アルゴリズム 34－1
（1993．8．19）

再構成メッシュ上の最適な初期化アルゴリズム

中野浩嗣

(株) 日立製作所 基礎研究所
〒350-03 埼玉県比企郡鳩山町赤沼2520

本論文では $n \times n$ 個のプロセッサからなる再構成メッシュ上において、各プロセッサに x 座標と y 座標を与える最適な初期化アルゴリズムを示す。このアルゴリズムの計算時間は、バスの転送モデルが bit-transfer のとき $O(\log n)$ 時間、exclusive-word-transfer のとき $O(\log \log n)$ 時間、bitwise-word-transfer のとき $O(\log^* n)$ 時間である。

Optimal Initializing Algorithms for Reconfigurable Meshes

Koji Nakano

Advanced Research Laboratory, Hitachi, Ltd.
Hatoyama, Saitama 350-03, Japan
e-mail : nakano@harl.hitachi.co.jp

This paper presents optimal initializing algorithms for a reconfigurable mesh of $n \times n$ processors arranged in a 2-dimensional grid with a reconfigurable bus system. The algorithms assign x and y coordinates to each processor under the following constraints: $O(\log n)$ time for the bit-transfer model, $O(\log \log n)$ time for the exclusive-word-transfer model, and $O(\log^* n)$ time for the bitwise-word-transfer model.

1 Introduction

A reconfigurable bus system is a bus system whose configuration can be dynamically changed. *A reconfigurable mesh* (abbreviated as RM) is a processor array that consists of processors arranged in a 2-dimensional grid with a reconfigurable bus system (Fig. 1). Processors on an $n \times n$ RM are denoted by $\mathtt{PE}(i, j)$ $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$. As shown in Fig. 1, each processor has four ports denoted by `N`, `S`, `E` and `W`. The ports facing each other are connected by static busses. All processors work synchronously and execute the following phases in a unit of time:

Phase 1 Change configuration of a reconfigurable bus system by connecting or disconnecting its own ports by busses internally.

Phase 2 Send data to each port.

Phase 3 Receive data from each port. The data sent in the previous phase is received in this phase.

Phase 4 Execute a constant number of RAM instructions.

All processors execute the phases synchronously, that is, no processor executes a phase before all processors finish the previous phase. Efficient algorithms for a reconfigurable mesh have been investigated, including, for example, sorting [1, 4, 6], graph problems [1–3, 5] and so on.

However, every algorithm presented in the above papers was based on *an initialized reconfigurable*, that is, every processor on the mesh learns its x and y coordinates beforehand, although the assumption sometimes may not be explicit in the literature. More precisely, every algorithm is implemented on a reconfigurable mesh as follows: All processors execute the same program to complete the algorithm, but each can refer its own coordinates stored in reserved variables of the program, so the performance of processors differs according to their coordinates. In this paper, we assume an *uninitialized* reconfigurable mesh. That is, all processors are completely identical: they have the same program, constants, and variables and cannot refer to their coordinates.

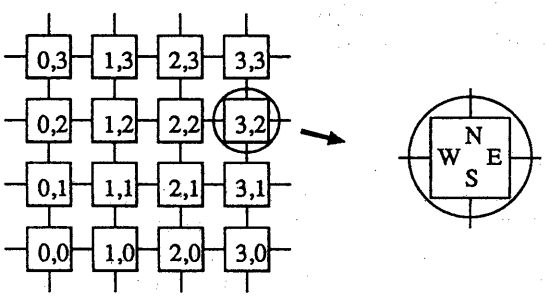

Figure 1: 4×4 reconfigurable mesh

If a reconfigurable mesh is implemented on material which may have faults, such as a WSI (Wafer Scale Integration), recovering fault processors becomes an issue to consider. In that case, extra processors must be used instead of fault processors, or a column which has fault processors must be bypassed, for example. In a fault environment, it is more desirable that every processor on a reconfigurable mesh be completely identical and a reconfigurable mesh be initialized when it is booted instead of when it is manufactured, because the coordinates of each processor may be changed. The objective of this paper is to show optimal *initializing algorithms* on an uninitialized reconfigurable mesh in which every processor executes the same program and learns its x coordinate and y coordinate.

In general, the computation times of algorithms on reconfigurable meshes differ with the communication capability of the bus system. This paper deals with three kinds of communication capabilities: *the bit-transfer model* (a value of a single bit can be transferred through a bus in a unit of time), *the exclusive-word-transfer model* (any value can be transferred through a bus in a unit of time), and *bitwise-word-transfer model* (any value can be transferred through a bus in a unit of time and the bitwise-or of the value is transferred in case of simultaneous sending to the same bus). We first show the lower bounds of the computation times for initializing reconfigurable meshes of the three models. Then, we present initializing algorithms on an $n \times n$ reconfigurable mesh for the following constraints: $O(\log n)$ time for the bit-transfer model, $O(\log \log n)$ time for the exclusive-word-transfer model, and $O(\log^* n)$ time for the bitwise-word-transfer model, where $\log^* n$ is the minimum integer k such that $\underbrace{\log \log \cdots \log}_{k\text{times}} n \leq 1$. These algorithms are optimal because their computation times are equal to the lower bounds.

2 Lower bounds

Let us start with the lower bounds of the computation time for initializing an RM.

For the bit-transfer model, it is easy to prove the following theorem:

Theorem 2.1 *Initializing an RM of the bit-transfer model requires at least* $\Omega(\log n)$ *time.*

Proof. For an initializing algorithm, let $S(t)$ be the number of states of processors at time t. On an RM of the bit-transfer model, a single bit value is transferred through a bus. Since each processor can receive from its four ports, it can get at most four bits of information from outside. Hence, the next states of two processors whose present states are identical may differ only if the four bits of information received by them are different. Thus, we

have

$$\begin{cases} S(0) &= 1, \\ S(t) &\leq S(t-1) \times 2^4 \quad (t \geq 1). \end{cases}$$

To solve the initializing problem in T time, all processors must fall into different states in T time. Hence, $S(T) \geq n^2$ holds. Therefore, $T = \Omega(\log n)$ holds. □

In order to fall into a different state, each processor must receive $\Omega(\log n)$ bits of information from its ports. From this fact, Theorem 2.1 can be understood intuitively.

For the word-transfer model, we have the following theorems:

Theorem 2.2 *Initializing an RM of the exclusive-word-transfer model requires at least* $\Omega(\log \log n)$ *time.*

Proof. $S(t)$ is defined in the same way as for Theorem 2.1. Since each processor may send different values to its four ports, $4 \cdot S(t)$ different values may be transferred through busses at time t. Hence, we have

$$\begin{cases} S(0) &= 1, \\ S(t) &\leq S(t-1) \times (4 \cdot S(t-1))^4 \quad (t \geq 1). \end{cases}$$

Since $S(T) \geq n^2$ must hold to solve the initializing problem in T time, $T = \Omega(\log \log n)$ holds. □

The crucial constraint of the bus system in the above proof is that every received value actually be sent. Therefore, this lower bound holds, even if one of the sent values is actually transferred in a case of simultaneous sending.

Theorem 2.3 *Initializing an RM of the bitwise-word-transfer model requires at least* $\Omega(\log^* n)$ *time.*

Proof. $S(t)$ is again defined, in the same way as for Theorem 2.1. Since each processor may send different values to its four ports, $4 \cdot S(t)$ different values may be sent to ports. Furthermore, since the bitwise-or of sent values is transferred in a case of simultaneous sending, at most $2^{4 \cdot S(t)}$ different values are transferred through busses at time t. Hence, we have

$$\begin{cases} S(0) &= 1, \\ S(t) &\leq S(t-1) \times (2^{4 \cdot S(t)})^4 \quad (t \geq 1). \end{cases}$$

Since $S(T) \geq n^2$ must hold to solve the initializing problem in T time, $T = \Omega(\log^* n)$ holds. □

The proofs of Theorems 2.2 and 2.3 are independent of the internal computation of processors. Thus, the theorems hold even if the processors have stronger internal instruction set.

3 Initializing for bit-transfer model

To clarify some concepts of initializing, this section starts with an inefficient but simple initializing algorithm on an RM of the bit-transfer model. Then, some basic algorithms are shown. Finally, we show an efficient initializing algorithm for the bit-transfer model whose computation time matches the lower bound.

3.1 Simple linear time initializing algorithm for bit-transfer model

At the first step, each processor learns whether it is on the boundary as follows: Every processor sends 1 to N and then tries to receive 1 from its S. A processor fails to receive 1 iff it is on the *north boundary* (i.e., PE(i, j) such that$j = n - 1$). Therefore, every processor can determine whether it is on the north boundary and, in similar process, whether it is on the *south boundary*, the *east boundary*, and/or the *west boundary*.

At the second step, every processor learns its x coordinate. Each PE$(0, j)$ (i.e., each processor on the west boundary) sends 1 to E, and every processor tries to receive it from its W. A processor succeeding in receiving 1 recognizes that it is PE$(1, j)$ for some j. Next, PE$(1, j)$ sends 1 to E, and every processor tries to receive it from its W. A processor succeeding in receiving 1 at this step recognizes that it is PE$(2, j)$ for some j. Repeating similarly towards the east, every processor recognizes its x coordinate. Each PE(i, j) receives 1 at the ith iteration. However, there is a problem: Each PE$(n-1, j)$ cannot notify the other processors that the repeating should end, that is, every processor keep trying to receive from its W. Thus, at each iteration every processor must check whether the repeating should end. To do this all processors connect ports *horizontally* (i.e. W · E) and each tries to receive from its W. Each PE$(n-1, j)$ sends 1 to the horizontal bus when it recognizes that it is PE$(n-1, j)$. As every processor succeeds in receiving, it terminates the iteration. Note that this check, the *background confirmation*, should be executed at every iteration.

After finishing the iteration, every processor learns its x coordinate. Then, substituting the vertical for the horizontal direction in the algorithm, every processor can also learns its y coordinate. As a result, using this simple algorithm, initializing an RM of the bit-transfer model can be performed in $O(n)$ time.

From this algorithm, we can informally state the following propositions on an RM of the bit-transfer model:

- Each processor can recognize whether and to which boundary it belongs in constant time.
- By using the background confirmation, if at least one processor recognizes the time when iteration must be terminated, each processor can terminate the iteration at that time.

From now on, we assume that each processor already knows which boundary it belongs to, by executing the first step of the above algorithm. Furthermore, for simple explanations, we will omit mention of the background confirmation in initializing algorithms when iteration is used.

3.2 Assigning quadrants

Determining which quadrants a processor belongs w.r.t. a given processor is fundamental for initializing a RM.

We say that a processor is *marked* if a single processor is specified and it knows that it is marked. For example, after each processor recognizes which boundary it belongs to, PE(0,0) (i.e., a processor both on the south boundary and on the west boundary) can be considered to be marked. That is, we can write a statement using a marked processor, say, "PE(0,0) sends 1 to W," on an initializing algorithm.

For a given marked processor, we will introduce relative coordinates of all processors and the *quadrants* w.r.t. the marked processor. Let the marked processor be PE(x, y). Then, PE$(x+i, y+j)$ is denoted as PE$[i, j]$, giving the relative position w.r.t. the marked processor. Note that, in general, each processor does not know its relative position and only the marked processor knows that it is PE[0,0]. For a given marked processor, we divide the RM into the quadrants as shows in Fig. 2.

It is not difficult to assign each processor its quadrant. The following algorithm determines whether each processor is on the x-axis and whether in the quadrant I. Determining whether each processor is on the other quadrants can be performed analogously.

[Assigning quadrants]

Step 1 All processors except PE[0,0] connect ports horizontally and try to receive from it. PE[0,0] sends 1 to E and 2 to W. Processors succeeding in receiving either 1 or 2 are on the x-axis.

Step 2 All processors except those on the x-axis connects ports *vertically* (i.e. N · S) and try to receive from it. Processors which received 1 in Step 1 send 1 to N. A processor succeeding in receiving 1 belongs to the quadrant I.

Furthermore, we define the *diagonal quadrants* w.r.t. a marked processor (Fig. 2) as quadrants rotated around the marked processor by 45 degrees.

It is not difficult to assign each processor its diagonal quadrant. The following algorithm determines whether each processor is on the /-diag, on the \-diag, and in the I-diag. Since determination of whether each processor is in the other diagonal quadrants, we will omit description of those processes.

[Assigning diagonal quadrants]

Step 1 Every processor except PE[0,0] connects ports /-*diagonally* (i.e., N · W and S · E). PE[0,0] sends 1 to N and 2 to S. Every processor tries to receive it from N and S. Processors which succeed in receiving 1 from N or 2 from S are on the /-diag.

Step 2 Every processor except PE[0,0] connects ports \-*diagonally* (i.e., N · E and S · W). PE[0,0] sends 1 to N and 2 to S. Every processor tries to receive from N and S. Processors which succeed in receiving 1 at N or 2 at S are on the \-diag.

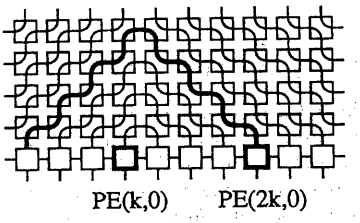

Figure 3: Marking double-distance processor

Step 3 Every processor except one either on the /-diag or on the \-diag connects ports vertically. Every processor which succeeded in receiving 1 either at Step 1 or at Step 2 tries to send 1 to N. Every processor tries to receive it from its S. Processors which succeed in receiving 1 are in the I-diag.

Each step can be performed in constant time on an RM of the bit-transfer model. Therefore, we have

Lemma 3.1 *For a given marked processor, the quadrants and the diagonal quadrants can be assigned in constant time on an RM of the bit-transfer model.*

3.3 Marking double and half-distance processors

Suppose that PE$(k, 0)$ is marked. Let PE$(2k, 0)$ and PE$(\lfloor k/2 \rfloor, 0)$ be *the double-distance processor* and *the half-distance processor* w.r.t. PE$(k, 0)$, respectively. This subsection shows algorithms which mark the double-distance and the half-distance processors with new marks for a given marked processor. Note that k is an unknown integer.

First, we show an algorithm for marking the double-distance processor.

[Marking the double-distance processor]

Step 1 Every processor recognizes its quadrant w.r.t. the marked processor.

Step 2 Every processor connect ports as shown in Fig. 3. After that, PE(0,0) sends 1 to N and each PE$(i, 0)$ with $k \geq i$ tries to receive it. Then, PE$(2k, 0)$, the double-distance processor, succeeds in receiving 1.

Next we will show the algorithm for marking the half-distance processor.

[Marking the half-distance processor]

Step 1 Every processor learns its quadrant w.r.t. PE(0,0) and w.r.t. the marked processor.

Step 2 Any processor at the intersection of the /-diag w.r.t. PE(0,0) and the \-diag w.r.t. the marked processor is in the same column as the half-distance

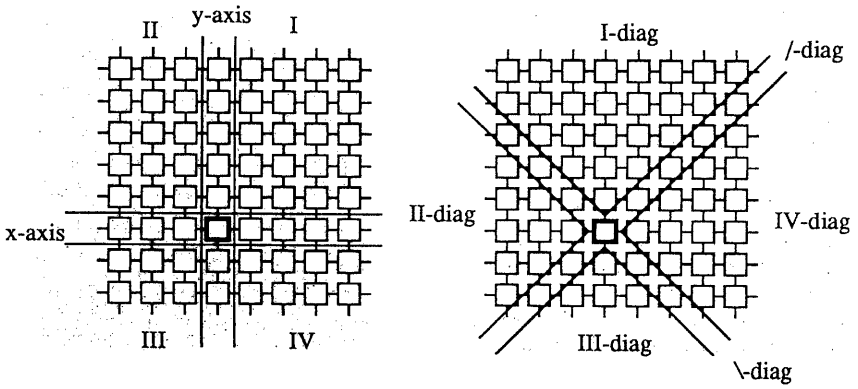

Figure 2: Quadrants and diagonal quadrants

processor. It notifies the half-distance processor by connecting ports vertically. In this case, k is odd. If there exists no intersection processor, k is even, and the following steps are executed.

Step 3 Every processor learns its quadrant w.r.t. PE$(k-1,0)$, the processor neighboring the marked processor on the west side.

Step 4 There exists a processor at the intersection of the /-diag w.r.t. the marked processor and to \-diag w.r.t. the neighbor processor. The intersection processor is in the same column as the half-distance processor. It notifies the half-distance processor, by connecting ports vertically.

From the above algorithms, we have

Lemma 3.2 *For a given marked processor, the double-distance and the half-distance processors can be determined in constant time on an RM of the bit-transfer model.*

3.4 Optimal initializing for bit-transfer model

Compared with the initializing algorithm for the word-transfer model, that for the bit-transfer model is fairly simple, because the lower communication capability of the bit-transfer model increases the initializing time. This subsection shows an optimal initializing algorithm for the bit-transfer model.

For each PE(i,j), let $i_p i_{p-1} \cdots i_1$ $(p = \lceil \log n \rceil)$ be the binary representation of i. We will design an initializing algorithm for the bit-transfer model which computes $i_p, i_{p-1}, \cdots, i_1$ step by step based on the divide-and-conquer method.

[Initializing for bit-transfer model]

Step 1 Determine PE$(2^p, 0)$ where p is the maximum integer such that $2^p < n$. This can be done by repeating the double-distance marking p times.

Step 2 Assign the quadrants w.r.t. PE$(2^p, 0)$. Each PE(i,j) can determine its i_p by the quadrant it belongs to.

Step 3 Consider that the RM is divided into 2 subRMs such that the size of one is $2^p \times n$ (i.e., PE(i,j)'s such that $i_p = 0$), and the other's size is $(n-2^p) \times n$ (i.e., PE(i,j)'s such that $i_p = 1$). Determine PE$(2^{p-1}, 0)$ for the first each subRMand PE$(2^p + 2^{p-1}, 0)$ (if it exists) for the second. This can be done using the generalized version of the half-distance processor marking.

Step 4 Assign the quadrants w.r.t. PE$(2^{p-1}, 0)$ on the first subRM and w.r.t. PE$(2^p + 2^{p-1}, 0)$ on the second subRM. Each PE(i,j) can determine its i_{p-1} by the quadrant it belongs to.

Step 5 Continuing analogously, $i_{p-2}, i_{p-2}, \ldots, i_1$ can be determined.

Step 1 takes $O(p)$ time. After that, determining i_k takes a constant time for each $k = p, p-1, \ldots 1$. Furthermore, the y coordinates can be determined similarly. Since $p = \lceil \log n \rceil$, we have

Theorem 3.3 *Initializing an RM of the bit-transfer model can be performed in $O(\log n)$ time.*

From Theorem 2.1, the above algorithm is optimal.

4 Initializing for word-transfer model

The initializing algorithm for the word-transfer model that we will present is fairly complicated, because its computation time is sublogarithmic and some bits of the coordinates must be simultaneously computed. It is not possible to attain the sublogarithmic time by the divide-and-conquer method used in the bit-transfer model. Let us start with subalgorithms.

From now on, without loss of generality, we assume that n is even. As shown in the algorithm which determines the half-distance processor, it is easy to determine whether n is even. If n is odd, by ignoring processors on the north boundary and on the east boundary, the algorithms designed for even n can be executed similarly.

4.1 Marking even columns

This subsection shows an algorithm for marking columns; that is, each PE(i, j) learns whether i is even or not. In the algorithm, processors connect ports such that the busses form two spirals (Fig. 4). Each spiral is shifted by two with each circuit. Then, by sending 1 to one of the spirals, 1 is transferred alternately.

[Marking even columns]

Step 1 Determine PE$(n/2, 0)$ and PE$(0, n/2)$ by marking the half-distance processor.

Step 2 Divide the RM into 4 subRMs of $n/2 \times n/2$ by assigning the quadrants w.r.t. PE$(n/2, 0)$ and PE$(0, n/2)$. After that, each processor knows which subRM it belongs to. Also, assign the quadrants w.r.t. PE$(0, n/2 - 2)$.

Step 3 Connect ports as shown in Fig. 4. Then, PE$(0, n/2)$ sends 1 to E. Each PE$(i, n/2)$ $(0 \leq i < n/2)$ tries to receive it from S. A processor which succeeds in receiving 1 is in an even column. Therefore, each processor succeeding in receiving 1 notifies processors on the same column that they are in an even column. After that, each PE(i, j) $(0 \leq i < n/2, 0 \leq j < n)$ knows whether its i is even.

Step 4 Analogously, each PE(i, j) $(n/2 \leq i < n, 0 \leq j < n)$ can recognize whether i is even.

Since each step of the above algorithm takes a constant time, we have

Theorem 4.1 *Marking even columns on an RM of the bit-transfer model can be performed in constant time.*

4.2 Marking power-of-two columns

This subsection shows an algorithm which marks $2^{0\text{th}}$, $2^{1\text{st}}$, $2^{2\text{nd}}$, ..., $2^{p\text{th}}$ columns, where p is the maximum integer satisfying $2^p < n$. That is, after marking power-of-two columns, each PE(i, j) learns whether there exists an integer k satisfying $i = 2^k$. As shown in the initialization for the bit-transfer model, by repeating the algorithm which marks the double-distance processor, it is easy to mark power-of-two columns in $O(\log n)$ time. However, this section shows a constant-time algorithm for marking power-of-two columns.

[Marking power-of-two columns]

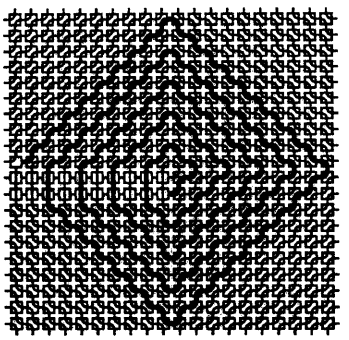

Figure 4: Marking even columns

Step 1 Divide the RM into 4 subRMs, just as for marking even columns, and assign diagonal quadrants w.r.t. PE$(0, 0)$.

Step 2 Connect ports as shown in Fig. 5. Then PE$(n/2-1, n/2)$ sends 1 to N. Each PE$(i, n/2-1)$ $(n/2 \leq i < n)$ tries to receive it from its N. Then, PE$(n/2-1+2^0, n/2-1)$, PE$(n/2-1+2^1, n/2-1)$, PE$(n/2-1+2^2, n/2-1)$, ..., PE$(n/2-1+2^{p-1}, n/2-1)$ succeed in receiving 1.

Step 3 Every processor connects ports /-diagonally, and PE$(n/2-1+2^0, n/2-1)$, PE$(n/2-1+2^1, n/2-1)$, PE$(n/2-1+2^2, n/2-1)$, ..., PE$(n/2-1+2^{p-1}, n/2-1)$, sends 1 to S. Each PE$(i, 0)$ tries to receive from its S. Only PE$(2^0, 0)$, PE$(2^1, 0)$, PE$(2^2, 0)$, ..., PE$(2^{p-1}, 0)$ succeed in receiving 1.

Step 4 Determine PE$(2^p, 0)$ by marking the double-distance processor of PE$(2^{p-1}, 0)$.

Step 5 By connecting ports vertically, PE$(2^0, 0)$, PE$(2^1, 0)$, PE$(2^2, 0)$, ..., PE$(2^p, 0)$ notify the processors in the respective columns that they are in power-of-two columns.

Since each step takes constant time, we have

Lemma 4.2 *Marking power-of-two columns can be performed in constant time on a bit-transfer model RM.*

4.3 Computing remainders

This subsection shows an algorithm for *computing remainders*: For a given marked processor PE$(0, 2^k)$, the computing remainders algorithm determines whether i mod $2^{k+1} < 2^k$ for each ith column on an RM. That is, each PE(i, j) recognizes whether i mod $2^{k+1} < 2^k$ (or whether $i_{k+1} = 0$). This algorithm is used for computing each digit of the binary representation of coordinates.

[Computing Remainders]

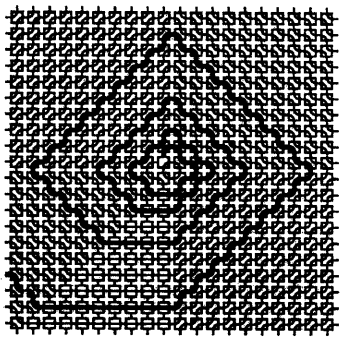

Figure 5: Marking power-of-two columns

Step 1 Determine whether each processor is on an even column by marking even.

Step 2 Mark PE$(0, 2^k - 1)$ by marking the half-distance processor.

Step 3 By assigning quadrants w.r.t. PE$(0, 2^k - 1)$, determine whether $j < 2^k$ for each PE(i, j).

Step 4 Each processor connects ports as shown in Fig. 6. That is, each PE(i, j) connects ports as follows:

S·E if i is even and $j = 2^k$,

S·W if i is odd and $j = 2^k$,

N·W if i is even and $j = 0$,

N·E if i is odd and $j = 0$,

/-diagonally if i is even and $0 < j < 2^k$,

N·S,W·E if i is odd and $0 < j < 2^k$.

Then, PE$(0, 0)$ sends 1 to W. Every processor on even columns tries to receive it from W. Since PE(i, j) ($i = 0, 2, 4, 6, \ldots$ and $0 \leq j < 2^{k-1}$) succeeds in receiving iff $i \bmod 2^{k+1} < 2^k$, it can be determined whether $i \bmod 2^{k+1} < 2^k$.

Step 5 Each PE(i, j) which succeeded in receiving 1 in Step 4 notifies processors in the same column whether $i \bmod 2^{k+1} < 2^k$. Furthermore, each PE(i, j) notifies it to its odd-column neighbor, PE$(i + 1, j)$.

Theorem 4.3 *Computing remainders can be performed in constant time on a bit-transfer model RM.*

Note that the above algorithm occupies $2^k + 1$ columns to compute remainders.

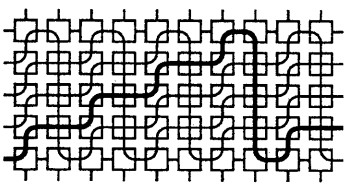

Figure 6: Computing remainder

4.4 Optimal initializing for the word-transfer models

Now, we will show an initializing algorithm for an RM of the word-transfer models. The algorithms for the exclusive-word-transfer and the bitwise-word-transfer are almost the same. The difference is how they implement Step 6.

Let $i_p i_{p-1} \cdots i_1$ ($p = \lceil \log n \rceil$) be the binary representation of i. The following algorithm computes $i_p i_{p-1} \cdots i_1$ for each ith column.

[Initializing for word-transfer models]

Step 1 Compute i_1 in each ith column by marking even columns.

Step 2 Mark PE$(0, 2^0)$, PE$(0, 2^1)$, PE$(0, 2^2)$, ..., and PE$(0, 2^{p-1})$ by marking power-of-two columns.

Step 3 Divide the RM into subRMs of size $n \times 2^2$, $n \times 2^3$, ..., $n \times 2^{p-2}$ (Fig. 7). This can be done by assigning quadrants w.r.t. PE$(0, 2^2)$, PE$(0, 2^3)$, ..., and PE$(0, 2^{p-1})$ simultaneously.

Step 4 In each ith column of each $2^{k+1} \times n$ subRM ($1 \leq k \leq p - 3$), by computing remainders, determine whether $i \bmod 2^{k+1} \leq 2^k$. The equation $i_{k+1} = 0$ holds iff $i \bmod 2^{k+1} \leq 2^k$. After that, processors in each ith column of each $2^{k+1} \times n$ subRM knows i_{k+1}.

Step 5 Compute i_p and i_{p-1} as in the initializing algorithm for the bit-transfer model.

Step 6 In each ith column, compute i from $i_p, i_{p-1}, \ldots, i_1$.

Steps 1, 2, 3, and 4 can be performed in constant time. Step 5 can be performed in constant time, because PE$(0, 2^{p-1})$ is already marked. Step 6 needs detailed explanation as follows.

After Step 5, on each ith column, the value i_k ($1 \leq k \leq p$) is known by at least one processor. Hence, for simplicity, assume that there exists $0 \leq j_1 < j_2 < \cdots < j_p < n$ such that each PE(i, j_k) ($1 \leq k \leq p$) knows i_k after Step 5 without loss of generality. First, construct an RM using all processors PE(i, j_k) such that $0 \leq i \leq n - 1$ and $1 \leq k \leq p$. In other words, all processors except PE(i, j_k)'s fix busses vertically. After that, it can be assumed that

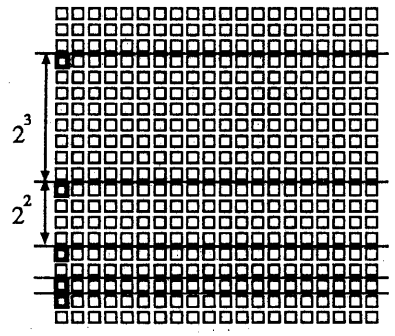

Figure 7: Division of reocnfigurable mesh

the new RM has p rows. Then, by recursively computing the y coordinate of the new RM with p rows, the integer k can be identified by PE(i, j_k).

Secondly, in the bitwise-word-transfer model, every processor connects ports vertically and each PE(i, j_k) sends $i_k \cdot 2^{k-1}$ to the bus. Then, since $i = i_p \cdot 2^{p-1} + i_{p-1} \cdot 2^{p-2} + \cdots + i_1 \times 2^0$ is transferred in each ith column, processors in the ith column can recognize i. In the exclusive-word-transfer model, i can be computed by the *binary tree method* as follows:

1. Compute $i_1 + i_2 \cdot 2$, $i_3 + i_4 \cdot 2$, $i_5 + i_6 \cdot 2$, ...
2. Compute $i_1 + i_2 \cdot 2 + (i_3 + i_4 \cdot 2) \cdot 2$, $i_5 + i_6 \cdot 2 + (i_7 + i_8 \cdot 2) \cdot 2$, ...
3. Compute $i_1 + i_2 \cdot 2 + (i_3 + i_4 \cdot 2) \cdot 2 + (i_5 + i_6 \cdot 2 + (i_7 + i_8 \cdot 2) \cdot 2) \cdot 2$, ...
4. Repeating in the same manner.

Since each PE(i, j_k) knows i_k and k, the binary tree method can be performed through vertical communication. By repeating the operation at most $\lceil \log p \rceil$ times, $i = i_1 + i_2 \cdot 2^1 + i_3 \cdot 2^2 + \cdots$ can be computed in $O(\log p) = O(\log \log n)$ time.

Consequently, we have

Theorem 4.4 *Initializing an RM of the exclusive-word-transfer model can be performed in $O(\log \log n)$ time.*

Proof. Let us estimate the computation time. Let $T(n)$ be the computation time of this algorithm. In the algorithm for the exclusive-word-transfer model, the algorithm for p columns is recursively executed. Hence, we have the following relation:

$$\begin{cases} T(1) &= O(1), \\ T(n) &= T(\lceil \log n \rceil) + O(\log \log n) \quad (n > 1). \end{cases}$$

Therefore, $T(n) = O(\log \log n)$. This completes the proof. □

Theorem 4.5 *Initializing an RM of the bitwise-word-transfer model can be performed in $O(\log^* n)$ time.*

Proof. Analogously to the proof of Theorem 4.4, let $T(n)$ be the computation time. Then, we have the following relation:

$$\begin{cases} T(1) &= O(1), \\ T(n) &= T(\lceil \log n \rceil) + O(1) \quad (n > 1). \end{cases}$$

Hence, $T(n) = O(\log^* n)$. This completes the proof. □

From Theorems 2.2 and 2.3, the algorithms are optimal.

5 Conclusions

This paper have shown optimal initializing algorithms for three kinds of reconfigurable meshes. It remains for researchers to create optimal initializing algorithms for three or more-dimensional reconfigurable meshes and for general rectangular reconfigurable meshes.

Acknowledgments

The author would like to thank Atsuko Yamaguchi for helpful comments.

References

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

[2] R. Lin. Fast algorithms for lowest common ancestors on a processor array with reconfigurable buses. *Information Processing Letters*, 40(4), November 1991.

[3] R. Miller, V. K. P. Kumar, D. Reisis, and Q. F. Stout. Meshes with reconfigurable buses. *Proc. of 15th MIT Conference on Advanced Research in VLSI*, pages 163–178, March 1988.

[4] K. Nakano, T. Masuzawa, and N. Tokura. A sub-logarithmic time sorting algorithm on a reconfigurable array. *IEICE Transactions*, E-74(11):3894–3901, November 1991.

[5] B. F. Wang and G. H. Chen. Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Trans. on Parallel and Distributed Systems*, 1(4):500–507, October 1990.

[6] 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*, 34(4):187–192, April 1990.