International Journal of Principles and Applications of Information Science and Technology
December 2007, Vol.1, No.1

Near Optimal Randomized Initialization on the 1-Dimensional Reconfigurable Mesh

Koji Nakano[1]

School of Engineering, Hiroshima University
Kagamiyama, Higashi-Hiroshima 739-8527, JAPAN
nakano@cs.hiroshima-u.ac.jp

Abstract. The reconfigurable mesh is a processor array that consists processors arranged in 1-dimensional or 2-dimensional grids with a reconfigurable bus system. We assume that processors are identical and have no unique IDs. Initialization is a task that assigns sequential unique IDs to processors in the reconfigurable mesh. The main contribution of this paper is to show initialization algorithms on the 1-dimensional reconfigurable mesh with n processors. We first show a simple deterministic initialization algorithm for the 1-dimensional reconfigurable mesh that runs in $O(n)$ time. This deterministic algorithm is optimal, because no deterministic solution can perform initialization in less than $O(n)$ time. Quite surprisingly, we show that expected sublinear-time initialization is possible if we use randomized techniques. Our initialization algorithm runs in $O((\log n + \log f) \log\log n)$ time with probability at least $1 - \frac{1}{f}$ for every real number $f > 1$. It follows that the initialization algorithm runs in expected $O(\log n \log\log n)$ time. We also proved that any randomized initialization need to run in $\Omega(\log n)$ time. Thus, our randomized initialization algorithm running in $O(\log n \log\log n)$ time is very close to a theoretical lower bound $\Omega(\log n)$ time.
Keywords: Parallel Algorithms, Reconfigurable Mesh, Reconfigurable Computing, Randomized Algorithms

1 Introduction

A *reconfigurable mesh* is a processor array that consists of processors arranged in 1-dimensional or 2-dimensional grids with a reconfigurable bus system. The reconfigurable mesh of size $m \times n$ consists of mn processors arranged in a 2-dimensional grid. Also, a 1-dimensional reconfigurable mesh of size m is the reconfigurable mesh with a single row. Let PE(j) ($0 \leq j \leq m-1$) denote a processor at position j. Although all processors execute the same instructions, their behaviors may differ, because they work on different input and different coordinates. On the reconfigurable mesh, any two adjacent processors are connected with a single fixed link. Each processor on the reconfigurable mesh has two ports E(*East*) and W(*West*). Thus, the port E of PE(j) and the port W of PE($j+1$) is connected by a link.

The connected components formed by adjacent fixed links and internal connections constitute *subbuses*, through which the processors can communicate.

Data sent to a port can be transferred through subbuses in one unit of time. Figure 1 illustrates an example of subbuses on the 1-dimentional reconfigurable mesh. The data sent to port E of PE(0) in the 1-dimensional reconfigurable mesh is transferred through subbus A and it can be received by processors $\mathrm{PE}(1), \mathrm{PE}(2), \mathrm{PE}(3)$, and $\mathrm{PE}(4)$. Note that subbus A can transfer any data sent by processors $\mathrm{PE}(0), \mathrm{PE}(1), \ldots,$ $\mathrm{PE}(4)$. Similarly, subbus B can transfer any data sent by processors $\mathrm{PE}(4), \mathrm{PE}(5), \mathrm{PE}(6)$, and $\mathrm{PE}(7)$. In this paper, we also assume that the subbus is *exclusive*, that is, no two processors can send the same subbus in the same time.

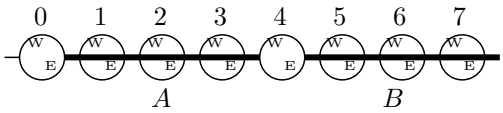

Fig. 1. An example of subbuses

The reconfigurable mesh also can be classified by the capacity of the subbuses as follows: *bit model*: The subbus can transfer 1-bit of data in a unit of time. It is not allowed to send two or more processors to the same subbus in the same time. *word model*: The subbus can transfer a word (usually, $\log n$-bit[1]) of data in a unit of time. It is not allowed to send two or more processors to the same subbus in the same time. *bitwise model*: The subbus can transfer a word (usually, $\log n$-bit) of data in a unit of time. If two or more data sent to the subbus, the bitwise OR of the data is transfered.

The reconfigurable mesh have attracted considerable attention as theoretical models of parallel computation, and many studies have been devoted to developing efficient parallel algorithms on reconfigurable meshes. For example, they efficiently solve problems such as sorting [11], selection [2], arithmetic operations [10], graph problems [3], geometric problems [7, 8], image processing [1]. See [12] for the comprehensive survey.

However, every algorithm presented in the above papers was based on *an initialized reconfigurable*, that is, every processor on the mesh know its x and y coordinates, 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. In other words, all processors are completely identical: they have the same program, constants, and variables and cannot refer to their coordinates. *Initialization* is a task that gives the correct coordinates to the processors in the reconfigurable mesh. The

[1] Let log and ln denote the logarithms to the base e and 2, respectively.

readers should refer to Figure 2 that illustrates the task of initialization on the 1-dimensional reconfigurable mesh.

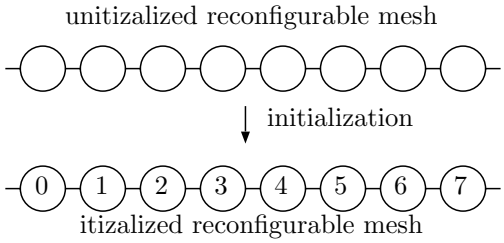

Fig. 2. Initialization on the 1-dimensional reconfigurable mesh

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 is completely identical and a reconfigurable mesh is initialized when it is booted instead of when it is manufactured, because the coordinates of each processor may be changed.

In [6], deterministic initialization algorithms on the 2-dimensional reconfigurable mesh of size $n \times n$ have been presented. It showed that the bit-model, the word-model, and the bitwise-model reconfigurable meshes can be initialize in $O(\log n)$ time, in $O(\log \log n)$ time and $O(\log^* n)$ time, respectively. Also it proves that these initialization algorithms are optimal. However, the algorithms presented in [6] do not work for 1-dimensional reconfigurable mesh.

The main contribution of this paper is to show deterministic and randomized initialization algorithms on the 1-dimensional reconfigurable mesh of the bit model. We first show a deterministic initialization algorithm that runs in $O(n)$ time. We also prove that this deterministic algorithm is optimal, that is, no deterministic solution can perform the initialization in less than $O(n)$ time. Quite surprisingly, sublinear-time initialization is possible if we use randomized techniques. More specifically, we show a randomized initialization algorithm on the bit model 1-dimensional reconfigurable mesh running in $O((\log n + \log f) \log \log n)$ time with probability at least $1 - \frac{1}{f}$ for every real number $f > 1$. It follows that this randomized initialization algorithm runs in expected $O(\log n \log \log n)$ time. The key idea of this randomized algorithm is to construct an ordered binary tree such that each processor is a node. The initialization is completed by providing the in-order numbers to all the processors. Since the ordered binary tree is balanced with high probability, we can guarantee that the initialization can be done in $O((\log n + \log f) \log \log n)$ time with probability at least $1 - \frac{1}{f}$ for every $f > 1$. We also proved that any randomized initialization

need to run in $\Omega(\log n)$ time. Thus, our randomized initialization is close to optimal.

2 A refresher of basic probability theory

This section offers a quick review of basic probability theory results that are useful for analyzing the performance of our randomized initilization algorithm. For a more detailed discussion of background material we refer the reader to [5].

Throughout, $\Pr[A]$ will denote the probability of event A. For a random variable X, $E[X]$ denotes the expected value of X. Let X be a random variable denoting the number of successes in n independent Bernoulli trials with parameters p and $1-p$. It is well known that X has a *binomial distribution* and that for every r, $(0 \leq r \leq n)$,

$$\Pr[X = r] \;=\; \binom{n}{r} p^r (1-p)^{n-r}.$$

Further, the expected value of X is given by

$$E[X] = \sum_{r=0}^{n} r \cdot \Pr[X = r] = np.$$

To analyze the tail of the binomial distribution, we shall make use of the following estimate, commonly referred to as *Chernoff bounds* [4, 5]:

$$\Pr[X \geq (1+\epsilon)E[X]] \leq e^{-\frac{\epsilon^2}{3}E[X]} \qquad (0 \leq \epsilon \leq 1). \tag{1}$$

$$\Pr[X < (1-\epsilon)E[X]] < e^{-\frac{\epsilon^2}{2}E[X]} \quad (0 \leq \epsilon \leq 1). \tag{2}$$

3 Deterministic Initialization and the Lower Bounds

This section shows a deterministic initialization algorithm on the 1-dimensional reconfigurable mesh of the bit model.

First, all processors disconnect ports and send 1 to E and try to receive 1 from W. Clearly, the processor that fails to receive 1 is the leftmost processor PE(0). Next, PE(0) sends 1 to E and all remaining processors try to receive it. The processor that succeeds in receiving 1 is PE(1). In the same way PE(1) sends 1 to E and all remaining processors try to receive it. By repeating this procedure, we have,

Lemma 1. *The 1-dimensional reconfigurable mesh of the bit model of size n can be initialized in n time.*

Next, let us formally prove the lower bound of the computing time for initialization. For this purpose, let us consider the configuration of the processors. The *configuration* of a processor is the contents of memories and registers of it.

Initially, all processors in the uninitialized reconfigurable mesh have the same configuration. It should be clear that all processors must have difference status at the end of the initialization, because each of them stores the unique ID in its memory.

Since all processors have the same configuration their behavior is the same at the beginning of the initialization. So, all of them connect W and E, or they do not connect W and E. Suppose that all of them connect W and E. Then, all processors are connected by a single subbus. Since it is not allowed that two or more processors send to the same subbus, no processor broadcast. Hence, all the processors still have the same configuration. Suppose that all of them do not connect W and E. Also, all processors can

Case 1 send data to W and receive it from E, or
Case 2 send data to E and receive it from W

Note that all the data must be the same because the configurations of the processors are identical. In Case 1, only the leftmost processor fails to receive, and the other processors succeed in receiving. So, all the processors excluding the leftmost one still have the same configuration. Similarly, in Case 2, all the processors excluding the rightmost one still have the same configuration. We can easily generalize this discussion as follows: if $\mathrm{PE}(i), \mathrm{PE}(i+1), \ldots, \mathrm{PE}(j)$ $(i < j)$ have the same configuration, after a unit of time,

- $\mathrm{PE}(i), \mathrm{PE}(i+1), \ldots, \mathrm{PE}(j-1)$ still have the same configuration, or
- $\mathrm{PE}(i+1), \mathrm{PE}(i+2), \ldots, \mathrm{PE}(j)$ still have the same configuration.

Therefore, if m (≥ 2) consecutive processors on the 1-dimensional reconfigurable mesh have the same configuration, at least $m-1$ consecutive processors have the same configuration. Since all the processors must have different configurations, we have,

Lemma 2. *Any initialization algorithm need to run in at least n time on the 1-dimensional reconfigurable mesh of size n.*

Thus, the initialization algorithm for Lemma 1 is optimal. Also, note that this lower bound holds for the reconfigurable meshes of the word model and the exclusive word model.

The lower bound of 2 is not applicable to randomized initialization. If we use randomized techniques, all processors can have different configuration without any communication. However, we can have the different lower bound for randomized algorithms. We assume that randomized algorithms are Las Vegas algorithms [5], that is, algorithms never give incorrect results but the running time is probabilistic. Thus, Las Vegas initialization algorithms always assign sequentail IDs to all the processors, but the running time is not fixed. Since at least $\log n$ bits must be sent to or receive from each processor, we have,

Lemma 3. *Any Las Vegas randomized initialization algorithm runs in at least $\log n$ time on the 1-dimensional bit model reconfigurable mesh of size n.*

4 Randomized Leader Election on the Reconfigurable Mesh

The main purpose of this section is to show a randomized leader algorithm, which is a key ingredient of our initialization algorithm. The leader election algorithm runs in $O(\log\log n+\log f)$ time with probability at least $1-\frac{1}{f}$ for every $f>1$ in the uninitialized 1-dimensional reconfigurable mesh with n processors. The idea of leader election is based on the leader election algorithm on the radio network presented in [9].

We assume that processor is not initialized, that is, no processor knows its position (or coordinate). Also, no processor knows the number n of the processors. However, the leftmost processor PE(0) knows that it is the leftmost one, and the rightmost processor PE$(n-1)$ knows that it is the rightmost one. As we have shown in Section 3, the leftmost and the rightmost processors can learn that they are the leftmost and the rightmost in $O(1)$ time.

Let $\mathcal{S}$ be the subset of the n processors. We first show that every processor can learn that if $|\mathcal{S}|=0$, $|\mathcal{S}|=1$, or $|\mathcal{S}|\geq 2$. The algorithm is spelled out as follows.

Step 1 Every processor that is not in $\mathcal{S}$ connects ports W and E. PE(0) sends 1 to W, PE$(n-1)$ sends 1 to E and every processor in $\mathcal{S}$ tries to receive it from both ports.

Step 2 Every processor connects ports W and E. A processor in $\mathcal{S}$ that has succeeded in receiving 1 from both W and E broadcast 1 and and all the processors try to receive it.

Step 3 Every processor connects ports W and E. A processors in $\mathcal{S}$ that has succeeded in receiving 1 from W and failed to receive 1 from E broadcast 1, and all the processors try to receive it.

It is clear that, if $|\mathcal{S}|=0$ then no processor broadcasts 1 in Steps 2 and 3. If $|\mathcal{S}|=1$, then the unique processor in $\mathcal{S}$ receives 1 from both ports in Step 1. Thus, it broadcasts 1 in Step 2. Also, if $|\mathcal{S}|\geq 2$ the leftmost processor in $\mathcal{S}$ receives 1 from W and broadcasts 1 in Step 3. Therefore, we can determine if $|\mathcal{S}|=0$, $|\mathcal{S}|=1$, or $|\mathcal{S}|\geq 2$ in 3 unit time.

Using this algorithm, we can find a leader at random in $O(\log\log n+\log f)$ time for every $f>0$ on the 1-dimensional reconfigurable mesh. Suppose that each of n processors on the reconfigurable mesh belongs to $\mathcal{S}$ with probability p. Let $\mathcal{S}(p)$ be the subset of processors thus obtained. It is clear that we can determine if $|\mathcal{S}(p)|=0$, $|\mathcal{S}(p)|=1$, or $|\mathcal{S}(p)|\geq 2$ in $O(1)$ time. The following algorithm elects a unique processor as a leader.

Phase 1 For each $t=0,1,2,\ldots$, find $\mathcal{S}(\frac{1}{2^{2^t}})$ and determine if $|\mathcal{S}(\frac{1}{2^{2^t}})|=0$ or not until, for the first time, $|\mathcal{S}(\frac{1}{2^{2^t}})|=0$. Let T be the value of t such that $|\mathcal{S}(\frac{1}{2^{2^t}})|=0$ for the first time.

Phase 2 Execute the binary search for the range $[0,2^T]$ as follows: Let $m=\lfloor\frac{0+2^T}{2}\rfloor$ be the median of $[0,2^T]$. Find $\mathcal{S}(\frac{1}{2^m})$ and check if $|\mathcal{S}(\frac{1}{2^m})|=0$ or

$|\mathcal{S}(\frac{1}{2^m})| \geq 1$. If $|\mathcal{S}(\frac{1}{2^m})| = 0$ then recursively execute the binary search for the range $[0, m]$. If $|\mathcal{S}(\frac{1}{2^m})| \geq 1$ then recursively execute the binary search for the range $[m, 2^T]$. Let U be the integer obtained by the binary search.

Phase 3 Find $\mathcal{S}(U)$ and checking if $|\mathcal{S}(U)| = 0$, $|\mathcal{S}(U)| = 1$, or $|\mathcal{S}(U)| \geq 2$. If $|\mathcal{S}(U)| = 1$ then the unique processor is declared as a leader and terminate the algorithm. If $|\mathcal{S}(U)| = 0$ then let $U \leftarrow U - 1$. If $|\mathcal{S}(U)| \geq 2$ then let $U \leftarrow U + 1$. Repeat finding $\mathcal{S}(U)$ for the first time $|\mathcal{S}(U)| = 1$.

Note that the expected number of processors in $\mathcal{S}(p)$ is $E[|\mathcal{S}(p)|] = np$. Hence, since $\mathcal{S}(\frac{1}{2^{2^T}}) = 0$ at the end of Phase 1, $\frac{n}{2^{2^T}} \leq 1$ holds with high probability. Hence, Phase 1 takes $O(T) = O(\log \log n)$ time with high probability. Since the binary search is executed for the range $[0, 2^T]$, Phase 2 takes $O(\log 2^T) = O(\log \log n)$ time. By a complicated proof that we can guarantee that $U \approx n$ with high probability and that Step 3 runs in $O(\log \log \log n)$ time with high probability. By the more detailed analysis of the performance, this algorithm runs in $O(\log \log n + \log f)$ time with probability at least $1 - \frac{1}{f}$ for all $f > 1$. See [9] for the details of the proof.

Lemma 4. *A leader processor on the 1-dimensional bit-model reconfigurable mesh of size n can be elected in $O(\log \log n + \log f)$ time with probability at least $1 - \frac{1}{f}$ for every $f > 1$.*

Also, note that, using the algorithm for Lemma 4 every processor can be a leader with the equal probability $\frac{1}{n}$.

5 Randomized Initialization on the Reconfigurable Mesh

The main purpose of this section is to show a randomized initialization algorithm on the 1-dimensional reconfigurable mesh.

Our randomized algorithm first make a binary tree as follows. Let p denote a processor of the reconfigurable mesh of size n. Further, let $I(p)$ be the index of processor p, that is, $p = \mathrm{PE}(I(p))$. Our goal is to find the index $I(p)$ for every processor p.

Phase 1 Find an binary tree such that every processor p is its node. We call this ordered tree *partition tree*.

Phase 2 Compute the number of nodes in every subtree of the partition tree.

Phase 3 Assign index $I(p)$ to every processor p.

The details of Phase 1 are spelled out as follows:

Phase 1 Find a partition tree such that every processor p is its node.

Phase 1.1 If the reconfigurable mesh has 1 processor, that is, $n = 1$, then select it as a leader and terminate the algorithm. Otherwise, select a leader processor using the algorithm for Lemma 4. Let p be the leader processor thus obtained.

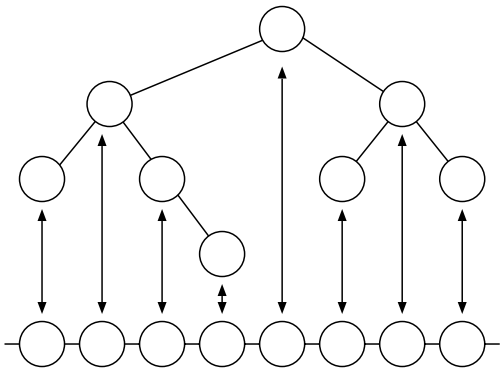

Fig. 3. The reconfigurable mesh and a partition tree computed in Phase 1

Phase 1.2 Partition the reconfigurable mesh into two submeshes as follows: The left submesh consists processors PE(0), PE(1), ..., PE($I(p)-1$) and the right submesh has processors PE($I(p)+1$), PE($I(p)+2$), ..., PE($n-1$). Recursively find the leaders of the left submesh and the right submesh.

Let us consider that the left child is the leader of the left submesh and the right child is that of the right submesh. Then, we can obtain an ordered binary tree such that the root is the leader of the whole reconfigurable mesh. Also, it is easy to confirm that the order of visit in each node p by the in-order traversal is the index $I(p)$.

For each node (or processor) p of the partition tree, let $N(p)$ denote the number of processors in the subtree with root p. Initially, we assume that $N(p)$ is undefined. Also, let p_L and p_R denote the left and the right children of node p if exist. The following algorithm compute $N(p)$ for every p.

Phase 2 Compute the number of nodes in every subtree.

Phase 2.1 For each leaf node p, let $N(p) = 1$. Each leaf processor p sends $N(p)$ to its parent.

Phase 2.2 For each internal node p, when processor p receives $N(p_L)$ and $N(p_R)$ from its children, it computes $N(p) \leftarrow N(p_L) + N(p_R) + 1$ and sends $N(p)$ to its parent. If the root node p receives $N(p_L)$ and $N(p_R)$ from its children, it computes $N(p) \leftarrow N(p_L) + N(p_R) + 1$ and terminate Phase 2.

At the end of this stage, each processor p knows the values of $N(p)$, $N(p_L)$ and $N(p_R)$. For each node p, let $M(p)$ denote the number of nodes whose indexes smaller than all nodes in subtree with root p. Phase 3 computes $M(p)$ and $I(p)$ for all node p. Initially, $M(p)$ and $I(p)$ are undefined.

Phase 3 Assign index $I(p)$ to every processor p.

Phase 3.1 For the root node p, let $M(p) \leftarrow 0$ and $I(p) \leftarrow N(p_L)$. The root node PE(p) sends $M(p)$ $(= 0)$ to the left child p_L and $I(p) + 1$ to the right child. to the right child p_R.

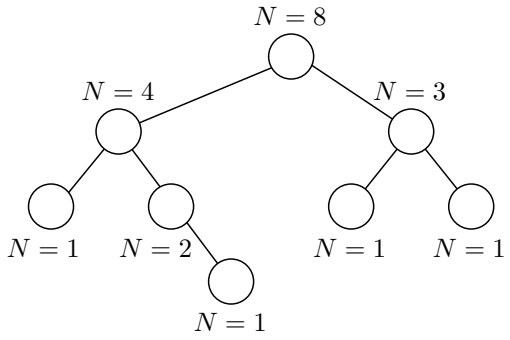

Fig. 4. The values of $N(p)$ computed in Phase 2

Phase 3.2 For each non-root node p, when it receives an integer m, let $M(p) \leftarrow m$ and $I(p) \leftarrow M(p) + N(p_L)$. Also, it sends $M(p)$ to the left child p_L and $I(p) + 1$ to the right child p_R if exist.

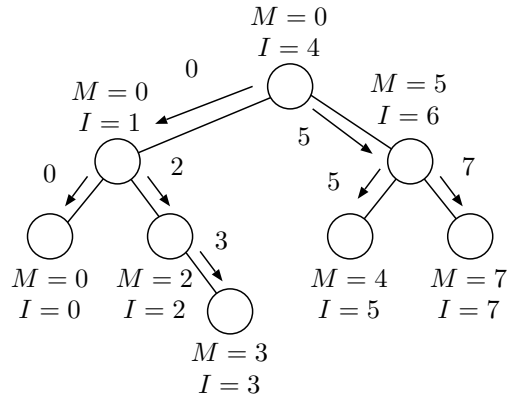

Fig. 5. The values of $M(p)$ and $I(p)$ computed in Phase 3

Let h be the height of the ordered binary tree, that is, the maximum number of nodes over all paths from the root to leaves. We will prove that $h = O(\log n)$ with high probability.

Recall that, the reconfigurable mesh is recursively partitioned into two submeshes in Phase 1. Let n be the number of processors in the reconfigurable mesh. We call the partitioning is *success* if both of the left and the right submeshes have at most $\frac{3}{4}n$ processors. Let p be the leader elected in Phase 1.1. Then, the left submesh has $I(p)$ processors and the right submesh has $(n-1) - I(p)$ processors. Thus, the partitioning is success if $\frac{1}{4}n - 1 \leq I(p) \leq \frac{3}{4}n$. Since a leader p

is elected at random from n processors, the partition is success with probability at least $\frac{1}{2}$. We can say that, for any path from the root to a leaf, it has at most $\frac{\log n}{\log \frac{4}{3}}$ success nodes. If there exists a path from the root to a leaf p that has more than $\frac{\log n}{\log \frac{4}{3}}$ success nodes, $N(p) < (\frac{3}{4})^{\frac{\log n}{\log \frac{4}{3}}} n \leq 1$ holds. However, since p is a leaf, $N(p) = 1$. Therefore, every path has at most $\frac{\log n}{\log \frac{4}{3}}$ success nodes.

Let X be a $\frac{4 \log n}{\log \frac{4}{3}} + 8 \ln(nf)$ Bernoulli trials with success probability $\frac{1}{2}$, where $f > 1$ is any real number. Clearly, $E[X] = \frac{2 \log n}{\log \frac{4}{3}} + 4 \ln(nf)$. From the Chernoff bound, The probability that X is less than $\frac{\log n}{\log \frac{4}{3}}$ is

$$\begin{aligned} \Pr[X < \frac{\log n}{\log \frac{4}{3}}] &< \Pr[X < (1 - \frac{1}{2})(2\frac{\log n}{\log \frac{4}{3}} + 8 \ln(nf))] \\ &< e^{-\frac{1}{8}E[X]} < e^{-\ln(nf)} = \frac{1}{nf}. \end{aligned}$$

Hence, X is at most $\frac{\log n}{\log \frac{4}{3}}$ with probability at most $\frac{1}{nf}$. Therefore, a particular path from the root to a leaf has at least $\frac{4 \log n}{\log \frac{4}{3}} + 8 \ln(nf)$ nodes with probability at most $\frac{1}{nf}$. The binary tree of n nodes has at most $\frac{n}{2}$ leaves, and thus, has $\frac{n}{2}$ paths. It follows that all of the paths have at most $\frac{4 \log n}{\log \frac{4}{3}} + 8 \ln(nf)$ nodes with probability at most $\frac{n}{2} \times \frac{1}{nf} < \frac{1}{f}$. Therefore, the height of the tree is at most $\frac{4 \log n}{\log \frac{4}{3}} + 8 \ln(nf) = O(\log n + \log f)$ with probability at least $1 - \frac{1}{f}$.

Lemma 5. *The height of the partition tree obtained in Phase 1 is at most* $O(\log n + \log f)$ *with probability at least* $1 - \frac{1}{f}$ *for every* $f > 1$.

Next, let us evaluate the computing time using Lemma 5. In Phase 2, the values of $N(p)$'s are sent from the leaf to the root. The operation in each node takes $O(1)$ time. Hence, the computing time of Phase 2 is proportional to the height of the partition tree. Similarly, in Phase 3, the values of $M(p)$'s and $I(p)$'s are sent from the root to the leaf and the operation in each node takes $O(1)$ time. Thus, the computing time of Phase 3 is also proportional to the height. Consequently, Phases 2 and 3 run in $O(\log n + \log f)$ time with probability at least $1 - \frac{1}{f}$.

Finally, let us evaluate the computing time of Phase 1. Recall that from Lemma 4, a leader can be elected in in $O(\log \log n + \log f)$ time with probability at least $1 - \frac{1}{f}$. For each integer $i \geq 1$, let T_i be the computing time such that a leader can be elected in T_i time with probability $1 - \frac{1}{2^i}$. Further, for each $i \geq 2$, let $t_i = T_i - T_{i-1}$. From $T_i = O(\log \log n + i)$, we have $T_1 = O(\log \log n)$ and $t_i = O(1)$. Clearly, the probability that the leader election does not terminate in T_i $(i \geq 1)$ time and terminates in additional t_{i+1} time is exactly $\frac{1}{2}$.

Suppose that the leader election algorithm for Lemma 4 is executed on the reconfigurable mesh with n processors. Then, it elects a leader in T_1 time with probability $\frac{1}{2}$. If it does not elect a leader in T_1 time, then it elects a leader in

additional t_2 time. Again, if it does not elect a leader in additional t_2 time, then it elects a leader in additional t_3 time. In general, for every $i \geq 1$, if a leader does not elected in T_i time then a leader is elected in additional $t_{i+1} = O(1)$ time with probability $\frac{1}{2}$. Let us put black and white pebbles in nodes of the partition tree for the purpose of evaluating the computing time. Recall that a node of the partition tree corresponds to a leader election and the partitioning is success if both of the left and the right submeshes have at most $\frac{3}{4}n$ processors. We put the black and white pebbles in every node according to the following rules. Note that each node may have two or more pebbles.

Rule 1 If the leader election terminates in T_1 time and the partitioning is success then put a black pebble in the node.

Rule 2 If the leader election terminates in T_1 time but the partitioning is failure then put a white pebble in the node.

Rule 3 If the leader election does not terminates in T_1 time, put a white pebble in the node.

Rule 4 For each i (≥ 1), if the leader election does not terminate in T_i time but terminates in additional t_{i+1} time and the partitioning is success then put a black pebble.

Rule 5 For each i (≥ 1), if the leader election does not terminate in T_i time but terminates in additional t_{i+1} time and the partitioning is failure then put a white pebble.

Rule 6 For each i (≥ 1), if the leader election does not terminate in $T_{i+1} = T_i + t_{i+1}$ time then put a white pebble.

Clearly, the leader election does not terminate in T_i time but terminates in additional t_{i+1}, the corresponding node has $i+1$ pebbles. Also if the partitioning is failure, all pebbles in the node is white. If the partitioning is success, then one of the pebbles is black and the others are white.

Next let us estimate the number of pebbles in the path of the partition tree from the root to a leaf. The leader election terminates in T_1 time with probability $\frac{1}{2}$. The probability that the leader election terminates in additional t_{i+1} time is $\frac{1}{2}$ if it has not terminate in T_i time. Also, the partition is success with probability at least $\frac{1}{2}$, Thus, a particular pebble is black with probability at least $\frac{1}{4}$. As we have proved, a path from the root to a leaf has at most $\frac{4\log n}{\log \frac{4}{3}}$ success nodes. Hence, any path has at most $\frac{\log n}{\log \frac{4}{3}}$ black pebbles. Using this fact, we will show that a path has no more than $\frac{8\log n}{\log \frac{4}{3}}$ pebbles with high probability.

Let Y be a $8\frac{\log n}{\log \frac{4}{3}} + 32\ln(nf)$ Bernoulli trials with success probability $\frac{1}{4}$, where $f > 1$ is any real number. Clearly, $E[Y] = 2\frac{\log n}{\log \frac{4}{3}} + 8\ln(nf)$. From the Chernoff bound, The probability that Y is less than $\frac{\log n}{\log \frac{4}{3}}$ is

$$\begin{aligned}\Pr[Y < \frac{\log n}{\log \frac{4}{3}}] &< \Pr[Y < (1-\frac{1}{2})(2\frac{\log n}{\log \frac{4}{3}} + 8\ln(nf))] \\ &< e^{-\frac{1}{8}E[Y]} < e^{-\ln(nf)} = \frac{1}{nf}.\end{aligned}$$

Hence, Y is at most $\frac{\log n}{\log \frac{4}{3}}$ with probability at most $\frac{1}{nf}$. It follows that, a particular path has at least $8\frac{\log n}{\log \frac{4}{3}} + 32\ln(nf)$ pebbles with probability at most $\frac{1}{nf}$. Since the partition tree has less than n paths, no path has more than $8\frac{\log n}{\log \frac{4}{3}} + 32\ln(nf)$ pebbles with probability at most $1 - \frac{1}{f}$. Since each pebble corresponds to $O(\log \log n)$ time, we have,

Theorem 1. *The 1-dimensional reconfigurable mesh of the bit model of size n can be initialized in $O((\log n + \log f) \log \log n)$ time with probability at least $1 - \frac{1}{f}$ for every $f > 1$.*

As we have shown, the expected value of random variable that is no more than $\log f$ with probability at lest $1 - \frac{1}{f}$ is $O(1)$. Thus, this algorithm to runs in expected $O(\log n \log \log n)$ time. Also, the initialization algorithm for 1 is a Las Vegas type randomized algorithm [5] because it always returns the correct results but the running time is probabilistic.

References

1. A. G. Bourgeois and J. L. Trahan. Fault tolerant algorithms for a linear array with a reconfigurable pipelined bus system. *Parallel Algorithms and Applications*, 18(3):139–153, 2003.
2. E. Hao, P. D. Mackenzie, and Q. F. Stout. Selection on the reconfigurable mesh. In *Proceedings of 4th Symposium on Frontiers of Massively Parallel Computation*, pages 38–45. IEEE, Oct. 1992.
3. T. Hayashi, K. Nakano, and S. Olariu. Efficient list ranking on the reconfigurable mesh, with applications. *Theory of Computing Systems*, 31:593–611, 1998.
4. J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
5. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
6. K. Nakano. Optimal initializing algorithms for a reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 24(2):218–223, Feb. 1995.
7. K. Nakano and S. Olariu. An optimal algorithm for the angle-restricted all nearest neighbor problem on the reconfigurable mesh, with applications. *IEEE Transactions on Parallel and Distributed Systems*, 8(9):983–990, 1997.
8. K. Nakano and S. Olariu. An optimal algorithm for the angle-restricted all nearest neighbor problem on the reconfigurable mesh, with applications. *IEEE Transactions on Parallel and Distributed Systems*, 8(9):983–990, 1997.
9. K. Nakano and S. Olariu. Uniform leader election protocols in radio networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(5):516–526, May 2002.
10. K. Nakano and K. Wada. Integer summing algorithms on reconfigurable meshes. *Theoretical Computer Science*, 197(57–77), 1998.
11. M. Nigam and S. Sahni. Sorting n numbers on $n \times n$ reconfigurable meshes with buses. *Journal of Parallel and Distributed Computing*, 23(1):37–48, Oct. 1994.
12. R. Vaidyanathan and J. L. Trahan. *Dynamic Reconfiguration: Architectures and Algorithms*. Kluwer Academic/Plenum Publishers, 2003.