size 4 with a 4-row by 4-column bus network, where the bus width of each bus network is 2 bits. Six interesting switch configurations derivable from a processor of an RAPWBN are also shown in Figure 5. For simplicity, if each bit of the i^{th}-row bus is connected to the j^{th}-column bus one by one, we use $\{-S_{i,j}, +S_{i,j}, ♯S_{i,j}\}$ representation instead of using the representation $\{-S_{i,j}(k)$, $+S_{i,j}(k)$, ♯$S_{i,j}(k)$, $0 \le k < m\}$.

For a unit time, we assume each processor can perform one of the following operations: execute one arithmetic or logic operation, access a local memory word, set the local switches with the same connection configuration on the same column bus, broadcast a data on the established bus, and receive a data from the established bus. We allow multiple processors to broadcast data on the different buses or to broadcast the same data on the same bus simultaneously at a time unit, if there is no collision.

An RAPWBN is operated in an SIMD (single instruction stream, multiple data streams) model. The bus width is not unlimited between processors. For transferring an m-bit data between processors in constant time, we assume the bus width is bounded by m-bit as stated before, where m is an integer. The time complexity an algorithm is assumed to be the sum of the maximal computation time among all processors and the communication time among all processors. This assumption was also used by many researchers

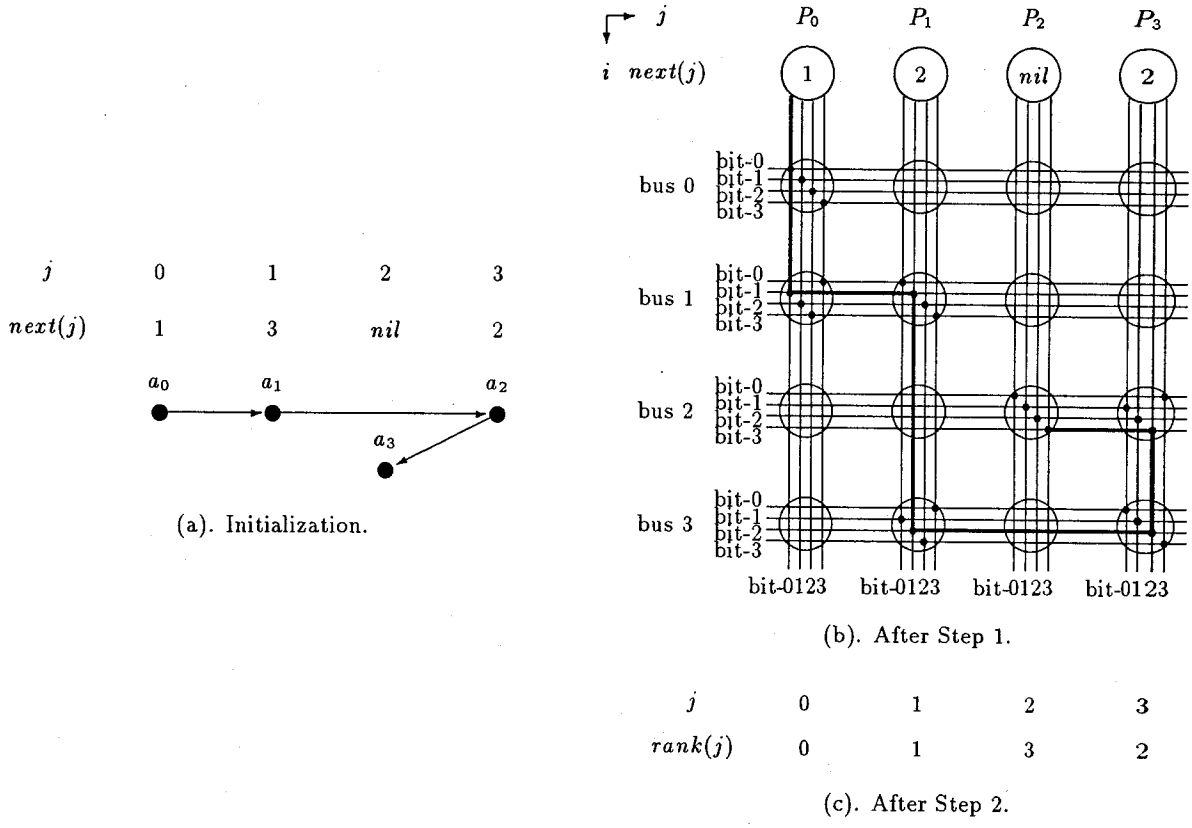

Figure 6: An illustration for ranking the linked list of size 4.

[6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 18].

3 Basic Operations

Lemma 1 *[7] Given a binary sequence b_j, $0 \leq j < N$, with $b_j \in \{0,1\}$, the prefix sum $\sum_{i=0}^{j} b_i$ can be computed in $O(1)$ time on a linear N RAPWBN with one row bus network, where the bus width of the bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$.* □

3.1 The List Ranking Problem

Assume there is a linked list a_0, a_1, ..., a_{N-1} with a_i following a_{i-1} in the list, and the definition of the list ranking problem is asked to find the rank for each element of the list. The list ranking problem discussed here is the *data dependent version*. Only the location of the first element is given along with a map from the i^{th} element to the $(i+1)^{th}$ element. Assume element a_i is contained in the j^{th} processor then element a_{i+1} will be contained in the $next(j)$ processor, where $next(j)$ is the link of element a_i. If $next(j) = nil$ then element a_i is the tail of the list and the rank of it is assumed to be $N-1$.

The main idea of the algorithm proposed by Kao and Horng [6] is to reduce the list ranking problem to the binary sequence prefix sum problem. It can be described by the following two steps. First, establish the local connection of the bus system on an RAPWBN according to the position of the next element. Then, compute the rank of each element in the linked list by performing the prefix sum along the established bus. Assume the head of the linked list is a_0. Initially, the linked list a_0, a_1, ..., a_{N-1} of N elements with a_{i+1} following a_i, a_i and a_{i+1} are stored in processors P_j and $P_{next(j)}$, respectively. That is, the link of element i is stored in the local variable $next(j)$ of processor P_j. Finally, the rank of element a_i is stored in the local variable $rank(j)$ of processor P_j. Let a_0 be stored in processor P_0. We simplify the list ranking algorithm proposed by Kao and Horng [6] in the following. Assume $next(0) = 1$, $next(1) = 3$, $next(2) = nil$ and $next(3) = 2$, and the bus width has 4 bits (i.e., $m = 4$). An illustration of computing the rank of each element of the linked list is shown in Figure 6.

Algorithm LRA;
Input: $next(j)$, $0 \leq j < N$.
output: $rank(j)$, $0 \leq j < N$.

0: begin

1: // Establish the local connection. //

Processor P_j, $0 \leq j < N$, establishes the local connection $\{\#S_{i,j}, -S_{i,j}, +S_{i,j},\ 0 \leq i < N\}$; then processor P_j with $next(j) \neq nil$, $0 \leq j < N$, establishes the local connection $\{\#S_{j,j}, -S_{j,j}, +S_{j,j}\}$ and $\{\#S_{next(j),j}(k), -S_{next(j),j}((k+1) \mod N), +S_{next(j),j}((k+1) \mod N),\ 0 \leq k < N\}$.

2: // Compute the rank of each element. //

2.1: Processor P_j with holding the data a_0, $0 \leq j < N$, writes a signal "1" to the bit 0 of the established bus on its port $\sharp S_{j,j}$.

2.2: Processor P_j, $0 \leq j < N$, sets $rank(j) = r$ if it can read the signal "1" from the bit r of the established bus on its port $\sharp S_{j,j}$.

3: **end**

Lemma 2 *[6] The list ranking problem can be solved in $O(1)$ time on a linear N* RAPWBN *with an N-row by N-column bus network, where the bus width of each bus network is N-bit.* □

Algorithm **LRA** can be easily modified to run in the case when the bus width is assumed to $N^{1/c}$-bit. The interested reader can refer to the literature [6] for details. From the result proposed by Kao and Horng [6], we have the following corollary.

Corollary 1 *[6] The list ranking problem can be solved in $O(1)$ time on a linear N* RAPWBN *with an N-row by N-column bus network, where the bus width of each bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$.* □

3.2 The Integer Sorting Problem

Given a data sequence $A = \{a_i\}$, $0 \leq i < N$, of N $O(\log N)$-bit integer numbers, the integer sorting problem is to rearrange these N numbers into ascending or descending order. Subbaraman et al. [17] have proposed an $O(1)$ time integer sorting algorithm on a linear $N^{1+1/c}$ RMBM with an N-row by N-column bus network, where c is any constant for $c \geq 1$, and each bus network is assumed to be $\log N$-bit. The main idea of the algorithm proposed by Subbaraman et al. [17] is to reduce the integer sorting problem to the list ranking problem. It can be described by the following four steps. First, link the numbers which have the same value into one linked list and then identify the head and the tail of each linked list. Next, link these linked lists to one linked list according to the value of the head and the tail. Then, apply the list ranking to rank these numbers on the linked list. Finally, each number is copied to the position according to its associated rank.

As stated by Subbaraman et al. [17], the time complexity of these four steps are analyzed as follows. On a linear RMBM, the first two and the last steps take $O(1)$ time using N processors with an N-row by N-column bus network, respectively. The third step takes $O(1)$ time using $N^{1+1/c}$ processors with an N-row by N-column bus network. Obviously, the total time complexity of the algorithm is bounded on the third step (i.e., list ranking algorithm). Therefore, based on the approach proposed by Subbaraman et al. [17] and the list ranking algorithm proposed by Kao and Horng [6], the integer sorting problem can be solved on a linear N RAPWBN with an N-row by N-column bus network, where the bus width of each bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$. This is an efficient approach to improve the integer sorting algorithm by increasing the bus capacity rather than by increasing the processor complexity. This leads to the following lemma.

Lemma 3 *Given N $O(\log N)$-bit integer numbers, these N numbers can be sorted in $O(1)$ time on a linear N* RAPWBN *with an N-row by N-column bus network, where the bus width of each bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$.* □

4 The Channel-Assignment Problem

Let $I = \{I_i\}$, $0 \leq i < N$, be a family of N intervals on a real line for an interval graph. Each interval I_i is represented by $[a_i, b_i]$, where a_i represents the left endpoint of interval I_i and b_i represents its right endpoint. Without loss of generality, we may assume that the endpoint a_i is smaller than that of b_i and these $2N$ endpoints all are distinct integers. The minimum coloring problem of an interval graph is defined as to assign a color to each interval such that the overlapping intervals share the distinct colors. Let these N intervals be corresponding to N pairs of components, where both the left and the right endpoints of each interval are corresponding to the position of a pair of components. Through such a transformation, the channel-assignment problem is equivalent to the minimum coloring problem of an interval graph.

For the channel-assignment problem, Gupta et al. [4] have proposed a popular algorithm to solve this problem. The main idea of the algorithm proposed by Gupta et al. [4] can be described as follows. Let these $2N$ endpoints of N intervals are sorted in ascending order. First, assume that all colors are available and a stack is created. Then the intervals are colored sequentially from the smallest left endpoint to the largest right endpoint. If the input interval is a left endpoint then assign a color to it and push it into the stack; otherwise, if the input interval is a right endpoint then release its color and pop it from the stack. Each released color can be reused for the next interval whose left endpoint is the nearest after it. Following this way, each interval will be assigned to its associated color after scanning these $2N$ endpoints. That is, all pairs of components will be assigned to their associated channels.

Based on the idea as proposed by Gupta et al. [4], this problem can be solved by the following four steps. Step 1, sort these $2N$ endpoints in ascending order. Step 2, determine the minimum number χ of colors which at least are required to color these N intervals and number these colors from 0 to $\chi - 1$. Step 3, determine the successive interval of each interval

which can share the same color provided that it exists. Thus, there are χ linked lists to be created. Step 4, find the head of each linked list and then broadcast its associated color number from it to the intervals along the linked list. In the following of this section, we will explain the detailed implementation of these four steps.

Let $c(j)$, $0 \leq j < 2N$, be the endpoints sequence of the left and right integer endpoints which are sorted in ascending order, where the endpoint a_i is smaller than that of b_i and these $2N$ endpoints all are distinct integers. For example, the sorted endpoints sequence of Figure 3 is as follow.

$$\{c(0),\ c(1),\ \ldots,\ c(14),\ c(15)\} = \{a_0,\ a_1,\ \ldots,\ b_7,\ b_6\}.$$

Without confusion, let each $c(j)$ represent not only the attribute of an interval but also its coordinate. For example, $c(14) = b_7$ represents the right endpoint of the interval I_7 with its coordinate being 14. Like Sprague and Kulkarni [16], we define a density sequence $d(0),\ d(1),\ \ldots,\ d(2N-1)$ as follows. For a fixed integer k, $0 \leq k < 2N$, $d(k)$ is the density of intervals at coordinate k to be the number of intervals which contain $k + \epsilon$, where $0 < \epsilon < 1$. For example, the density sequence of Figure 3 is as follows.

$$\{d(0),\ d(1),\ \ldots,\ d(14),\ d(15)\} = \{1,\ 2,\ \ldots,\ 1,\ 0\}.$$

Note that $|d(j) - d(j-1)| = 1$ for $1 \leq j < 2N$.

The density sequence $d(j)$, $0 \leq j < 2N$, can be obtained by first setting $w(j)$ as follows.

$$w(j) = \begin{cases} 1 & \text{if } c(j) \text{ is a left endpoint,} \\ -1 & \text{if } c(j) \text{ is a right endpoint.} \end{cases} \quad (1)$$

Then,

$$d(j) = \sum_{i=0}^{j} w(i). \quad (2)$$

Let χ be the minimum number of colors which are needed to color the family of intervals I. Thus,

$$\chi = \max\{d(j) | 0 \leq j < 2N\}. \quad (3)$$

Based on the density sequence $d(j)$, for each interval, define a successor sequence, denoted by $next(j)$, $0 \leq j < 2N$, as the nearest successive interval which can share the same color among all intervals. $next(j)$, $0 \leq j < 2N$, can be formularized as follows.

$$next(j) = \begin{cases} b_i & \text{if } c(j) = a_i; \\ \begin{cases} a_r & \text{if } c(j) = b_i \text{ and } a_r = \min\{a_k | d(a_k) = d(b_i) + 1,\ b_i < a_k < 2N\}; \\ nil & \text{otherwise.} \end{cases} \end{cases} \quad (4)$$

That is, if $c(j)$ is a left endpoint a_i of an interval I_i then the $next(j)$ is set to its right endpoint b_i. If $c(j)$ is a right endpoint b_i of an interval I_i then the $next(j)$ is set to the left endpoint a_k of interval I_k, where I_k is the nearest successive interval for $c(b_i) < c(a_k) < 2N$ and the density $d(a_k)$ of a_k is equal to the density $d(b_i) + 1$ of b_i; otherwise, if such a left endpoint a_k cannot be found by b_i then the $next(j)$ is set to nil. For example in Figure 3, $c(1)$ is a left endpoint a_1, $next(1)$ is therefore set to the coordinate 2 of the right endpoint b_1. Since $c(2)$ is a right endpoint b_1, both $d(a_2)$ and $d(a_4)$ are equal to $d(b_1) + 1$ for $c(b_1) < c(a_2) < c(a_4)$. a_2 is the nearer to b_1 of the two left endpoints a_2 and a_4, $next(2)$ is therefore set to the coordinate 3 of the left endpoint a_2 of the interval I_2. Since we cannot find a left endpoint as the successor of b_7, $next(14)$ is set to nil.

Based on $d(j)$, χ and $next(j)$ as defined above, we have the following theorem.

Theorem 1 *Given a family I of intervals with the successor sequence $next(j)$, there are χ linked lists, where each interval belongs to exactly one linked list and the intervals in the same linked list can share the same color.* □

From the successor function $next(j)$, the head of each linked list, denoted by $head(j)$, $0 \leq j < 2N$, is identified and marked by

$$head(j) = \begin{cases} 1 & \text{if } c(j) = a_i \text{ and } a_i \neq next(b_k), \\ & 0 \leq k < N, \\ 0 & \text{if } c(j) = b_i. \end{cases} \quad (5)$$

Hence, the color number of each head can be numbered by

$$color(j) = \begin{cases} (\sum_{k=0}^{j} head(k)) - 1 & \text{if } head(j) = 1, \\ nil & \text{otherwise,} \end{cases} \quad (6)$$

where $0 \leq j < 2N$.

Based on Equations (1)-(6) and Theorem 1, we will propose a constant time parallel algorithm for the channel-assignment problem on a linear $2N$ RAPWBN with a $2N$-row by $2N$-column bus network. Initially, the left and right endpoints of these N intervals a_j and b_j are stored in the local variables a_j and b_j of processor P_j, $0 \leq j < N$, respectively. Finally, the associated channel of ach interval I_j is stored in the local variable $asgn(j)$ of processor P_j, $0 \leq j < N$. The detailed channel-assignment algorithm (**CAA**) is shown in the following.

Algorithm CAA

Input: A family I of intervals $\{I_i \mid 0 \leq i < N\}$ of an interval graph, where $I_i = [a_i,\ b_i]$.

Output: The channel-assignment $asgn(j), 0 \leq j < N$.

0: begin

1: // Sort the left and right endpoints of N intervals into ascending order.//

1.1: Processor P_j, $0 \leq j < 2N$, establishes the local connections $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$ and $\{\sharp S_{(j \bmod N),j}, -S_{(j \bmod N),j}, +S_{(j \bmod N),j}\}$. Then processor P_j, $0 \leq j < N$, broadcasts b_j to processor P_{j+N-1} using the j^{th}-row bus network.

1.2: By Lemma 3, sort a_j and b_{j+N-1}, $0 \leq j < N$, into ascending order and store each endpoint a_α (or b_α) to the local variable $c(\alpha)$ of processor P_α for $c(0) \leq c(1) \leq \cdots \leq c(2N-1)$.

2. // Compute $d(j)$. //

2.1: Processor P_j, $0 \leq j < 2N$, sets $w(j)$ to 1 if it holds a left endpoint; sets $w(j)$ to -1, otherwise.

2.2: By Lemma 1, perform the prefix sum on $w(j)$ to obtain $d(j) = \sum_{i=0}^{j} w(i)$ for $0 \leq j < 2N$.

3. // Compute $next(j)$. //

3.1: Processor P_j, $0 \leq j < 2N$, establishes the local connection $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$, and then establishes the local connection $\{\sharp S_{i,j}, -S_{i,j}, +S_{i,j}\}$ if $c(j) = a_i$ or $c(j) = b_i$, $0 \leq i < N$. Then processor P_j with $c(j) = b_i$, $0 \leq j < 2N$, $0 \leq i < N$, broadcasts the coordinate of b_i on the i^{th}-row bus network from the port $\sharp S_{i,j}$.

3.2: Processor P_j with $c(j) = a_i$, $0 \leq j < 2N$, $0 \leq i < N$, reads the data from the port $\sharp S_{i,j}$ through the i^{th}-row bus and stores it into $next(j)$.

3.3: Processor P_j, $0 \leq j < 2N$, establishes the local connection $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$, then establishes the local connection $\{\sharp S_{r,j}, -S_{r,j}, r = d(j) - 1\}$ if $c(j) = a_i$, and then establishes the local connection $\{\sharp S_{r,j}, +S_{r,j}, r = d(j)\}$ if $c(j) = b_i$. Then, processor P_j with $c(j) = a_i$ and $r = d(j)-1$, $0 \leq j < 2N$, $0 \leq i < N$, broadcasts the coordinate of a_i on the established bus from the port $\sharp S_{r,j}$.

3.4: Processor P_j with $c(j) = b_i$ and $r = d(j)$, $0 \leq j < 2N$, $0 \leq i < N$, reads the data from the port $\sharp S_{r,j}$ through the r^{th}-row bus and stores it into $next(j)$.

4. // Compute $head(j)$. //

4.1: Processor P_j, $0 \leq j < 2N$, establishes the local connections $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$, then establishes the local connection $\{\sharp S_{j,j}, -S_{j,j}\}$ if $c(j) = a_i$, and then establishes the local connection $\{\sharp S_{next(j),j}, +S_{next(j),j}\}$ if $c(j) = b_i$ and $next(j) \neq nil$. Then, processor P_j with $c(j) = b_i$ and $next(j) \neq nil$, $0 \leq j < 2N$, $0 \leq i < N$, broadcasts a signal $*$ on the established bus from the port $\sharp S_{next(j),j}$.

4.2: Processor P_j with $c(j) = a_i$, $0 \leq j < 2N$, $0 \leq i < N$, sets $head(j)$ to 0 if it can read the signal $*$ from the port $\sharp S_{j,j}$ through the established bus; sets $head(j)$ to 1, otherwise.

4.3: Processor P_j, $0 \leq j < 2N$, set $head(j) = 0$ if $c(j) = b_i$, $0 \leq i < N$.

5. // Compute $color(j)$. //

By Lemma 1, perform the prefix sum on $head(j)$ to obtain $color(j) = (\sum_{i=0}^{j} head(i)) - 1$ for $0 \leq j < 2N$. Then, processor P_j, $0 \leq j < 2N$, sets $color(j)$ to nil if $head(j) = 0$.

6. // Assign the color to all intervals. //

6.1: Processor P_j, $0 \leq j < 2N$, establishes the local connections $\{-S_{i,j}, +S_{i,j}, 0 \leq i < 2N\}$ and $\{\#S_{j,j}, -S_{j,j}, +S_{j,j}\}$, and then establishes the local connection $\{\#S_{next(j),j}, -S_{next(j),j}, +S_{next(j),j}\}$ if $next(j) \neq nil$. Then, processor P_j with $c(j) = a_i$ and $head(j) = 1$, $0 \leq j < 2N$, $0 \leq i < N$, broadcasts $color(j)$ on the established bus from the port $\sharp S_{j,j}$.

6.2: Processor P_j with $c(j) = a_i$, $0 \leq j < 2N$, $0 \leq i < N$, reads the data from the port $\sharp S_{j,j}$ through the established bus and stores it into $asgn(j)$.

7. // Copy the assigned color number of each interval back to its corresponding position. //

7.1: Processor P_j establishes the local connection $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$ if $0 \leq j < 2N$, then establishes the local connection $\{\sharp S_{j,j}, -S_{j,j}, +S_{j,j}\}$ if $N \leq j < 2N$, and then establishes the local connection $\{\sharp S_{i,j}, -S_{i,j}, +S_{i,j}\}$ if $c(j) = a_i$, $0 \leq i < N$, $0 \leq j < N$. Then, processor P_j with $c(j) = a_i$, $0 \leq i < N$, $0 \leq j < N$, broadcasts $asgn(j)$ on the i^{th}-row bus from the port $\sharp S_{i,j}$.

7.2: Processor P_j, $N \leq j < 2N$, reads the data from the port $\sharp S_{j-N,j}$ through the $(j-N)^{th}$-row bus and stores it into $temp(j)$.

7.3: Processor P_j, $0 \leq j < 2N$, establishes the local connections $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$ and $\{\sharp S_{(j \bmod N),j}, -S_{(j \bmod N),j}, +S_{(j \bmod N),j}\}$. Then processor P_j, $N \leq j < 2N$, broadcasts $temp(j)$ back to $asgn(j\text{-}N)$ of processor $P_{j\text{-}N}$ asing $(j\text{-}N)^{th}$-row bus network.

7.4: Processor P_j establishes the local connection $\{-S_{i,j}, +S_{i,j}, 0 \leq i < N\}$ if $0 \leq j < 2N$, then establishes the local connection $\{\sharp S_{j,j}, -S_{j,j}, +S_{j,j}\}$ if $0 \leq j < N$, and

then establishes the local connection $\{\sharp S_{i,j}, -S_{i,j}, +S_{i,j}\}$ if $c(j) = a_i$, $0 \leq i < N$, $N \leq j < 2N$. Then, processor P_j with $c(j) = a_i$, $0 \leq i < N$, $N \leq j < 2N$, broadcasts $asgn(j)$ on the i^{th}-row bus from the port $\sharp S_{i,j}$.

7.5: Processor P_j, $0 \leq j < N$, reads the data from the port $\sharp S_{j,j}$ through the j^{th}-row bus and stores it into $asgn(j)$.

8. end

Theorem 2 *Given a family I of N intervals, the channel-assignment problem can be solved in $O(1)$ time on a linear $2N$* RAPWBN *with a $2N$-row by $2N$-column bus network, where the bus width of each bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$.* □

5 Concluding Remarks

As we can see, the silicon area of the reconfigurable array of processors with wider bus network can be saved much by increasing the bus capacity rather than increasing the processor complexity according to the result as shown by Li and Maresca [9, 11]. It is not only to make lots of improvement for saving the silicon area but also to increase the system power enormously. Based on the wider bus network architecture, the channel-assignment problem can be solved in $O(1)$ time on the reconfigurable array of processors with wider bus system using $2N$ processors and a $2N$-row by $2N$-column bus network, where the bus width of each bus network is $N^{1/c}$-bit ($N^{1/c} > \log N$) for any constant c and $c \geq 1$. Compared to the well-known algorithms as proposed by Olariu et al. [13] and Lin [10], our algorithm is the best in both time complexity and processor complexity.

References

[1] E. Dekel and S. Sahni, "Parallel Scheduling Algorithms," *Operat. Research*, **31**, 24-49 (1983).

[2] T. Y. Feng, "A Survey of Interconnection Networks," *IEEE Comput. Mag.*, 12-27 (1981).

[3] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, (1980).

[4] U. I. Gupta, D. T. Lee and J. Y. T. Leung, "An Optimal Solution for the Channel-Assignment Problem," *IEEE Trans. on Comput.*, 807-810 **C-28** (1979).

[5] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, New York: McGraw-Hill, (1984).

[6] T. W. Kao and S. J. Horng, "Computing List Ranking on a RAP with Wider Bus Networks," *Proc. of the Int. Conf. on Parallel and Distributed Systems*, 28-33 (1994).

[7] T. W. Kao, S. J. Horng, Y. L. Wang and H. R. Tsai, "Designing Efficient Parallel Algorithms on a CRAP," *IEEE Trans. on Parallel and Distributed Systems*, **6**, 554-559 (1995).

[8] S. S. Lee, S. J. Horng, T. W. Kao and H. R. Tsai, "Optimal Computing Hough Transform on a Reconfigurable Array of Processors with Wider Bus Networks" *Patt. Recogn.* **29**, 603-613 (1996).

[9] H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **11**, 233-243 (1989).

[10] S. S. Lin, "Constant-Time Algorithms for the Channel Assignment Problem on Processor Arrays with Reconfigurable Bus Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, **13**, 884-890 (1994).

[11] M. Maresca and H. Li, "Connection Autonomy in SIMD Computers: A VLSI Implementation," *Journal of Parallel and Distributed Computing*, **7**, 302-320 (1989).

[12] R. Miller, V. K. P. Kumar, D. Reisis and Q.F. Stout, "Meshes with Reconfigurable Buses," *Proc. of the MIT Conf. on Advanced Research in VLSI*, 163-178 (1988).

[13] S. Olariu, J. L. Schwing and J. Zhang, "A Constant-Time Channel-Assignment Algorithm on Reconfigurable Meshes," *BIT*, **32**, 586-597 (1993).

[14] S. Sahni, "Data Manipulation on the Distributed Memory Bus Computer," *Parallel Processing Lett.*, **5**, 3-14 (1995).

[15] J. E. Savage and M. G. Wloka, "A Parallel Algorithm for Channel Routing," *Lecture Notes in Computer Science*, **344**, 288-303 (1989).

[16] A. P. Sprague and K. H. Kulkarni, "Optimal Parallel Algorithms for Finding Cut Vertices and Bridges of Interval Graphs," *Information Processing Lett.*, **42**, 229-234 (1992).

[17] C. P. Subbaraman, J. L. Trahan and R. Vaidyanathan, "List Ranking and Graph Algorithms on the Reconfigurable Multiple Bus Machine," *Proceedings of the Int. Conf. on Parallel Processing*, **3**, 244–247 (1993).

[18] B. F. Wang, G. H. Chen and F. C. Lin, "Constant Time Sorting on a Processor Array with a Reconfigurable Bus System," *Information Processing Lett.*, **34**, 187-192 (1990).

[19] M. S. Yu, and C. L. Chen and R. C. T. Lee, "An Optimal Parallel Algorithm for Minimum Coloring of Intervals," *Proc. of the Int. Conf. on Parallel Processing*, **III**, 162-168 (1990).

[20] M. S. Yu and C. H. Yang, "A Simple Optimal Algorithm for the Minimum Coloring Problem on Interval Graphs," *Information Processing Lett.*, **48**, 48-51 (1993).