アルゴリズム 18－2
（1990．11．22）

メッシュバス機械上での並列ソーティングアルゴリズムについて[1]

岩間 一雄　宮野 英次
九州大学工学部

上林 弥彦
京都大学工学部

2次元メッシュ状にバスを配したメッシュバス機械（MB）と呼ばれる並列計算機モデル上で、n^2 個のデータを $O(n)$ 時間でソートする（確率）並列ソーティングアルゴリズムを与える。バスではなく局所通信を用いるメッシュコンピュータ（MC）上では多くの $O(n)$ ソーティングアルゴリズムが知られているが、MB は MC の場合の様に再帰的な分割統治法が使えそうにない（バスは”切れない”）。よって、ソーティングは MB にとって、ほぼ唯一の弱点となっていたが、今回それを一応解決できた。アルゴリズムの基本的アイディアは2項分布曲線の対称性を利用することである。

A Parallel Sorting Algorithm on the Mesh-Bus Machine

Kazuo Iwama and Eiji Miyano
Department of Information Science and Communication Engineering
Kyushu University
Fukuoka 812, Japan

Yahiko Kambayashi
Department of Information Science
Kyoto University
Kyoto 606, Japan

A randomized parallel sorting algorithm which sorts $n \times n$ integers in $O(n)$ time on the mesh-bus machine (MB), is presented. MB is a two-dimensional parallel model having $n \times n$ processors and $2n$ buses (one for each column and row). Many $O(n)$ algorithms are known on not MBs but the mesh computer (MC) that has local communication facilities instead of buses. In the case of MBs, it is not likely that we can use the recursive divide-and-conquer which has been a key technique for fast sorting algorithms. Our new approach needs only row and column operations. The basic idea is to make full use of the probabilistics, symmetry of the binomial density curve.

[1]本研究は一部文部省科学研究費 No.0232047, No.02650278, No.01302059, No.01550294 による。

1. Introduction

Probably no other problems have drawn more attention of both designers and analysts of algorithms than *sorting*. Considerations of different machine models (sequential or parallel, strong parallel or weak parallel), input data (general or restricted integers), or applications (theoretical or practical) generated numerous sorting algorithms based on different ideas. These include the AKS network [1], the parallel merge sort [2] and the optimal sort on mesh-computers [7], to mention just the recent significant developments. In this paper, we present yet another sorting algorithm called *Binomial Curve Sort* or, in short, *BCSORT*. It is a randomized, divide-and-conquer algorithm, that sorts a two-dimensional input in the snake-like order. Time complexity $T(N)$ follows recurrence relation

$$T(N) = c\sqrt{N}T(\sqrt{N}) + O(N).$$

It should be noted that the equation looks a little different from the equation of the usual divide-and-conquer method. We wish to call our method *mince-and-conquer* instead; the problem is partitioned into *many tiny* parts rather than a few smaller ones. Of course, this mince-and-conquer can be used recursively. One can easily verify that (i)$T(N) = o(N^{1+\alpha})$ for any small $\alpha > 0$ or (ii) $T(N) = O(N \log N)$ if $c = 2$. However, that is not our real interest, since, for example, to achieve (ii) for sorting seems much harder than to achieve the similar performance in the usual divide-and-conquer (i.e. $T(N) = 2T(N/2) + O(N)$). We can enjoy the true benefit of mince-and-conquer when we can assume the availability of an algorithm that performs very well, for some reason, on "small" inputs. Suppose, for instance, that the constant c above is 4 and that we have a sorting algorithm which takes N steps for N less than 1000. We can then apply our BCSORT, which will produce a sorting algorithm running in $4N$ steps for N less than 10^6. That is much better than $N \log N$ algorithms, since $\log 10^6 \approx 20 \gg 4$. Our algorithm is optimal in the sense that we can never achieve $o(\sqrt{N})$ for the coefficient of $T(\sqrt{N})$. BCSORT is probably the first nontrivial example of mince-and-conquer achieving optimality.

Such a benefit we really wish comes true in a more direct form when we implement BCSORT on the mesh-connected computer (MC), shown in Fig.1 [9], and the mesh-bus computer (MB) in Fig.2 [3]. Both implementations are straightforward . In the former case, we can sort an $N = n \times n$ array in $O(n)$ (expected) time using *only row and column sort*. (Such a restricted environment is referred to as "a true two dimensional technique" in [6].) The best known upper bound is $O(n \log \log n)$ in [7]. Our present algorithm is, again, optimal (within the difference of constant factors), since the well-known $\Omega(n)$ lower bound for deterministic sorting algorithms on MCs also applies to the randomized case. Note that several $O(n)$ (deterministic) sorting algorithms have been already developed on MCs since [9]. They all need, however, recursive divide-and-conquer (typically, dividing $n \times n$ into four $\frac{n}{2} \times \frac{n}{2}$, then $\frac{n}{2} \times \frac{n}{2}$ into for $\frac{n}{4} \times \frac{n}{4}$, and so on) rather than simple and practical row and column operations.

Non-recursive structure of BCSORT is much more desirable for the second two dimensional model. MBs have no local communication facilities, they only have row and column buses for global communication, which means that the recursive divide-and-conquer as mentioned above is not available (buses cannot be "cut"). Since the sophisticated divide-and-conquer seems essential to $O(n)$ sorting algorithms, we have had a long-time impression that sorting is a typical example to exhibit the limitation of the bus communication. (There is also a fairly large literature discussing the bus-communication from rather negative angles; see e.g., [8]). Our new $O(n)$ algorithm based on BCSORT removes this major weak point of MBs. Although we have no formal proof, the algorithm seems to be optimal, once again, since it is quite likely that all n^2 (or almost all) elements have to be put on the $2n$ buses at least once. It should be noted that, for most of other problems, MBs are at least as good as MCs. For example, MBs exhibit performance similar to even the most powerful PRAMs (logarithmic time) for several graph problems including the connectivity [3], which MCs can by no means achieve.

As its name suggests, the key idea of BCSORT is to make a full use of the symmetry of the binomial density curve. The crucial point is to manage the "anti-symmetry" of the binomial curve when the probability is very low. We also need a zero/one-principle specific for our algorithm. We will finally present a simulation result.

2. The Algorithm and Its Behavior

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 (see Fig.3 for 4×4). To *randomize a row* (*a column*) means to permute the elements of 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.

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 [6]). 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, it takes $n \times T(n)$ time by sequential models and $T(n)$ time by MBs or MCs, where $T(n)$ is the time complexity necessary to sort a single row or column. In both MBs and MCs, $T(n) = O(n)$. For MCs, see e.g., [5]. For MBs, a method called "counting sort" is available: Each element is placed on the bus in a regular order while the other elements know its position 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. 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.4. 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.5. 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. Two iterations of Steps 3-6 would be thus sufficient. Otherwise Step 6 plays its role. Suppose that we accidentally encounter a tall "pile" of wrongly-placed 1's as shown in Fig.6. 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.5 or Fig.6 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 :

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. 7).

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.8. 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 forth to the center. In the third iteration, the borders are shifted two forths, they are shifted three forths 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:

$$s(0) = 0,$$

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

The values of $t(i)$ will be defined precisely in the next section. 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 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 preceeding 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.

3. Justification

Main Theorem. BCSORT stops at the second 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. We first modify the original zero/one-principle [4] to fit our probabilistic environment.

Lemma 1. Suppose that BCSORT 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. 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, $(a_1, \ldots, a_{n^2})$ of integers 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 is turned to

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

exactly as the original zero/one-principle 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,

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

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. Q.E.D.

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

Proof (Sketch). We assume that the input contains exactly λn 1's for an integer λ, and further more that $\lambda \leq \frac{n}{2}$. Extension to the general case is not difficult and may be omitted. Steps 1 and 2 distribute the 1's evenly on the two-dimensional plain. Clearly, the probability that each cell on the plain contains a 1 is $\frac{\lambda n}{n^2} = \frac{\lambda}{n}$. Then we can write the probability that a single column (n cells) contains k 1's as

$$\binom{n}{k} \left(\frac{\lambda}{n}\right)^k \left(1 - \frac{\lambda}{n}\right)^{n-k}$$

That is so-called the binomial distribution, which we denote by $b(k, n, \frac{\lambda}{n})$. Since we discuss the asymptotic behaviors, we can use the Poisson approximation

$$p(k, \lambda) = e^{-\lambda} \frac{\lambda^k}{k!}$$

for $b(k, n, \frac{\lambda}{n})$. Another approximation is 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.9. This figure illustrates the case for $\lambda = 10$, namely, the vertical direction is hugely magnified and only its very bottom is shown. 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 are placed where 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 9 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. (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, (and of course $zero(\lambda + 1)$ does not exists but $one(\lambda + 1)$ has some value). Therefore, after the column sorting, the configuration can be shown as in Fig.10. 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.10, 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 shfting 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 coincide the theoretical distribution of the probabilities. Suppose that $\lambda = \frac{n}{2}$ and consider the probability that a row contains $(\lambda + \sqrt{\frac{\lambda}{2}})$ 1's or more ($\sqrt{\frac{\lambda}{2}}$ is the standard deviation). The normal approximation easily gives us about 0.16 as this probability. (If $\sqrt{\frac{\lambda}{2}}$ is doubled, i.e., if $\lambda + \sqrt{\frac{\lambda}{2}}$ is increased to $\lambda + 2\sqrt{\frac{\lambda}{2}}$, then this value decreases to some 0.02.) That means we can again assume the binomial distribution for the number of columns that contains $(\lambda + \sqrt{\frac{\lambda}{2}})$ 1's or more which we denote by $U(n, \lambda, \sqrt{\frac{\lambda}{2}})$. Exactly as above we can conclude that $U(n, \lambda, \sqrt{\frac{\lambda}{2}})$ can differ from its expected value $m \times 0.16$ by the amount of, say $2\sqrt{m \times 0.16 \times 0.84}$, with probability 0.02. (For a while, we use m instead of n for the number of columns.)

Let $\Phi(x) = \int_{-\infty}^{x} \phi(y)dy$. Then

$$1 - \Phi(x) \approx \frac{1}{x}\phi(x)$$

is known as a good approximation. Let $x = 2\sqrt{\log m}$. Then, $1 - \Phi(x) = \frac{1}{2\sqrt{\pi \log m} \cdot m^2}$, which means, in general, that the number of occurrences of some random variable can differ from its expected value by at most $2\sqrt{\log m}\times$ (its standard deviation) with probability $1 - \frac{1}{2\sqrt{\pi \log m} \cdot m^2}$. This is applied to above $U(n, \lambda, h\sqrt{\frac{\lambda}{2}})$, the number of columns containing at least $(\lambda + h\sqrt{\frac{\lambda}{2}})$ 1's ($\lambda = \frac{n}{2}$). Note that the probability that such a column exists is $p_h = 1 - \Phi(h)$. Its

standard deviation is therefore about $\sqrt{mp_h}$ if we assume $h \geq 2$ or $p_h \ll 1$. Now we can conclude that $U(n, \lambda, h\sqrt{\frac{\lambda}{2}})$ can differ from mp_h (= the expected value) by at most

$$2\sqrt{\log m}\sqrt{mp_h} = 2\sqrt{\log m}\sqrt{m}\frac{\phi(h)}{h}$$

with probability $1 - \frac{1}{2\sqrt{\pi \log m \cdot m^2}}$.

We next calculate the expected value of the number of columns containing *exactly* $(\lambda + h\sqrt{\frac{\lambda}{2}})$ 1's, which is

$$\frac{1}{\sqrt{\frac{\lambda}{2}}}\phi(h)m = 2\frac{m}{\sqrt{n}}\phi(h).$$

It then follows that the random perturbation of $U(n, \lambda, h\sqrt{\frac{\lambda}{2}})$ can change the number of 1's of some column (if the rows are sorted as in Fig.9) up to about

$$\frac{2\sqrt{\log m}\sqrt{m}\frac{\phi(h)}{h}}{\frac{2m}{\sqrt{n}}\phi(h)} = \frac{\sqrt{n \log m}}{h\sqrt{m}}$$

from its expected value. Since $m = n$, this value is $\frac{\sqrt{\log n}}{h}$. One can see that this amount of the perturbation generates the pile of 1's (or 0's) whose height is $\frac{\sqrt{\log n}}{h}$, or at most $\sqrt{\log n}$, after Step 13. Those piles are fairly high and the algorithm will not stop in Step 14 when it first comes there. However, in the next iteration of Step 13 -16, we can assume that we have a hypothetical layer of width $2\sqrt{\log n}$ in which almost equal number of 0's and 1's exist. (Thus we are assuming that the 1's piles of height $i(\leq \sqrt{\log n})$ and the 0's piles of height $-j$ are generated with an equal probability if $i = j$. We may need more careful analysis for this assumption, or, if we take a safety side, we may need to apply the layering technique also for Steps 13 - 16). Then by the same discussion as before, we can conclude the length of piles is much less than one with the same probability as before. (Substitute $2\sqrt{\log m}$ into n and n into m.) Thus the algorithm will stop in Step 14 and the lemma follows.

Recall that we assumed that $\lambda = \frac{n}{2}$. Fortunately, that is the worst case for us since the largest length of the piles decreases as λ decreases. For, the perturbation of $U(n, \lambda, h\sqrt{\frac{\lambda}{2}})$ depends on λ very little (none if we assume the normal approximation) while the expected number of the columns having $(\lambda + h\sqrt{\frac{\lambda}{2}})$ 1's increases.

Thus there are two undesirable phenomena, the antisymmetry of the binomial curve when λ is small and the similar antisymmetry caused at random. We have discussed those two independently, which should be combined for a more 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. Q.E.D

4. Simulation

Our simulation is not for complete BCSORT but for its simplified version (denoted by BC_1 below) given in Section 2 and for BC_2 that is BC_1 with the following modification. After Step 5, we add Step 5.5, which, after sorting all rows to the right, shifts the right (left) most elements 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. T_i below denotes the average number of iterations of Step 3 - 6 before BC_i stops. The number of trials is at least 50. Input data was generated at random.

(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_2	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_2	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_2	2.60	3.00	3.00	3.00	3.00

References

[1] M. Ajtai and J. Komlos and E. Szemeredi, "An $O(n \log n)$ sorting network", 15th ACM Symposium on Theory of Computing, 83, pp.1-9.

[2] R. Cole, "Parallel merge sort", Proc. 27th IEEE Symp. on Foundations of Computer Science, 86, pp.511-516.

[3] K. Iwama and Y. Kambayashi, "An $O(\log n)$ parallel connectivity algorithm on the mesh of buses", Proc. 11th IFIP World Computer Congress, 89, pp.305-310.

[4] D. Knuth, "The art of computer programming", Addison-Wesiey, 73.

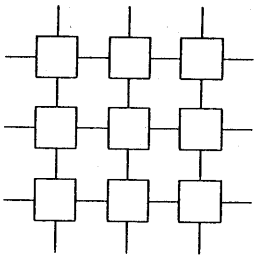
[5] H. Lang and M Schimmler and H. Schmeck and H. Schroder, "Systolic sorting on a mesh-connected network", IEEE Trans. Comp., 85, vol.c-34, pp.652-658.

[6] Scherson and Sen and Shamir, "Shear sort; A true two dimensional sorting technique for VLSI networks", University of California, Santa Barbara, 85, Technical Report.

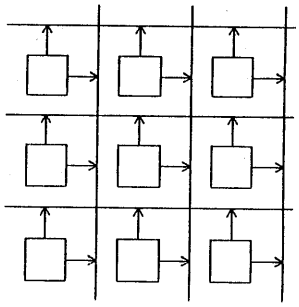
[7] C. Schnorr and A. Shamir, "An optimal sorting algorithms for mesh connected computers", 18th ACM Symposium on Theory of Computing, 86, pp.255-263.

[8] Q. Stout, "Meshes with multiple buses", Proc. Proc. 27th IEEE Symp. on Foundations of Computer Science, 86, pp.264-273.

[9] C. Thompson and H. Kung, "Sorting on a mesh-connected parallel computer", Comm. ACM, 77, vol.20, pp.263-271.



F i g . 1



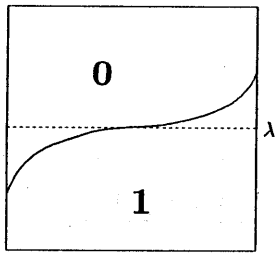
F i g . 2

1	7	1 1	3
1 2	1 4	8	1 3
5	2	1 6	1 5
6	4	1 0	9

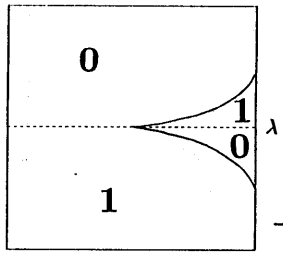
→

1	2	3	4
8	7	6	5
9	1 0	1 1	1 2
1 6	1 5	1 4	1 3

F i g . 3



F i g . 4



F i g . 5

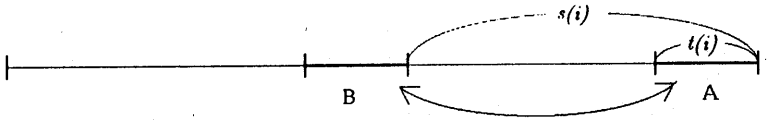


Fig. 8

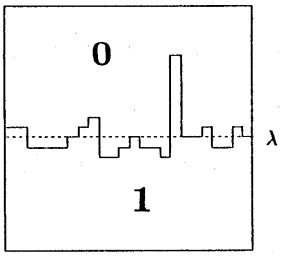


Fig. 6

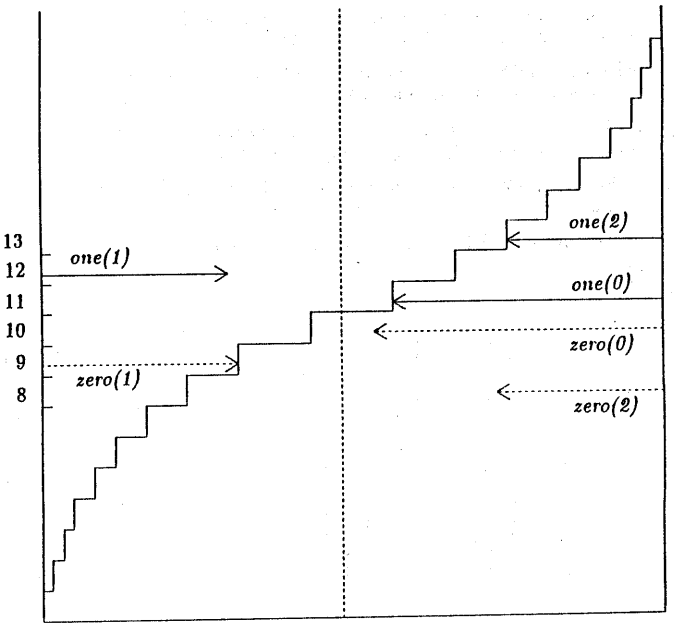


Fig. 9

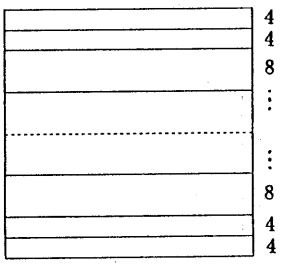


Fig. 7

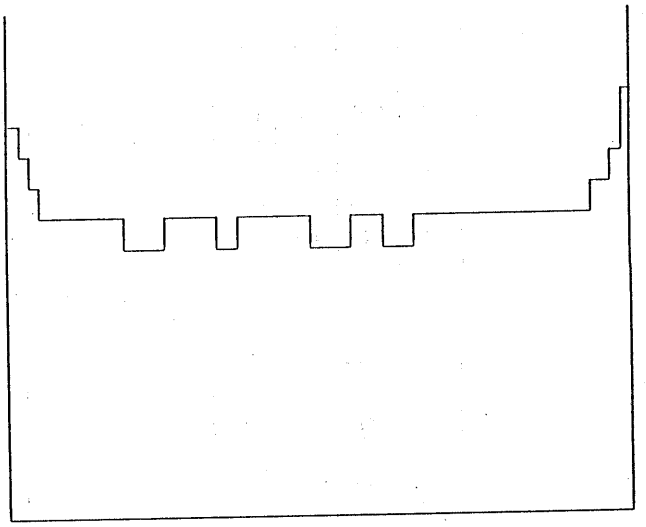


Fig. 10