数理解析研究所講究録
第754巻 1991年 257-266

メッシュバス機械の性能評価
Computational Performances of Mesh-Bus Machines

岩間一雄　宮野英次　上林弥彦
(九大・工)　(京大・工)

1. Introduction

In [5], the authors introduced a new two dimensional parallel model, called the mesh-bus computer (MB, Fig.1), by replacing the local connections of the common mesh-connected computer (MC, Fig.2) with row and column buses. In this paper, we apply a "benchmark test" to this new model MB by selecting five typical problems appearing often in the literature. The result is a little surprising; it seems quite reasonable to claim MB's superiority over MC. Out of the five problems, MB is clearly better than MC for three and is not worse for the rest.

The five problems include graph connectivity, finding maximum, semigroup operations, routing and sorting. They take $n \times n$ data ($n \times n$ incidence matrices for graphs) as their inputs. For the first two problems, MB allows us to develop logarithmic depth algorithms, while MC has a trivial lower bound of $\Omega(n)$ (see [2] for graph problems). Routing also needs trivial $2n$ steps over MC but $1.5n$ ($1.0n$ for some useful subsets of instances) is enough on MB. Most semigroup operations are carried out in time $\Theta(n)$ over both MB and MC. For several reasons mentioned later, sorting is probably the hardest problem for MB. We nevertheless present, as the most significant result of this paper, an $O(n)$ randomized algorithm, called the binomial curve sort, which depends on only row (column)-based sorting.

Note that MB and MC are incomparable in the worst case analysis, namely, it is not hard to think of trivial operations for which either of them needs $\Omega(n)$ steps to simulate the other's $O(1)$ steps. However, our main claim appears to be quite legitimate for nontrivial and natural problems including not only the above five ones but many others. It seems hard to find such problems for which MC is essentially better than MB and this will be a nice target for the future study.

Remark 1. There is a huge literature on the mesh-connected computers. Also a fairly large number of papers discussed mesh-type bus-communication [1] [3] [11] [12]. However, all of them introduced the bus-communication as an *addition* to the local connections in order to avoid the large diameter of mesh-connected computers. To supply with both buses and local connections may seem an easy and reasonable solution but actually is not: There is a wide agreement that the degree of integration depends on the number of communication lines rather than on the number of logic gates. If this statement is true, and if we make a rough estimation, to use both buses and local connections needs doubled space for low and column directions. It then needs four times as large space as either only buses or only local connections, or the maximum number of processors for a limited space becomes one fourth.

Remark 2. Even if we do assume that buses and local connections are both available, the known results do not seem so positive. The most successful one would be the reduction of time from n (without buses) to $n^{\frac{1}{3}}$ for semigroup operations [12]. For sorting (and probably for routing) the added buses do not contribute to any reduction of time [11] under the big-O notation. No logarithmic depth has ever been reported for nontrivial problems. Our achievement of that small depth largely (but not exclusively) depends on our new assumption that two or more processors can write data into a single bus at the same step. It should be noted that this assumption, although no previous reports adopted it, is quite natural if we realize our bus system as a bunch of so-called wired-or lines. Our regulation for the simultaneous write is called BUS: If the written values are all the same then the value survives. Otherwise, i.e., if different values are written, then the collision is the only information we know. Note that we do not need the simultaneous write for the $O(n)$ algorithms.

Remark 3. Both MB and MC can be regarded as the parallel random access machines (PRAMs) with restriction: MB can only use $2\sqrt{N}$ memory cells for N processors but each cell can be accessed from (fixed) $\sqrt{N}$ processors. (BUS regulation is weaker than ARBITRARY.) MC can use about N cells but each cell can be accessed from only two processors. (It is a nice property for parallel models that they can be simulated by PRAMs with a similar or less amount of resources in the sense that the models at least do not violate the rule of the most standard model. It should be noted that recently developed reconfigurable arrays [13] do not have this property.)

2. Graph Connectivity and Finding Maximum

Algorithms in this section are randomized and need the simultaneous write.

Theorem 1 [5]. Suppose that each of $n \times n$ processors initially holds each value of an $n \times n$ incidence matrix of a graph G. Then we can compute connected components of G over MB (BUS regulation) in $O(\log n)$ expected time.

Thus the performance of MB does not differ so much from CRCW-PRAMs for this particular problem [10]. We conjecture that MB also has (poly)log-depth algorithms for other graph problems like obtaining a minimum spanning tree. This conjecture would be strengthened by the following result:

Theorem 2. MB (BUS regulation) can find a maximum among $n \times n$ data in $O(\log n)$ expected time.

Proof (Sketch). Using the randomized procedure mentioned later, we first try to pick out a single data, at random, among n ones per each row, using constant steps. It is not necessary to actually succeed in doing so for all the rows but is enough to succeed for "many" rows. Then we gather those data to the processors at the diagonal position and we compute the maximum among those (at most n) data in constant steps using the standard technique. Then all the processors know this maximum M through buses and "kill" its own data if it is less than M. Now the number of live data should be reduced, ideally, from n^2 to some n, or a constant number per row on average. This might be too optimistic but we can certainly expect, with a good probability, that the largest number of live data on any single row is reduced to a factor of $\frac{1}{c}$ for some constant $c > 1$.

Now we repeat this operation. Again we wish to pick out a single data for all the rows or at least for the rows on which a large number of data remain live. Then the number of those data should be reduced to a factor of $\frac{1}{c}$ again and so on.

The key tool for this operation is the procedure, called Pick_One, introduced in [5]. Roughly speaking, each processor on a single row holding a live data tries to "raise a flag" with probability q ($= \frac{1}{n}$ initially). If no processors on that row actually raise flags, then they simply finish that iteration of the procedure's main loop after executing $q := 2q$. Otherwise, if, fortunately, exactly one processor actually raises a flag, then it is picked out. If two or more processors raise flags (we can tell that fact thanks to the BUS regulation) then they further try to pick out exactly one among them using a constant steps (e.g., by trying to raise flags again with probability, say, $\frac{1}{2}$). □

3. Routing

In the rest of the paper no algorithms assume the availability of the simultaneous write. (It does not seem to work effectively for the rest of the problems.) All the algorithms in this section are deterministic. Routing is the following problem: Given $n \times n$ pairs of (data, destination)'s, we are supposed to move all the data to their destinations. We need a somehow precise definition for one-step computation, which includes (i) reading the data on both row and column buses, (ii) executing constant steps of local computation and (iii) possibly writing data into row and/or column buses. It is easy to develop an elementary algorithm for routing which runs in $2n$ steps. Our improved result is:

Theorem 3. $1.5n$ steps are enough for routing over MB.

Proof (Sketch). $n \times n$ processors are divided into four $\frac{n}{2} \times \frac{n}{2}$ ones. Using the first $\frac{n}{2}$ steps, we move upper-left $\frac{n}{2} \times \frac{n}{2}$ data, sequentially from left to right, horizontally to their correct positions with respect to column (see Fig.3) The lower-right $\frac{n}{2} \times \frac{n}{2}$ data are also moved horizontally and the rest half of data are moved vertically in a similar fashion. Note that a single processor may hold up to n data at this moment. Using the next n steps, we move all those data (at the correct positions either in row or column) to their final positions. They are put on the buses not in the order of their current positions as before but in the order of their destinations. □

In this algorithm, we can reduce the n steps of the second stage into $\frac{n}{2}$ steps if the input has some regularity. Those regularities include practical ones such as the transposition and the 180° rotation.

For this problem, a lower-bound analysis might be more interesting. We have the following result under the assumption that (i) every data written on a bus must be one of the original $n \times n$ input pairs and (ii) the simultaneous write is prohibited.

Theorem 4. Suppose that the input data can take the value up to C. Then if $C > (\alpha n)^{\frac{\alpha}{1-\alpha}}$ for a constant $\alpha(< 1)$, then routing needs more than αn steps (Proof will be given in a future report).

It should be noted that the condition $C > (\alpha n)^{\frac{\alpha}{1-\alpha}}$ only says that the number of bits for each data is up to $\frac{\alpha}{1-\alpha} \log n$, which is quite reasonable, even if we set $\alpha \approx 1$, for a large n. Thus the theorem gives us the lower-bound of roughly n steps. It seems difficult to narrower the gap between $1.5n$ and n.

4. Sorting

A lot of $O(n)$ algorithms over MC have been developed (e.g., [7] [9]). Their common techniques include: (i) Recursive divide-and-conquer (typically, dividing the whole $n \times n$ into four $\frac{n}{2} \times \frac{n}{2}$, then each $\frac{n}{2} \times \frac{n}{2}$ into four $\frac{n}{4} \times \frac{n}{4}$ and so on). (ii) Partial sorting, for example, as shown in Fig.4. (iii) The odd-even-transposition sort, which is in particular useful when the data is almost sorted (we can execute this sorting algorithm for a small number of steps.) One can see that our global bus system is quite undesirable to those methods since the buses cannot be "cut". Our new binomial curve sort (BCSORT) depends only on row (column)-based sorting and makes full use of the randomization.

We first introduce a simpler version of BCSORT to make the basic idea clear. The input data is given as an $n \times n$ array, which we sort into the snake-like row-major order . To *randomize a row* (*a column*) means to permute the elements on that row (column) at random. We can do so by generating random numbers, one for each element, and then sorting the elements by those random numbers.

Algorithm BCSORT-I :

Step 1. Randomize all rows.

Step 2. Randomize all columns.

Step 3. Sort all columns from top to bottom.

Step 4. Sort all odd-numbered rows from left to right and all even-numbered rows from right to left. (That is called *shearing* in [8]). Test if the sorting is completed, and stop if it is.

Step 5. Sort all columns from top to bottom.

Step 6. Randomize all rows. Go to Step 3.

To execute each of Step 1 to Step 6, we apparently need $\Omega(n)$ steps for both MB and MC. For MB, a method called "counting sort" seems to be convenient. Each element is placed on the bus in a regular order while the other elements know their positions by counting the number of smaller elements. To achieve $O(n)$ total running time, therefore, we have to keep the number of iterations from Step 3 to Step 6 within $O(1)$.

We shall take a look at the behavior of the algorithm when the input consists of only 0's and 1's (see Lemma 1 for our version of the zero/one-principle.). We also assume, for simplicity, that the number of 1's is exactly λn, for some integer $\lambda \leq n$. Steps 1 and 2 deliver all 1's at random on the $n \times n$ plane (actually Step 2 is not necessary). Step 3 moves all 1's towards the bottom. Suppose, hypothetically, that we sort all rows to the right after that. The result would look like Fig.5. One can see that the curve becomes close to the binomial distribution $b(k, n, \frac{\lambda}{n})$, i.e., (the number of columns including k 1's)$/n$ should be nearly equal to $b(k, n, \frac{\lambda}{n})$. Again, suppose hypothetically that we can change the direction of rows (from left-to-right to right-to-left) at the λ'th row as in Fig.6. It is a well-known fact that the two curves on both sides of the border (the λ'th row) are nearly symmetric unless λ is close to zero. This implies that if we sort columns again, then both the 1's misplaced above the border and the 0's under the border would be "neutralized" almost completely. In reality, however, we cannot know where the border is, and for a general input, such a border does not exist. Fortunately, Step 4 has almost the same effect as changing the direction at the λ'th row, which will be shown in the next section.

Thus, the configuration after Step 5 should look like Fig.6, where most of wrongly-placed 0's and 1's stay very close to the border. If these 0's and 1's stay only on either the λ'th row or the $(\lambda + 1)$st row (i.e., 0's on λ'th and 1's on $(\lambda + 1)$st), then one can see that they will definitely move to the right place in the next iteration. Otherwise, Step 6 plays its role. Suppose that we accidentally encounter a tall "pile" of wrongly-placed 1's as shown in Fig.7.

Such a pile is dangerous since (if Step 6 is missing) its height may decrease only by half during a single iteration. By Step 6, however, these piles are demolished evenly and the probability of encountering such piles again in the next iteration should be decreased greatly.

This observation depends on a key assumption that the binomial curve in Fig.6 or Fig.7 is nearly symmetric. However, it is also well known that it is not the case if λ is small. Then we should encounter, not accidentally, a large number of the high piles mentioned above. It is not clear any longer if Step 6 can manage the situation only by itself. Here is our algorithm in a complete form :

Algorithm BCSORT-II :

Step 1 - Step 5. Exactly the same as the simplified version above.

Step 6. Partition the rows into "layers". Borders between the layers are at 4th row, 8th, 16th, 32th, ..., 2^ith, ... from both the top and the bottom rows (see Fig. 8).

Step 7. Randomize all rows.

Step 8. Randomize all columns *within* the layers (i.e., any element never goes outside the borders of its own layer).

Step 9. Shear the rows as in Step 4 and sort all columns again *within* the layers.

Step 10. Sort all rows from left to right (not shearing).

Step 11. See Fig.9. At each row i, in the lower half plane, i.e., $(1 \leq i \leq \frac{n}{2})$, the right most $t(i)$ elements (see below for $t(i)$ and the next $s(i)$), denoted by A in the figure, are exchanged with the same number of elements, B, beginning with the $s(i)$th column from the right. We do the same for the upper half plain but the left most portion is shifted to the right.

Step 12. Repeat Step 3 - Step 11 another three times (four in total). At the second iteration, layering in Step 6 is a little different. The borders set at 5th, 10th, 20th, 40th, ..., $(2^i + \frac{1}{4}2^i)$th, ..., i.e., the borders are "shifted" one fourth to the center. In the third iteration, the borders are shifted two fourths, they are shifted three fourths in the final iteration.

Step 13 - Step 15. The same as Step 3 - Step 5.

Step 16. Randomize all rows. Go to Step 13.

$s(i)$ in Step 11 is determined as follows:

$$s(0) = 0, \qquad s(i) = s(i-1) + t(i-1)n \quad \text{for } i \geq 1.$$

The values of $t(i)$ will be defined precisely in the proof of Lemma 2. Roughly speaking, we first determine its values at the borders of the layers, $t(b_1 = 4)$, $t(b_2)$, $t(b_3)$, and so on. $t(b_1)$ and $t(b_2)$ are given explicitly. Starting from $j = 3$, $t(b_j)$ is given as $\beta t(b_{j-1})$ for a constant $\beta < 1$. For rows between the borders, it is given as $t(b_j + k) = t(b_j)\alpha^k$ for a constant $\alpha < 1$. α and β, given also in the proof, are fairly small, and it is guaranteed that $\sum t(i) < 1$ for any large n.

As mentioned before, we cannot expect a desirable symmetry of the binomial curve when λ is very small, e.g., when $\frac{\lambda}{n}$ (the ratio of 1's) is $O(\frac{1}{n})$. Intuitively, we can change this ratio up to some $\frac{1}{4}$ by the layering and the following operations within each layer in Steps 6 - 10 . For the sake of the proof, we want this ratio to be between 0.25 and 0.5. That is the reason for Steps 3 - 11 to be repeated four times. (It is interesting to see that in Step 8, we "de-sort" columns that are once sorted.)

In Step 12, we move the right-end portions of each row, so that they will never overlap with the same portions on the other rows. The piles of wrongly-placed 1's were collected to those right-end portions by the preceding step. One can see that the piles are thus demolished in a deterministic fashion rather than in a probabilistic one in Step 6 of the old simplified version. The crucial point is to make the size of those portions large enough to include all possible piles, with, at the same time, keeping the total length of the portions within n (the length of a single row). As one can see later, the layering plays a key role to this end. We should not forget that there is another reason for the symmetry to be undermined. It is "by accident". The iteration of Steps 13-16 is introduced to manage this indispensable perturbation for probabilistic algorithms.

Remark 4. BCSORT-I steps within $O(n \log\log n)$ steps with a high probability [4]. [9] presents another $O(n \log\log n)$ deterministic algorithm over MC but based on only row (column)-based sorting, which MB can simulate. We have a conjecture that BCSORT-I is much more efficient that the above analysis.

Remark 5. We carried out a simulation for BCSORT-I and its slight modification BCSORT-III. In the latter, we have Step 5.5 after Step 5 in which, after sorting all rows to the right, the right (left) most elements are shifted to the left (right) as in Step 11. However, we only shift a fixed number ($\sqrt{n}$) of elements and therefore they overlap with their counterparts of every $\sqrt{n}$ rows. In Table 1, T_i (T_3) denotes the average number of iterations of Step 3 - 6 before BCSORT-I (BCSORT-III) stops. The number of trials is at least 50. Input data was generated at random.

Now we will show our main theorem in this paper.

Theorem 5. BCSORT-II stops at the third iteration of Steps 13-16 with probability $1 - \frac{c}{\sqrt{\log n}}$ for some constant c.

Two lemmas will be given to justify this theorem.

Lemma 1. Suppose that BCSORT-II sorts any (single) input data consisting of only 0's and 1's, after executing k "Steps" with probability at least $1 - p$. Then it sorts any (single) input of integers after the same execution sequence of Steps with probability at least $1 - n^2 p$.

Proof (Sketch). Without loss of generality, we can assume that the general input is a permutation of $(1, 2, \ldots, n^2)$. The algorithm is oblivious, i.e., its control sequence does not depend on the values of input data or on the sequence of generated random numbers. Suppose that the algorithm generates a sequence, T, of random numbers. If we fix this sequence, the algorithm can be regarded as a deterministic one.

Now suppose that the input sequence, $(a_1, \ldots, a_{n^2})$, is changed to $(b_1, \ldots, b_{n^2})$ under the sequence T after the execution of k Steps. If $(b_1, \ldots, b_{n^2})$ is not sorted, then we can find the least integer b_{x+1} such that $b_x > b_{x+1}$. (We say that the algorithm does not sort the input $(a_1, \ldots, a_{n^2})$ at b_{x+1}.) Now we consider an input sequence $(f_{b_{x+1}}(a_1), \ldots, f_{b_{x+1}}(a_{n^2}))$ of only 0's and 1's where

$$f_b(a) = \begin{cases} 0 & \text{if } a \leq b, \\ 1 & \text{otherwise.} \end{cases}$$

Under the same sequence T of random numbers, this input should be turned to

$$(f_{b_{x+1}}(b_1), \ldots, f_{b_{x+1}}(b_{n^2})),$$

exactly as the original zero/one-principle [6] claims. Clearly, this sequence is not sorted either.

Now consider all possible sequences of random numbers of proper length for k Steps. The discussion above leads us to the conclusion that if the algorithm does not sort a (general) input $(a_1, \ldots, a_{n^2})$ at b_{x+1} within k Steps with probability q, then it does not sort a binary input $(f_{b_{x+1}}(a_1), \ldots, f_{b_{x+1}}(a_n))$ with probability at least q either. As a result,

$$\begin{aligned} &P_r((a_1, \ldots, a_{n^2}) \text{ is sorted within } k \text{ Steps }) \\ &\geq 1 - \sum_{b=1}^{n^2} P_r((a_1, \ldots, a_{n^2}) \text{ is not sorted at } b) \\ &\geq 1 - \sum_{b=1}^{n^2} P_r((f_b(a_1), \ldots, f_b(a_{n^2})) \text{ is not sorted }) \end{aligned}$$

Then, the lemma follows since the assumption says that $P_r((f_b(a_1), \ldots, f_b(a_{n^2}))$ is not sorted $)$ is at most p. □

Lemma 2. BCSORT-II stops for any binary input at the third iteration of Steps 13-16 with probability $1 - \frac{c}{n^2\sqrt{\log n}}$ for some constant c.

Proof (Sketch). As before, the probability that a single column contains k 1's is denoted by $b(k, n, \frac{\lambda}{n})$. Since we discuss asymptotic behaviors, we can use the normal approximation given by

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

for $b(\lambda + x\sqrt{\lambda q}, n, \frac{\lambda}{n})\sqrt{\lambda q}$ where $q = (1 - \frac{\lambda}{n})$.

Now suppose that we are done with Step 3. If we hypothetically sort all the rows to the right, the configuration would look like Fig.10. This figure illustrates the case for $\lambda = 10$.

Recall that in Step 4, the direction of sorting is changed at every row. Look at, for example, the 10th row. If this row is sorted to the right, then 1's take the place denoted by $one(0)$ in the figure. The 9th row should be sorted to the left and the 0's should stay where denoted by $zero(0)$, and so on.

Then all columns are sorted in Step 5. It is convenient for us to consider that by this sorting the 1's denoted by $one(0)$ is exchanged with 0's denoted by $zero(0)$, $one(2)$ with $zero(2)$, ..., on the right side of the plane and $one(1)$ with $zero(1)$, ..., on the other side. We can use a numerical table for such small λ and can verify the following properties : (i) $zero(i)$ is larger than $one(i)$ where i is close to 0. The reason is that the largest stair of Fig.10 that corresponds to $b(\lambda, n, \frac{\lambda}{n})$ does not exist at the exact center but is shifted a little to the right. $b(\lambda - i, n, \frac{\lambda}{n})$ is larger than $b(\lambda + i, n, \frac{\lambda}{n})$ at the beginning but this relation soon overturns as i grows. (ii) Hence, at some point, $one(i)$ becomes larger than $zero(i)$. This tendency becomes more and more extreme as i is getting close to λ. Therefore, after the column sorting, the configuration can be shown as in Fig.11. Recall that the average is $\lambda(= 10)$. Below the average we find wrongly-placed 0's only on the 10th row, we find wrongly-placed 1's on both left and right ends, which can become relatively tall piles. It should be noted that this observation also holds for larger λ.

Now we are going to the next important step, Step 8. Recall that we repeat Steps 3 - 11 four times with slightly changing the layering. Suppose, again, that $\lambda = 10$. Then the (final) border between 0's and 1's fit the third layer (beginning with the 9th row and ending with 16th) in the first iteration. That layer can be regarded as including about $\frac{1}{4}$ 1's and $\frac{3}{4}$ 0's. After Steps 9 and 10, we will again encounter, as in Fig.11, the wrongly-placed 0's on the 10th row and piles of wrongly-placed 1's at both left and right ends. This time, however, one should notice a great difference; the ratio of 1's is changed from very small $\frac{10}{n}$ to $\frac{1}{4}$. We can show that this new ratio is large enough to claim that these 1's piles only appear at the "far end" of the row as follows:

In this specific example, we can calculate explicitly the values of $b(0, 8, \frac{1}{4})$, $b(1, 8, \frac{1}{4})$, ..., $b(8, 8, \frac{1}{4})$, which shows that (i) $b(2, 8, \frac{1}{4})$ is the largest, (ii) $b(0, 8, \frac{1}{4})$ and $b(1, 8, \frac{1}{4})$ are very close to $b(4, 8, \frac{1}{4})$ and $b(3, 8, \frac{1}{4})$, respectively and (iii) $b(5, 8, \frac{1}{4})$, $b(6, 8, \frac{1}{4})$ and so on are very small. In other words, the 1's piles can only appear according to $b(5, 8, \frac{1}{4})$, ..., $b(8, 8, \frac{1}{4})$. It is not hard to show this in general, i.e., we only have to consider the 1's piles according to $b(2wp + i, w, p)$ for $i \geq 0$, where w is the width of the layer and p is the ratio of 1's *inside* that layer. Since we can assume $\frac{1}{4} \leq p \leq \frac{1}{2}$, $b(2wp, w, p)$ becomes maximum (the worst case for us) at $p = \frac{1}{4}$.

Now we are ready to determine the value of $t(i)$ used in Step 11. When w (= the width of the layer) is 8, $b(5, 8, \frac{1}{4}) + \ldots + b(8, 8, \frac{1}{4}) < 0.04$, which is enough for $t(8)$. Similarly we take 0.06 for $t(0)$ and $t(4)$. $b(2wp + i, w, p)$ decreases sharply as i increases and therefore we can choose, e.g., $\frac{1}{4}$ for factor α. Using the normal approximation one can also verify that $b(4wp, 2w, p)$ is much smaller than $b(2wp, w, p)$. Therefore we can choose, e.g., $\frac{1}{4}$ again for β. Thus, by the shifting operation in Step 11, all of the wrongly placed 1's are completely distributed so that any two of them never occupy the same column. That means all the wrongly placed 0's and 1's are gathered to only two consecutive rows after Step 13 and the algorithm must stop at Step 14 with probability one!

Of course that is too optimistic since the discussion so far assumes that the distribution of 0's and 1's completely follows the theoretical distribution. Consider, for example, the probability, $p(h)$, that a row contains $(\lambda + h\sqrt{\lambda q})$ 1's or more. The normal approximation gives us, for example, nearly 0.16 as $p(1)$. It should be noted that the actual number, say $\sharp(h)$, of such rows can change again following the binomial distribution. Let m be the number of columns, which we wish to distinguish with n = the number of rows. Then the random perturbation to $\sharp(h)$ should follow the distribution curve whose expected value is $p(h) \cdot m$ and whose standard deviation is roughly $\sqrt{p(h)m}$ if $p(h) \ll 1$.

$$p(h) = \int_{-\infty}^{h} \phi(y)dy \approx \frac{1}{h}\phi(h)$$

is well known as a good approximation. One can easily verify that $p(2\sqrt{\log m}) = 1 - \frac{1}{2\sqrt{\pi \log m} \cdot m^2}$. It means that $\sharp(h)$ can change from its expected value $mp(h)$ by at most

$$2\sqrt{\log m}\sqrt{mp(h)} = 2\sqrt{\log m}\sqrt{m\phi(h)/h} \tag{1}$$

if we guarantee our desired probability $1 - \frac{1}{2\sqrt{\pi \log m} \cdot m^2}$.

Note that the number of columns containing *exactly* $(\lambda + h\sqrt{\lambda q})$ 1's is

$$m \cdot \phi(h)/\sqrt{\lambda q}. \tag{2}$$

It follows that the random perturbation given by (1) can create the "pile" of wrong 1's whose height can be up to

$$(1) \div (2) = 2\sqrt{\frac{\lambda q \log m}{m}} \cdot \frac{1}{\sqrt{h\phi(h)}}$$

Since this pile is created with probability $p(h)$, its average height should be

$$\sum_h 2\sqrt{\frac{\lambda q \log m}{m}} \cdot \frac{p(h)}{\sqrt{h\phi(h)}} = \sum_h \sqrt{\frac{\lambda q \log m}{m}} \cdot \frac{2\sqrt{\phi(h)}}{h\sqrt{h}} \leq \sqrt{\log m} \tag{3}$$

because $m > \lambda q$ and $2\sqrt{\phi(h)}/h\sqrt{h} \leq 1$ for $h \geq 1$.

Now for the second iteration of Step13-16 we can repeat the above discussion by setting $\lambda = \sqrt{\log m}$. At the third iteration, λ is again reduced to $(\log m)^{0.75}/m^{0.5}$ by formula (3), which is enough for us since we can conclude: The probability that the wrong "piles" of height one or more appear is at most

$$\frac{(\log m)^{0.375}}{m^{0.25}} \cdot \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\frac{m^{0.5}}{(\log m)^{0.75}}},$$

which becomes infinitely small for large m.

Thus there are two undesirable phenomena, the antisymmetry of the binomial curve when λ is small and the difference from the theoretical curve caused at random. We have discussed those two independently, which should be combined in a formal proof. One can see, however, that the first phenomenon is important when λ is small and the second when λ is large. Furthermore, by slightly more detailed analysis, we can show each is negligible where we have to consider the other. The detailed proof is left to the full paper. □

References

[1] A. Aggarwal, "Optimal Bounds for Finding Maximum on Array of Processors with k Global Buses",IEEE Trans. Comput., C-35, 1, pp.62-64, 1986.

[2] M. Atallah and S. Kosaraju, "Graph Problems on a Mesh-Connected Processor Array", J. ACM, 31, 3, pp.649-667, 1984.

[3] S. H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus", IEEE Trans. Comput., C-33, 2, pp.133-139, 1984.

[4] H. Imai, personal communication, 1991.

[5] K. Iwama and Y. Kambayashi, "An $O(\log n)$ parallel connectivity algorithm on the mesh of buses", Proc. 11th IFIP World Computer Congress, pp.305-310, 1989.

[6] D. Knuth, "The art of computer programming", Addison-Wesiey, 1973.

[7] H. Lang and M Schimmler and H. Schmeck and H. Schroder, "Systolic sorting on a mesh-connected network", IEEE Trans. Comp., C-34, pp.652-658, 1985.

[8] Scherson and Sen and Shamir, "Shear sort; A true two dimensional sorting technique for VLSI networks", Technical Report, University of California, Santa Barbara, 1985.

[9] C. Schnorr and A. Shamir, "An optimal sorting algorithms for mesh connected computers", 18th ACM Symposium on Theory of Computing, pp.255-263, 1986.

[10] Y. Shiloah and U. Vishkin, "An O(log n) parallel connectivity algorithm", J. Algr., 3, pp-57-67, 1982.

[11] Q. F. Stout, "Mesh-connected Computers with Broadcasting", IEEE Trans. Comput., C-32, 9, pp.826-830, 1983.

[12] Q. Stout, "Meshes with multiple buses", Proc. Proc. 27th IEEE Symp. on Foundations of Computer Science, pp.264-273, 1986.

[13] 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, pp.187-192, 1990.

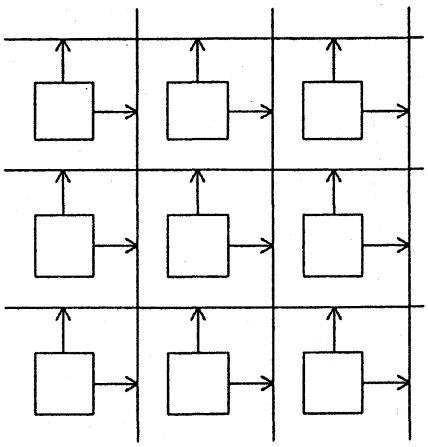

Fig.1

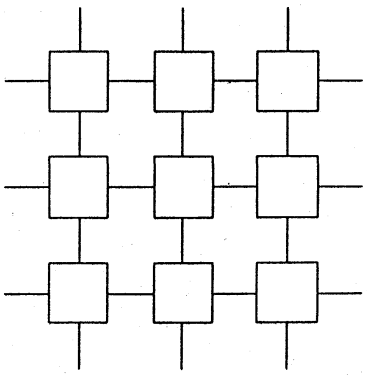

Fig.2

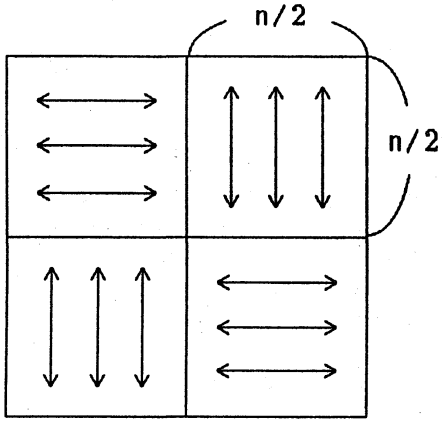

Fig.3

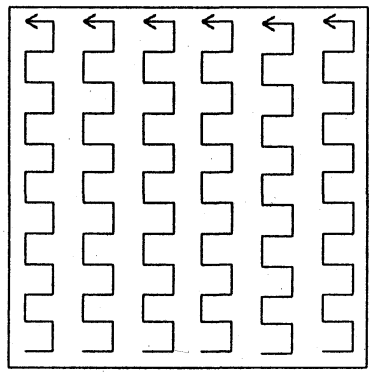

Fig.4

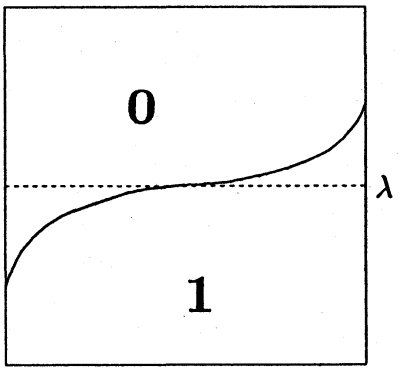

Fig.5

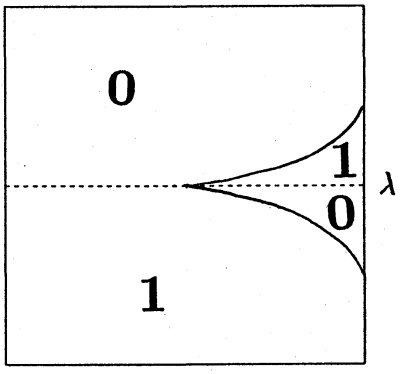

Fig.6

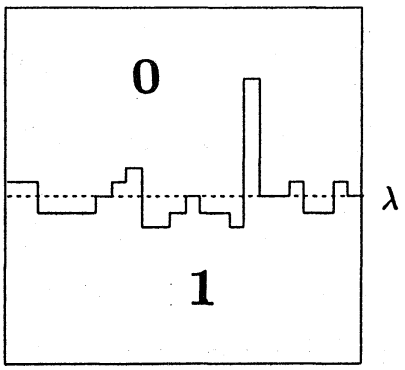

Fig.7

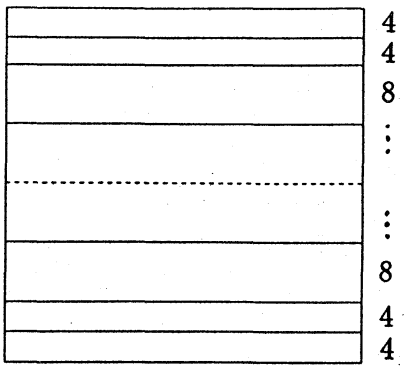

Fig.8

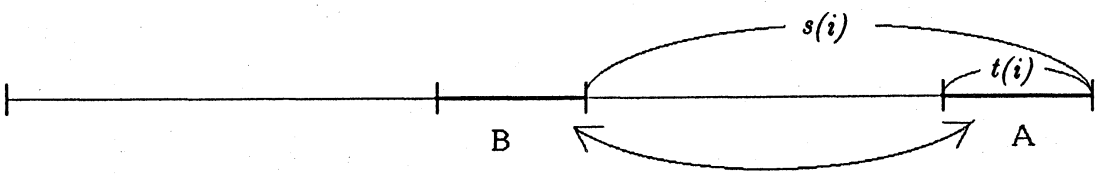

Fig.9

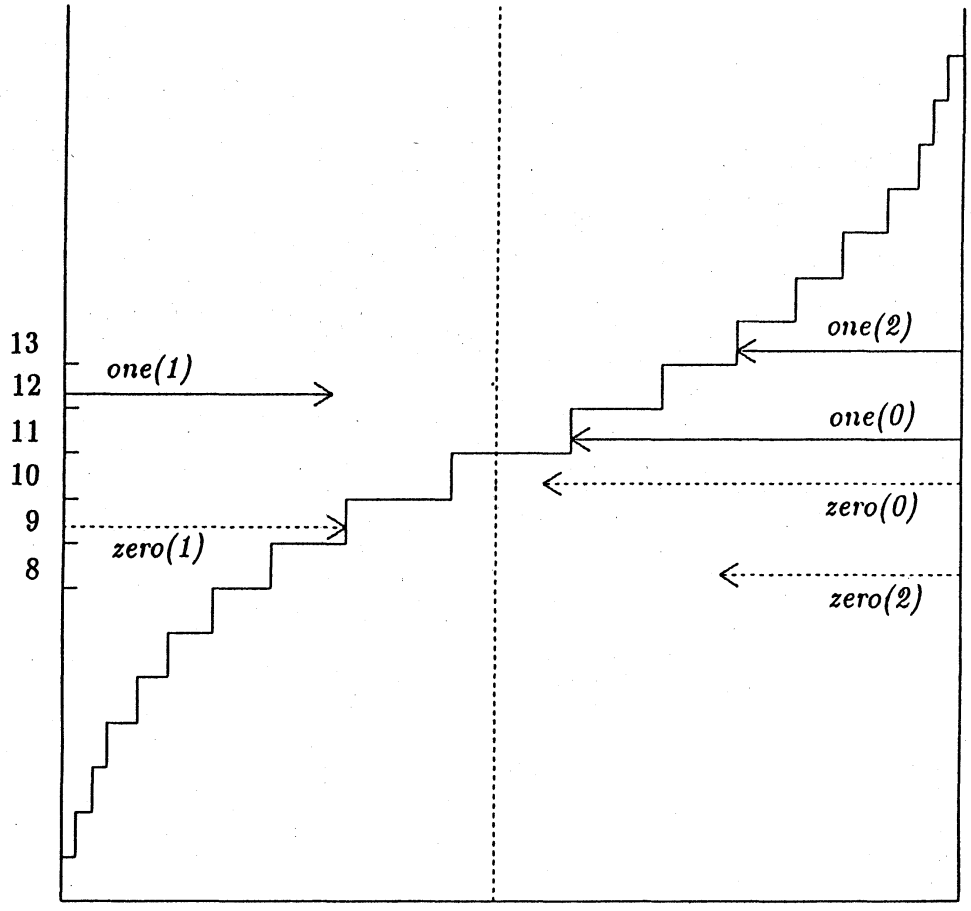

Fig.10

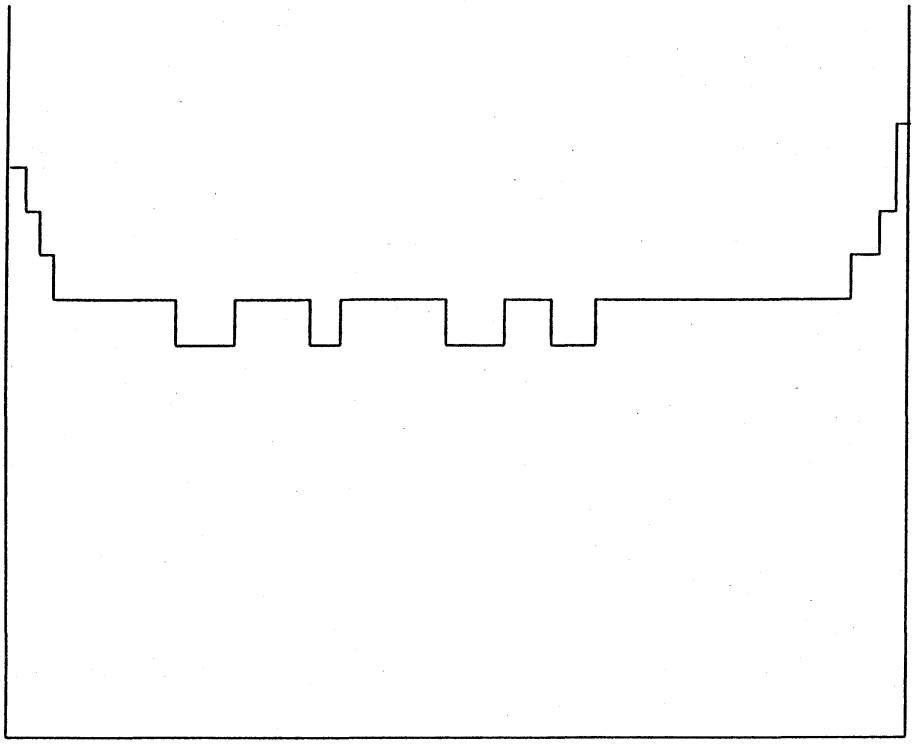

Fig.11

(1) For general data.

n^2	250^2	500^2	10^6	4×10^6	1.6×10^7
T_1	4.00	4.00	4.00	4.00	4.00
T_3	3.34	3.90	4.00	4.00	4.00

(2) For binary data ($\lambda = \frac{n}{2}$).

n^2	10^4	10^6	10^8	4×10^8	1.6×10^9
T_1	3.03	3.17	3.17	3.17	3.27
T_3	2.40	2.90	3.00	2.73	3.16

(3) For binary data ($\lambda = 5$).

n^2	10^4	10^6	10^8	4×10^8	1.6×10^9
T_1	2.77	3.10	3.50	3.47	3.70
T_3	2.60	3.00	3.00	3.00	3.00

Table.1