SIAM J. COMPUT.
Vol. 15, No. 1, February 1986

EFFICIENT SIMULATIONS AMONG SEVERAL MODELS OF PARALLEL COMPUTERS*

FRIEDHELM MEYER AUF DER HEIDE†

Abstract. A parallel computer (PC) with fixed communication network is called fair if the degree of this network is bounded, otherwise it is called unfair. In a PC with predictable communication each processor can precompute the addresses of the processors it wants to communicate with in the next t steps in $O(t)$ steps. For an arbitrary $\varepsilon > 0$ we define fair PC's M and M' with $O(n^{1+\varepsilon})$ processors each. $M(M')$ can simulate each unfair PC with predictable communication and $O(\log(n))$ storage locations per processor (each fair PC) with n processors with constant time loss. M' improves a result from [Acta Informatica, 19 (1983), pp. 269-296] where a time loss of $O(\log\log(n))$ was achieved. Assuming some reasonable properties of simulations we finally prove a lower bound $\Omega(\log(n))$ for the time loss of a fair PC which can simulate each unfair PC. Applying fast sorting or packet switching algorithms (Proc. 15th Annual ACM Symposiums on Theory of Computing, Boston, 1983, pp. 1-9; 10-16; Proc. ACM Symposiums on Principles of Distributed Computing, Ottawa, 1982) one sees easily that this bound is asymptotically tight.

Key words. parallel computers, general purpose machines, simulations

1. Introduction. In this paper we deal with the following question: How efficiently can one parallel computer (PC) with fixed communication network with bounded degree simulate all members of a certain class of PC's?

By a PC we mean a finite set of n processors which have the usual sequential capabilities. They are partially joined by wires. The graph defined by the processors and the wires is its communication network. In one step each processor is allowed to read a piece of information from a (relative to the communication network) neighboring processor. We allow several processors to read from the same processor at the same time. We assume the PC is synchronized.

Technological restrictions demand the degree of a PC, i.e. the degree of its communication network, to be bounded by a small constant.

We shall call such PC's fair. Those with large degree we call unfair. Later we shall always assume that their degree is $n-1$, i.e. that their communication network is the complete graph. We furthermore assume that each processor only has $O(\log(n))$ storage locations, each able to store one integer.

An important class of unfair PC's are those with predictable communication (unfair PC's with pred. com.).

Such a PC has the additional property that for each integer t, each processor can compute for itself the sequence of addresses of processors it wants to read from during the next t steps in $O(t)$ steps.

Famous examples of unfair PC's with pred.com. are the ascend- and descend-programs for cubes defined by Preparata and Vuillemin in [5]. Such a cube is an unfair PC with $N = 2^k$ processors, and its communication network is a k-dimensional cube. The prediction of the communication is very easy for each processor: Neighbour in direction of the first dimension, neighbour in the direction of the second dimension, and so on.

Preparata and Vuillemin could simulate this special, very regular PC with pred.com. by a fair PC with N processors, the Cube-Connected Cycles, and constant time loss.

* Received by the editors May 19, 1983, and in revised form October 1, 1984.

† Fachbereich Informatik, Johann Wolfgang Goethe-Universität, 6000 Frankfurt a.M., Federal Republic of Germany.

The aim of this paper is to determine the efficiency of the following types of fair PC's:

- n-universal PC's. They can simulate each fair PC with n processors and fixed degree.
- n-simulators. They can simulate each unfair PC with pred.com. and n processors.
- general n-simulators. They can simulate each unfair PC with n processors.

In [3] an n-universal PC with $O(n)$ processors and time loss $O(\log(n))$ is constructed. In [4] an n-universal PC with $O(n^{1+\varepsilon})$ processors for some arbitrary $\varepsilon > 0$ is constructed which has only time loss $O(\log\log(n))$.

In the third chapter of this paper this last result is improved by presenting an n-universal PC with the same number of processors as above but with constant time loss only.

In the fourth chapter we refine the ideas of the simulation of the third chapter to obtain an n-simulator. It also has $O(n^{1+\varepsilon})$ processors for some arbitrary $\varepsilon > 0$ and constant time loss.

For the above constructions we need a PC which can execute packet switching in a much more general way than for example permutation networks. In the second chapter, such PC's, so-called (a, b)-distributors, are introduced.

The last section of this paper shows that we cannot hope for fast simulations if we want to construct a general n-simulator. We prove a lower bound $\Omega(\log(n))$ for the time loss of a general n-simulator, independent on the number of its processors. This result holds when assuming some reasonable properties of the design of simulations as they are already defined in [4] for proving a time-processor trade-off for n-universal PC's.

By the results of Ajtai, Komlos and Szemeredi [1] or Reif and Valiant [6] one can show that this lower bound is tight within a constant factor.

The above authors have developed fair sorting PC's which can sort n numbers using only $O(\log(n))$ steps (with overwhelming probability in the case of Reif and Valiant).

One can easily see that with such a fair PC one can simulate one step of an unfair PC in $O(\log(n))$ steps, using such a fair sorting PC as "post-office" for transporting the respective informations between the processors. The same result can be achieved by using the parallel packet switching algorithm for Cube-Connected Cycles presented by Upfal in [7] who generalizes the corresponding algorithm for cubes due to Valiant and Brebner [8].

2. The construction of (a, b)-distributors. In this section we construct PC's which are able, in a more general way than permutation networks, to broadcast information. They are constructed similarly to the distributors shown in [4]. These PC's are needed for the constructions of the n-universal PC and the n-simulator in the next sections.

Let a, b be integers, $a \leqq b$. An *(a, b)-distributor* $D_{a,b}$ is a fair PC which has $a+b$ distinguished processors, a input processors $A_1, \cdots, A_a$ and b output processors $B_1, \cdots, B_b$. It has the following property:

If each B_i, $i \in \{1, \cdots, b\}$, has stored an integer $c_i \in \{1, \cdots, a+1\}$, then $D_{a,b}$ can initialize itself such that afterwards the following holds:

If each A_j, $j \in \{1, \cdots, a\}$, contains an integer string x_j of length $O(\log(n))$, then $D_{a,b}$ can distribute $(x_1, \cdots, x_a)$ according to $(c_1, \cdots, c_b)$, i.e. can transport each x_j, $j \in \{1, \cdots, a\}$, to each B_i with $c_i = j$, $i \in \{1, \cdots, a\}$, in $O(\log(b) + \log(n))$ steps. The above initialization is called the initialization of $D_{a,b}$ for $(c_1, \cdots, c_b)$, and the time it needs is the initialization time of $D_{a,b}$.

We now present an (a, b)-distributor based on the well-known Waksman permutation network (see [9]).

Let $b' := 2^{\lceil \log(b) \rceil}$. The b'-Waksman permutation network $W_{b'}$ is a fair PC with b' input processors $A_1, \cdots, A_{b'}$ and b' output processors $B_1, \cdots, B_{b'}$. For each partial permutation Π on $\{1, \cdots, b'\}$, $W_{b'}$ can initialize itself for Π such that afterwards there are marked pairwise disjoint paths of length $O(\log(b'))$ in $W_{b'}$ form A_i to $B_{\Pi(i)}$, $i \in \{1, \cdots, b'\}$.

In [3], Galil and Paul have shown that this initialization of $W_{b'}$ for Π can be executed in $O(\log(b')^4)$ steps. Furthermore it is well known that Batcher's sorting algorithm [2] can be implemented on $W_{b'}$ and sorts b' numbers in $O(\log(b')^2)$ steps. $W_{b'}$ has $2b' \log(b')$ processors. W_8 is shown in Fig. 1.

We now shall insert some additional wires in $W_{b'}$ as illustrated for W_8 in Fig. 2. The resulting network we call $\bar{W}_{b'}$.

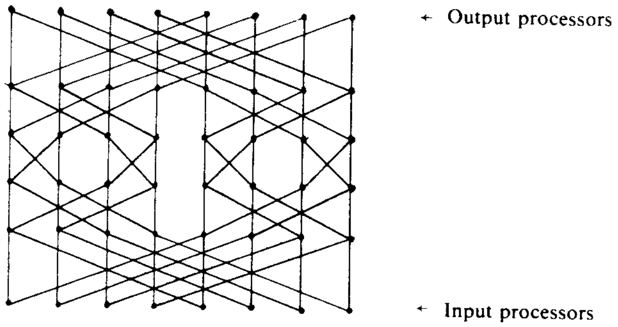

FIG. 1. *The fair* PC W_8.

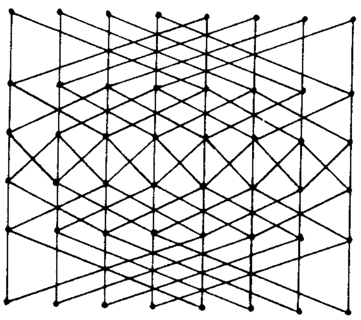

FIG. 2. *The fair* PC $\bar{W}_8$.

In addition to the capabilities of $W_{b'}$ this network can do the following: If for $i \in \{1, \cdots, b'\}$, A_i contains a number c_i, $c_1 \leqq \cdots \leqq c_{b'}$, $\bar{W}_{b'}$ can mark pairwise disjoint trees of depth $2 \cdot \log(b') - 1$ in $\bar{W}_{b'}$, one for each $x \in \{c_1, \cdots, c_b\}$. The leaves of a tree belonging to some such x are those A_i with $c_i = x$, its root is that B_i with $i = \min\{i', c_{i'} = x\}$. The existence and construction of such trees is obvious; an example is shown in Fig. 3.

We shall use these trees for transporting data from one input to many output processors along the paths in the trees. For $i \in \{1, \cdots, b\}$ let $c_i \in \{1, \cdots, a+1\}$ be stored in B_i. The following algorithm will mark a trees $G_1, \cdots, G_a$ in $\bar{W}_{b'}$. For $i \in \{1, \cdots, a\}$, the root of G_i is A_i and its leaves are all B_j's with $c_j = i$. G_i has depth

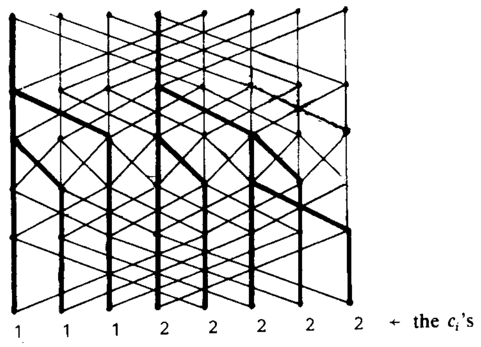

FIG. 3. *$\bar{W}_8$ with marked trees for* $(c_1, \cdots, c_8) = (1, 1, 1, 2, 2, 2, 2, 2)$.

$O(\log(b'))$. Each processor of $\bar{W}_{b'}$ lies on at most three such trees. Clearly after such a marking $\bar{W}_{b'}$ can distribute a strings each of length $O(\log(n))$ according to $(c_1, \cdots, c_b)$ in $O(\log(b') + \log(n))$ steps by sending them along the paths of the respective trees. Thus $\bar{W}_{b'}$ is initialized for $(c_1, \cdots, c_b)$.

The algorithm works as follows.

Part 1. $\bar{W}_{b'}$ sorts $c_1, \cdots, c_b$ to the sequence $d_1, \cdots, d_b$.

Part 2. $\bar{W}_{b'}$ initializes itself for the partial permutation which maps i to j, if $d_i = c_j$, $i \in \{1, \cdots, a\}$.

Remark 1. Now there are pairwise disjoint paths marked for each $i \in \{1, \cdots, b\}$ from B_i to the A_j where c_i is transported to in Part 1. For $i \in \{1, \cdots, a+1\}$ let s_i denote the smallest $j \in \{1, \cdots, b\}$ with $d_j = i$. Then for each $i \in \{1, \cdots, a\}$, $d_j = i$ for each $j \in \{s_i, \cdots, s_{i+1} - 1\}$.

Part 3. Mark a pairwise disjoint trees in $\bar{W}_{b'}$. For $i \in \{1, \cdots, a\}$, the ith tree has the root B_{s_i} and the leaves $A_j, j \in \{s_i, \cdots, s_{i+1} - 1\}$.

Remark 2. As shown above such trees can be marked in $O(\log(b'))$ steps. Each tree has depth $2 \cdot \log(b') - 1$.

Part 4. $\bar{W}_{b'}$ initializes itself for the partial permutation which maps s_i to i, $i \in \{1, \cdots, a\}$.

For $i \in \{1, \cdots, a\}$, G_i now is the tree which consists of the ith path from Part 4, the ith tree from Part 3 and the jth paths from Part 2, $j \in \{s_i, \cdots, s_{i+1} - 1\}$.

By the explanation above we know that we hereby have initialized $\bar{W}_{b'}$ for $(c_1, \cdots, c_b)$. Furthermore we know that Part 1 can be executed in $O(\log(b')^2)$ steps, Part 3 in $O(\log(b'))$ steps and Part 2 and 4 in $O(\log(b')^4)$ steps each. Thus the initialization time of $\bar{W}_{b'}$ is $O(\log(b')^4)$.

THEOREM 1. *Let a, b be integers, $a \leqq b$, $b' = 2^{\lceil \log(b) \rceil}$. $\bar{W}_{b'}$ is an (a, b)-distributor with $O(b \log(b))$ processors and initialization time $O(\log(b)^4)$.*

Without proof we will point out two possible improvements of this theorem. We can construct (a, b)-distributors with initialization time $O(\log(b))$ if we are able to sort b numbers in $O(\log(b))$ steps. Ajtai, Komlos and Szemeredi [1] have done so with the help of a fair PC with $O(b \log(b))$ processors. This fair PC can also sort packets of length s according to some keys in $O(\log(b) + s)$ steps. With this result we can construct an (a, b)-distributor with $O(b \log(b))$ processors and initialization time $O(\log(b))$. A similar construction can be found in [10].

A similar result can be achieved when using the sorting algorithm from [6] due to Reif and Valiant. They have sorted b numbers on Cube-Connected Cycles using $O(\log(b))$ steps with overwhelming probability. In order to sort packets of length

$O(\log(n))$ we here need $O(\log(n))$ such fair PC's in order to do so in $O(\log(b)+\log(n))$ steps. Thus we obtain an (a, b)-distributor with $O(b\log((n))$ processors and initialization time $O(\log(b))$ which allows distributions using $O(\log(b)+\log(n))$ steps with overwhelming probability.

3. The construction of an *n*-universal PC.

In this section we will construct an n-universal PC M_0, that means a fair PC which can simulate each fair PC with n processors and fixed degree c.

Let H be a fair PC with n processors $R_1, \cdots, R_n$ and communication network G.

The idea of our simulation is as follows. We construct a fair PC D_t^* which can simulate H for t steps, if it is prepared suitably. Furthermore we shall see that we can apply an (a, b)-distributor to prepare D_t^* before each phase of t simulated steps as demanded above. We shall choose t such that a preparation needs $O(t)$ steps. Thus t steps of H are simulated in $O(t)$ steps which yields constant time loss.

We shall now construct D_t^*. It consists of n copies $D_t^1, \cdots, D_t^n$ of a fair PC D_t, whose communication network is a complete c-ary tree of depth t. We now show how to initialize D_t^* such that it can simulate t steps of H. First we attach to each processor P of D_t^* an address $l(P)$ of the processor of H, P has to simulate.

For $i \in \{1, \cdots, n\}$ the root P of D_t^i gets $l(P)=i$. If R_i has $c' \leqq c$ neighbours $R_{i_1}, \cdots, R_{i_{c'}}$ in H, the first c' neighbours $P_1, \cdots, P_{c'}$ of P get $l(P_j)=i_j$, $j \in \{1, \cdots, c'\}$. This attachment is completed in the obvious way.

Let K be a configuration of H represented by the tuple $(K_1, \cdots, K_n)$ of configurations of the processors of H. We say, D_t^* is prepared for K, if each processor P of D_t^* for which $l(P)$ is defined contains $K_{l(P)}$. We say D_t^* simulates t steps of H started with K, if it computes K_i' in the root of D_t^i, $i \in \{1, \cdots, n\}$, where $K'=(K_1', \cdots, K_n')$ is the tth successor-configuration of K.

LEMMA 2. *If D_t^* is prepared for K, it can simulate t steps of H started with K in $O(t)$ steps.* □

The proof can be done by induction on t and is obvious. Now suppose that D_t^* has simulated t steps of H started with some configuration K. We now have to prepare D_t^* for K', the tth successor configuration of K. We know that for each $i \in \{1, \cdots, n\}$, the root processor of D_t^i has computed K_i', but the other processors P of D_t^* with $l(P)=i$ have not computed this configuration because they have earlier stopped simulating. Therefore the root processor of D_t^i has to inform these processors about K_i'. Applying an idea from [4] it suffices to transport the string Info(K, R_i, t) of numbers R_i reads from neighbours during t steps of H started with K. As a processor P of D_t^* with $l(P)=i$ knows K_i, it can, with the help of Info(K, R_i, t), compute K_i' in $O(t)$ steps. Info(K, R_i, t) has length at most t. Thus we need a network which transports Info(K, R_i, t) from the root processor of D_t^i to each processor P with $l(P)=i$, $i=1, \cdots, n$. Let m be the number of processors of D_t^*, then the above is exactly what an $(n, m-n)$-distributor as defined and constructed in the last section can do, if we identify its input processors with the root processors of $D_t^1, \cdots, D_t^n$ and its output processors with the other processors of D_t^*. This fair PC we call M_0.

Thus the exemplary of D_t^* in M_0 can be prepared for K' needing $O(\log(m-n)+t)$ steps for distributing Info(K, R_i, t)'s and $O(t)$ steps for computing K_i' in each processor P with $l(P)=i$, $i=1, \cdots, n$. Therefore, the preparation needs $O(\log(m-n)+t)+O(t)$ steps. Thus $O(\log(m-n)+t)$ steps are necessary to simulate t steps of H.

We now choose $t=\lfloor \varepsilon \log_c(n) \rfloor$ for some arbitrary $\varepsilon>0$. Then M_0 has $O(n^{1+\varepsilon}\log(n))$ processors and needs $O((1+\varepsilon)\log(n)+\varepsilon\log(n))=$

$O((1+2\varepsilon)\log(n))$ steps for simulating $\lfloor \varepsilon \log_c(n)\rfloor$ steps of H. Thus its time loss is $O((1+2\varepsilon)/\varepsilon)$.

THEOREM 3. *M_0 is an n-universal* PC *with $O(n^{1+\varepsilon}\log(n))$ processors and time loss $O((1+2\varepsilon)/\varepsilon)$ (which is a constant for fixed $\varepsilon>0$).*

4. The construction of an *n*-simulator.

In this section we shall construct an n-simulator. The basic idea of the construction is similar to the one in the last section. Let H be an unfair PC with pred.com. and n processors $R_1, \cdots, R_n$. We again describe a fair PC T_t^* which can simulate t steps of H in $O(t)$ steps if it is prepared in a suitable way. But in this case we see that we are not able to prepare T_t^* in $O(\log(n))$ steps, if we chose $t=O(\log(n))$. The reason is that in the case of an n-simulator, we may not demand that a processor of T_t^* simulates the same processor of H all over the simulation. Therefore we have to inform the processors of T_t^* after each phase of t simulated steps which processors of H they have to simulate now. Although it is possible to compute fast which processor of H has to be simulated by which processor of T_t^*, it turns out that broadcasting this information needs $\Omega(\log(n)^2)$ steps if $t=O(\log(n))$. The idea of how to solve this problem is the following. We shall execute an initialization of T_t^* each time before d phases of simulating t steps of H. This initialization will not be much slower than one preparation and will guarantee that afterwards d preparations can be done fast. Thus we obtain constant time loss by choosing d in an appropriate way. For simplifying the description of the simulation we shall use d copies of T_t^* in our simulation, each responsible for one of the d phases between two initializations. Each of these exemplaries, together with some (a, b)-distributors, will be called a weak n-simulator.

First we describe the fair PC T_t^* for some fixed integer t. T_t^* can simulate t steps of each unfair PC with pred.com. in $O(t)$ steps if it is prepared in an appropriate way. T_t^* consists of n exemplaries $T_t^1, \cdots, T_t^n$ of a fair PC T_t which we will define now. Its communication network is a tree whose vertices are replaced by cycles. The cycle corresponding to its root is called the root cycle, each processor on it is a root processor and one of them is the main root.

T_t is inductively defined as follows: T_0 consists of one processor, it is its main root and forms its root cycle. For $t>0$, T_t consists of exemplaries of $T_0, \cdots, T_{t-1}$ and t new processors $P_0, \cdots, P_{t-1}$. These processors form the root cycle of T_t by wires between P_p and $P_{(p+1)\text{mod}(t)}$ for $p\in\{0, \cdots, t-1\}$. P_0 is the main root. Furthermore, for each $p\in\{0, \cdots, t-1\}$, P_p is joined to the main root of T_p.

An example of this fair PC is shown in Fig. 4. The following lemma can easily be proved by evaluating the obvious recursion for the number of processors of T_t and by the above definition.

LEMMA 4. *For $t\geqq 1$, T_t has $3\cdot 2^{t-1}-1$ processors and degree* 3.

Now let H be an unfair PC with pred. com. and n processors $R_1, \cdots, R_n$. A configuration $K=(K_1, \cdots, K_n)$ of H consists of configurations K_i for each processor R_i of H, $i\in\{1, \cdots, n\}$. Recall that each processor has only $O(\log(n))$ storage locations. Thus each K_i can be represented by a coding of its program and a list of the contents of its storage locations. This representation is an integer string of length $O(\log(n))$. In the sequel we shall identify this string with the configuration.

Let $\bar{K}=(\bar{K}_1, \cdots, \bar{K}_n)$ be the pth successor-configuration of K. Then for $i\in\{1, \cdots, n\}$, $\bar{K}_i$ is the pth successor-configuration of K for R_i.

For an integer p and $i\in\{1, \cdots, n\}$, Com (K, R_i, p) denotes the string of addresses of processors R_i reads from during p steps of H started with K. For $q\in\{1, \cdots, p\}$, Com $(K, R_i, p)_q$ denotes the qth element of Com (K, R_i, p). If for some such q, H does

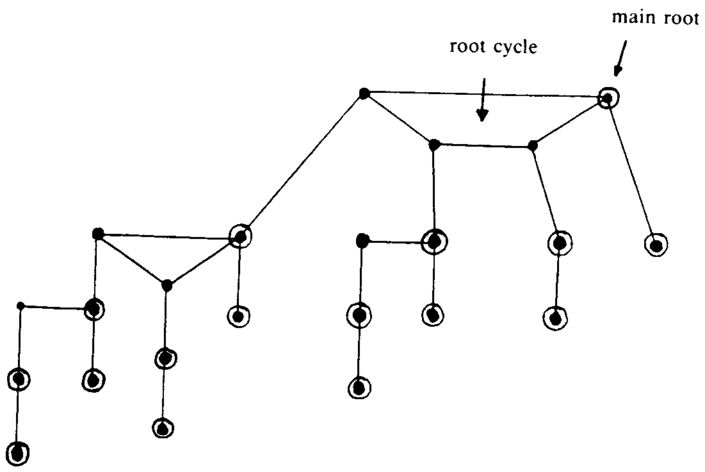

FIG. 4. *The fair* PC T_4.

not read from another processor in the qth step started with K, we assume that $\mathrm{Com}(K, R_i, p) = i$.

Let $i \in \{1, \cdots, n\}$. We say T_t is prepared for K and R_i for t steps, if the following holds:

If $t = 0$ then T_t contains K_i.

Let $t > 0$. Then each root processor contains K_i, and for each $p \in \{0, \cdots, t-1\}$ the exemplary of T_p joint to the pth root processor is prepared for K and R_j for p steps, if $j = \mathrm{Com}(K, R_i, t)_{p+1}$ and $j \neq i$. If $j = i$, there is no condition on T_p.

T_t^* is prepared for K if for each $i \in \{1, \cdots, n\}$ T_t^i is prepared for K and R_i for t steps.

The processor of H being attached by the above preparation to some processor P of T_t^* is said to be represented by P relative to K.

We say T_t^* simulates t steps of H started with K, if T_t^* executes a computation which finishes with the tth successor-configuration of K for R_i in each root processor of T_t^i, $i \in \{1, \cdots, n\}$.

LEMMA 5. *If T_t^* is prepared for K, it can simulate t steps of H started with K in $O(t)$ steps.*

Proof. Let $i \in \{1, \cdots, n\}$ be fixed, $P_0, \cdots, P_{t-1}$ be the root processors of T_t^i. Suppose that T_t^* is prepared for K.

For $p \in \{1, \cdots, t\}$ we say that the root cycle of T_t^i is p-prepared if P_{p-1} and $P_{p \bmod (t)}$ contain the pth successor-configuration of K for R_i and for each $q \in \{1, \cdots, p-1\}$, $P_{(p+q) \bmod (t)}$ contains the $(p-q)$th successor configuration of K for R_i. The root cycle of T_t^i is 0-prepared, if P_0 contains K_i. (This is fulfilled for example, when T_t^* is prepared for K.)

We now want to find an algorithm which transfers a p-prepared root cycle to a $(p+1)$-prepared one. For this purpose we first assume that for each $q \in \{0, \cdots, t-1\}$ the main root Q of the exemplary of T_q joint to P_q contains the qth successor-configuration of K for the processor R_j being represented by Q. Thus Q contains the message R_i wants to read from R_j in the $(q+1)$th step of H started with K.

Now if the root cycle is p-prepared for $p \in \{0, \cdots, t-2\}$, it becomes $(p+1)$-prepared by the following algorithm.

Part 1. For each $q \in \{1, \cdots, p\}$, $P_{(p+q) \bmod (t)}$ simulates the $(p-q+1)$th step of R_i with the help of $P_{(p+q-1) \bmod (t)}$.

Remark 1. As $P_{(p+q-1) \bmod (t)}$ has already executed this step by the definition of "p-prepared", Part 1 can be done in constant time.

Part 2. P_p simulates the $(p+1)$th step of R_i.

Remark 2. This can be done in constant time because we have assumed that the message R_i perhaps wants to read from another processor is stored in the main root of the T_p joint to P_p.

Part 3. For each $q \in \{1, \cdots, p+1\}$, $P_{(p+q) \bmod (t)}$ simulates the $(p-q+2)$th step of R_i with the help of $P_{(p+q-1) \bmod (t)}$.

Remark 3. This works in constant time, because in step 1 (resp. step 2) $P_{(p+q-1) \bmod (t)}$ just has simulated this step.

Thus T_t^* is $(p+1)$-prepared in a constant number s' of steps. Now we may inductively assume that after $s' \cdot p$ steps the root cycle of the exemplary of T_p joint to P_p is p-prepared. But this means that its main root contains the message R_i needs to execute its $(p+1)$th step after $s' \cdot p$ steps.

By our algorithm this message is needed after $s' \cdot p + (\text{time for step 1})$ many steps that means it is available when it is required by P_p. Thus P_0 contains the tth successor-configuration of K for R_i after $s' \cdot t$ steps. Clearly in further $s'' \cdot t$ steps each root processor can have stored this configuration.

Executing this algorithm in parallel for each $i \in \{1, \cdots, n\}$ we have simulated t steps of M started with K in $(s'+s'') \cdot t$ steps. □

Figure 5 shows the states of the p-prepared root cycle of T_8^i for some $i \in \{1, \cdots, n\}$ and each $p \in \{1, \cdots, 8\}$. A number l in the qth column and pth row, $q \in \{0, \cdots, 7\}$, $p \in \{1, \cdots, 8\}$ means: If T_8 is p-prepared, P_q contains the lth successor configuration of K for R_i.

	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
1	$\underline{1}$	1	0	0	0	0	0	0
2	0	$\underline{2}$	2	1	0	0	0	0
3	0	0	$\underline{3}$	3	2	1	0	0
4	0	0	0	$\underline{4}$	4	3	2	1
5	2	1	0	0	$\underline{5}$	5	4	3
6	4	3	2	1	0	$\underline{6}$	6	5
7	6	5	4	3	2	1	$\underline{7}$	7
8	8	7	6	5	4	3	2	$\underline{8}$

FIG. 5. *The design of a p-prepared root cycle.*

In order to obtain a fast simulation of arbitrarily many steps of H we have to prepare T_t^* before each phase of t steps for the appropriate configuration of H. In order to obtain an n-simulator of at most polynomial size we have to choose $t = O(\log(n))$ because of Lemma 4. But then we have to prepare T_t^* before each phase of t steps in $O(\log(n))$ time in order to obtain a constant time loss. We shall see later that such algorithms would need $\Omega(\log(n)^2)$ steps. They have to execute $\Omega(\log(n))$ initializations of (a, b)-distributors sequentially each of which needs $\Omega(\log(n))$ steps (see § 2). Therefore we will execute an initialization each time before d such phases of t steps, where d is chosen suitably. It turns out that this initialization for d preparations can be done in parallel and does not need much more time than one preparation.

The effect of this initialization is that afterwards d preparations can be executed, each in $O(\log(n))$ steps. This trick will guarantee constant time loss. Let in the sequel $\varepsilon > 0$ be fixed and $t := \lfloor \varepsilon \log(n) \rfloor$. Then by Lemma 4, T_t^* has at most $3n^{1+\varepsilon}$ processors.

Now we shall first define a type of fair PC's, so-called weak n-simulators, which will be used for constructing n-simulators. An explicit construction of a weak n-simulator will be given later.

A weak n-simulator M is a fair PC with the following properties.

- M contains an exemplary of T_t^*.
- If $K = (K_1, \cdots, K_n)$ is some configuration of H and for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i contains $\mathrm{Com}(K, R_i, t)$, then M can initialize itself such that afterwards the following holds: If for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i contains K_i, then M can prepare T_t^* for K in $O(\log(n))$ steps.

The above initialization we call the initialization of M for K and the time it needs the initialization time of M. Note that the initialization does not include the preparation of T_t^* but only guarantees that this preparation can be done fast.

We now shall construct n-simulators. Let M be some weak n-simulator with initialization time d. Then the fair PC M^* consists of $r := \lceil d/t \rceil$ exemplaries of M called $M^0, \cdots, M^{r-1}$. For each $l \in \{0, \cdots, r-1\}$, $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^l is joint to the corresponding processor in $M^{(l+1) \bmod (r)}$.

THEOREM 6. *Let M be a weak n-simulator with initialization time d. Then M^* is an n-simulator which can simulate l steps of some arbitrary unfair* PC *with pred. com. and n processors in $O(d + l/\varepsilon)$ steps. If M has m processors, M^* has $\lceil d/t \rceil \cdot m$ processors. (For fixed $\varepsilon > 0$, we thus have achieved constant time loss, if $l = \Omega(d)$.)*

Proof. The computation of the number of processors of M^* is clear. We shall construct an algorithm which simulates $d' := t \cdot r$ steps of H started with $K^0 = (K_1^0, \cdots, K_n^0)$. For $j \in \{1, \cdots, r\}$ let $K^j = (K_1^j, \cdots, K_n^j)$ be the $(t \cdot j)$th successor-configuration of K^0.

Assume that for each $i \in \{1, \cdots, n\}$, $q \in \{0, \cdots, r-1\}$, K_i^q is stored in each root processor of T_t^i in M^q.

Now d' steps of H started with K^0 can be simulated as follows.

Part 1. For each $q \in \{0, \cdots, r-1\}$, $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^q computes $\mathrm{Com}(K^q, R_i, t)$.

Remark 1. This can be done in $O(r \cdot t) = O(d')$ steps because of the definition of predictable communication.

Part 2. For each $q \in \{0, \cdots, r-1\}$, M^q initializes itself for K^q.

Remark 2. This can (after having executed Part 1) be done in d steps as d is the initialization time of the M^q's.

Part 3. For $q = 0, \cdots, r-1$ do (sequentially)

```
Begin
    a) $M^q$ prepares the exemplary $T'$ of $T_t^*$ in $M^q$ for $K^q$.
    b) $T'$ simulates $t$ steps of $H$ started with $K^q$.
       Comment. Now for each $i \in \{1, \cdots, n\}$ each root processor of $T_t^i$ in $T'$
       contains $K_i^{(q+1)}$.
    c) For each $i \in \{1, \cdots, n\}$, each root processor of $T_t^i$ in $T'$
       transports $K_i^{(q+1)}$ to the corresponding processor in $M^{(q+1) \bmod (r)}$.
End
```

Remark 3. Now for each $i \in \{1, \cdots, n\}$, each root processor of T_t^i in M^0 has stored K_i^r, the d'th successor-configuration of K^0 for R_i.

Remark 4. Each pass of the loop of Part 3 needs $O(\log(n))$ steps: $O(\log(n))$ for a) because of the definition of a weak n-simulator, $O(t)$ for b) because of Lemma 2, $O(\log(n))$ for c) because we have assumed that each configuration of a processor is represented by an integer string of length $O(\log(n))$. Thus Part 3 needs $O(r\cdot\log(n))=O(d/\varepsilon)$ steps.

Part 4. For each $q\in\{0,\cdots,r-1\}$, $i\in\{1,\cdots,n\}$, K_i^r is transported to each root processor of T_t^i in M^q.

Remark 5. This can be done in $O(r\cdot\log(n))=O(d')$ steps because of the above bound for the lengths of the representations of configurations.

Now we have achieved all preconditions for starting this algorithm again with $K^0\leftarrow K^r$. Remarks 1, 2, 4 and 5 guarantee that we have only needed $O(d/\varepsilon)$ steps for simulating d' steps of H. Repeating this algorithm we obtain that we need $O(l/\varepsilon)$ steps for simulating l steps of H, if $l=\Omega(d)$. If l is smaller, we still have to execute Parts 1 and 2 once. Thus we need $O(d)$ steps also in this case. Therefore in general we need $O(d+l/\varepsilon)$ steps for simulating l steps of H. □

Now the problem of constructing n-simulators is reduced to constructing weak n-simulators.

This will be done with the help of (a,b)-distributors.

For $j\in\{0,\cdots,t-1\}$ let L_j be the following subset of the set of processors of T_t^*.

L_0 is the set of root processors of $T_t^1,\cdots,T_t^n$.

For $j>0$, L_j is the set of all processors which belong to cycles which are joint to processors of L_{j-1} and which do not belong to L_{j-2} or L_{j-1}.

Informally, L_j consists of those processors which belong to a cycle in depth j of some T_t^i in T_t^*. Let $\# L_j=: m_j$, $j\in\{0,\cdots,t-1\}$.

For $j\in\{0,\cdots,t-1\}$ let D_j be an (n,m_j)-distributor with initialization time d_j.

Then the fair PC M based on $D_0,\cdots,D_{t-1}$ is defined as follows: M consists of T_t^* and $D_1,\cdots,D_{t-1}$ where for $j\in\{0,\cdots,t-1\}$ L_j is the set of input processors of D_j and the jth root processors of $T_t^1,\cdots,T_t^n$ are its output processors.

LEMMA 7. *M is a weak n-simulator with initialization time $O(\log(n)^2+\sum_{j=0}^{t-1}d_j)$.*

Proof. Let $K=(K_1,\cdots,K_n)$ be a configuration of H, and suppose that for each $i\in\{1,\cdots,n\}$, each root processor of T_t^i has stored $\mathrm{Com}(K,R_i,t)$. Let for each processor P of T_t^* $l(P)$ be the number $i\in\{1,\cdots,n\}$ such that R_i is represented by P relative to K. Clearly, for each $i\in\{1,\cdots,n\}$, $l(P)=i$ for each root processor P of T_t^i. The following algorithm initializes M for K.

```
For $j=0,\cdots,t-1$ do (sequentially)
Begin
  a) $D_j$ initializes itself for $(l(P), P\in L_j)$.
  b) $D_j$ distributes $(\mathrm{Com}(K,R_i,t), i\in\{1,\cdots,n\})$ according to $(l(P), P\in L_j)$.
  c) For each $P\in L_j$; If for $q\in\{1,\cdots,t-1\}$, $p\in\{0,\cdots,q-1\}$, $P$ is the $p$th root
     processor of an exemplary of $T_q$ in $T_t^*$, then $P$ sends $z:=\mathrm{Com}(K,R_{l(P)},t)_{p+1}$
     to its neighbour $Q$ in $L_{j+1}$ and $l(Q):=z$.
     Comment: Now for each cycle whose processors belong to $L_j$, one of its
     processors $Q$ knows $l(Q)$.
  d) For each $Q\in L_{j+1}$ which knows $L(Q)$: $Q$ transports $l(Q)$ to each processor $Q'$
     of the cycle it belongs to, and $l(Q'):=l(Q)$.
End
```

Obviously this algorithm attaches the correct address $l(P)$ to each processor P of T_t^*. Because of the initializations of $D_0,\cdots,D_{t-1}$ in step a) of the passes of the loop, finally M is initialized for K.

For $j \in \{0, \cdots, t-1\}$, the jth pass of the loop needs $O(d_j)+O(\log(n))+O(1)+O(t)=O(d_j+\log(n))$ steps. Thus the initialization time of M is $O(\log(n)^2+\sum_{j=0}^{t-1} d_j)$. Now a preparation of T_t^* in M can be executed in $O(\max_j\{\log(d_j)+\log(n)\})=O(\log(n))$ steps. □

We now apply Theorem 1 to Lemma 7 and obtain a weak n-simulator M_1. Because of the choice of t the number of processors of T_t^* is at most $3n^{1+\varepsilon}$. The distributors $D_0, \cdots, D_{t-1}$ all together have $\sum_{j=0}^{t-1} O(m_j \log(m_j)) = O(\log(n) \sum_{j=0}^{t-1} m_j) = O(n^{1+\varepsilon} \log(n))$ processors, as $\sum_{j=1}^{t-1} m_j$ is the number of processors of T_t^*.

Thus M_1 has $O(n^{1+\varepsilon} \log(n))$ processors.

Inserting the bound for the initialization time of $W_{h'}$ from Theorem 1 in Lemma 7, we obtain that the initialization time of M_1 is $O(\log(n)^5)$.

Inserting these results in Theorem 6 we obtain

THEOREM 8. *M_1^* is an n-simulator with $O(n^{1+\varepsilon}\log(n)^5)$ processors. M_1^* can simulate l steps of some arbitrary unfair* PC *with pred.com. and n processors in $O(\log(n)^5+l/\varepsilon)$ steps.*

With the help of the two distributors mentioned at the end of § 2, one can construct weak n-simulators M_2 and M_3.

THEOREM 9. *M_2^* (M_3^*) is an n-simulator with $O(n^{1+\varepsilon}\log(n)^2)$ processors. M_2^* (M_3^*) can simulate l steps of some arbitrary unfair* PC *with pred.com. and n processors in $O(\log(n)^2+l/\varepsilon)$ steps (with overwhelming probability).*

5. A lower bound for the time loss of general n-simulators.

In this section we show that we may not hope for such fast simulations as described in the previous sections, if we want to construct general n-simulators, i.e. if we remove the restriction "predictable communication" from the unfair PC's being simulated.

We now shall define a graph-theoretical model of simulations of unfair PC's by a fair PC. For this purpose we first note that the problem arising during a simulation is that of realizing the communications between processors. Therefore we represent a step of a computation by the directed graph F with n vertices $R_1, \cdots, R_n$, in which (R_i, R_j) is an edge for some $i, j \in \{1, \cdots, n\}$, $i \neq j$, if R_i reads a piece of information from R_j in this step. As each processor can read information from at most one other processor in one step, F has outdegree one. We now call such a directed graph F with n vertices and outdegree one a computation step. A sequence $F_1, \cdots, F_l$ of computation steps is called a computation (of length l).

We now define a graph theoretical model of a simulation of a computation of length l by a fair PC M with degree c and m processors $Q_1, \cdots, Q_m$ for some integer $m \geqq n$. We shall identify M with its communication network.

The following model of simulation is that of a simulation of type 3 as defined in [4]. This model is very general: for example, each simulation developed so far in the articles quoted in this paper or in previous sections is of this type.

Let $F_1, \cdots, F_l$ be a computation. Then a simulation of $F_1, \cdots, F_l$ by M consists of pairwise disjoint, nonempty subsets $A_1^t, \cdots, A_n^t$ of $\{Q_1, \cdots, Q_m\}$ for each $t \in \{0, \cdots, l\}$. For $t \in \{0, \cdots, l\}$, $i \in \{1, \cdots, n\}$, A_i^t contains those processors which simulate R_i when t steps of the computation are simulated. The members of A_i^t are called the representatives of R_i at time t. $A_1^1, \cdots, A_n^1$ have one element, each. In order to simulate the t-th step, $t \in \{1, \cdots, l\}$, each $Q \in A_i^t$, $i \in \{1, \cdots, n\}$, must be joint by a path in M to some $Q' \in A_i^{t-1}$ and some $Q'' \in A_j^{t-1}$, if (i, j) is an edge in F_t. These paths are called the t-transport paths and their maximal length is k_t, the t-time loss of simulation. The time loss of the simulation is

$$k = \frac{1}{l} \sum_{j=1}^{l} k_j.$$

We now describe the difference between a simulation of an unfair PC with pred.com. and an unfair PC whose communication can not become predicted fast. In the first case we may assume that M knows the whole computation $F_1, \cdots, F_l$ already in the beginning of the simulation, because it can precompute it without loss of time. In the second case this is impossible; we only may assume that a general n-simulator simulates "on-line" i.e. it gets to know F_t after having simulated $F_1, \cdots, F_{t-1}$ for each $t \in \{2, \cdots, l\}$.

Now let the time loss of a general n-simulator M be the maximal time loss of the simulation of some computation.

We shall prove:

THEOREM 10. *Each general n-simulator has time loss* $\Omega(\log(n))$.

As pointed out in the introduction, this bound can be achieved with the help of the fair PC's from [1], [6], or [7].

In order to prove this theorem let M be a fair PC with m processors. We shall define a computation of infinite length for which each finite initial sequence is simulated by M with time loss $\Omega(\log(n))$.

If in some step t of a simulation $A_1, \cdots, A_n$ are the sets of representatives at time t, we say M is initialized with $A_1, \cdots, A_n$. The key observation of this proof is as follows:

There is a computation F such that either M needs at least $\gamma \log(n)$ steps to simulate it for some suitable $\gamma > 0$ or the number of representatives at time $t+1$ decreases considerably relative to the number of representatives at time t. As this number may not become smaller than n, we may not too often simulate fast. This will prove the theorem.

LEMMA 11. *For each $\varepsilon \in (0, \frac{1}{2})$ there is $\gamma > 0$ such that for each initialization of M with some $A_1, \cdots, A_n$ there is a computation step F with the property: M needs at least $\gamma \log(n)$ steps to simulate F or the sets of representatives $A'_1, \cdots, A'_n$ after the simulation of this step fulfills*

$$\sum_{i=1}^{n} \# A'_i \leqq \frac{1}{n^{\varepsilon}} \sum_{i=1}^{n} \# A_i.$$

Proof. Let $i \in \{1, \cdots, n\}$ be fixed, and for $j \in \{1, \cdots, n\}$, $j \neq i$, let F^j be the computation step with one edge, namely (i, j). Let M be initialized with $A_1, \cdots, A_n$.

Let $\varepsilon \in (0, \frac{1}{2})$ be fixed.

Claim. There is $\gamma > 0$ such that the following holds: if for each $j \in \{1, \cdots, n\}$, $j \neq i$, M can simulate F^j in less than $\gamma \log(n)$ steps, then there is $j' \in \{1, \cdots, n\}, j' \neq i$, such that M simulates $F^{j'}$ in at least $\gamma \log(n)$ steps, or such that the set A'_i of representatives for R_i after the simulation of F^j fulfills $\# A'_i \leqq \# A_i / n^{\varepsilon}$.

Proof. Let k be the maximal time loss of M when simulating some F^q, $q \in \{1, \cdots, n\}$, $q \neq i$. Let $j \in \{1, \cdots, n\}, j \neq i$. Let $A'_1, \cdots, A'_n$ be the sets of representatives after the simulation of F^j. We say, a processor $P \in A_i$ survives, if there is a transport path from P to some processor Q of A'_i. In this case we say Q is created by P. We denote the set of surviving processors form A_i by B^j. For each $P \in B^j$ we fix some $C(P) \in A'_i$ which is created by P. The element of A_j which is joint to $C(P)$ by a transport path is called the partner of P. (If there are more than one such element in A_j, pick one of them.) We now may conclude the following:

1) The partner of each $P \in B^j$ belongs to $U_{2k}(P)$[1].

2) Each $Q \in A_j$ is the partner of at most $\# U_{2k}(P)$ processors of B^j.

[1] For a processor P of M, $U_r(P)$ denotes the set of all processors of M which can be reached from P by a path of length at most r. For a subset A of the processors of M, $U_r(A) := \bigcup_{P \in A} U_r(P)$.

1) and 2) hold because P and its partner are joined by a path of length at most $2k$ via $C(P)$.

As for each processor P of M, $\# U_{2k}(P) \leqq c^{2k+1}$ we may conclude by 2) that there are at least $(\# B^j)\cdot c^{-2k-1}$ different partners of processors from B^j which all belong to $U_{2k}(B^j)$ because of 1).

As this holds for each $j \in \{1, \cdots, n\}, j \neq i$, and as these sets of partners are pairwise disjoint we obtain:

$$\# U_{2k}(A_i) \geqq \frac{1}{c^{2k+1}} \cdot \sum_{\substack{j=1\\ j\neq i}}^{n} \# B^j.$$

As on the other hand $\# U_{2k}(A_i) \leqq c^{2k+1}\cdot \# A_i$, we obtain that

$$\sum_{\substack{i=1\\ i\neq j}}^{n} \# B^j \leqq \# A_i \cdot c^{4k+2}.$$

Now let $j' \in \{1, \cdots, n\}$, $j \neq i$, be chosen such that $\# B^{j'}$ is minimal.
Then

$$\# B^{j'} \leqq \frac{1}{n-1}\cdot c^{4k+2}\cdot \# A_i \leqq \frac{1}{n^{2\varepsilon}}\cdot \# A_i,$$

if we choose $k \leqq \gamma' \log(n)$ for some suitably chosen $\gamma' > 0$. Note that $\varepsilon < \frac{1}{2}$, thus $1 - 2\varepsilon > 0$ and we may choose

$$\gamma' \approx \tfrac{1}{4}(1-2\varepsilon)\log(c) > 0.$$

Here we assume that each F^j can be simulated with less then $\gamma' \log(n)$ steps.

As we know that only surviving processors from A_i can create members of A_i', we know that only processors of $B^{j'}$ can do so if $F^{j'}$ is simulated. We may choose $\gamma'' > 0$ such that $\# U_{\lfloor \gamma'' \log(n)\rfloor}(P) \leqq n^{\varepsilon}$ for each processor P of M. Let $\gamma := \min\{\gamma', \gamma''\}$. Now assume that $F^{j'}$ is simulated with less than $\gamma \log(n)$ steps. As $\gamma \leqq \gamma'$ we know from above that $\# B^{j'} \leqq \# A_i / n^{2\varepsilon}$. As $\gamma \leqq \gamma''$ we know that each member of $B^{j'}$ can only create n^{ε} elements of A_i'. Thus $\# A_i' \leqq \# B^{j'}\cdot n^{\varepsilon} \leqq \# A_i / n^{\varepsilon}$ which proves the claim. □

We now can define the computation step F demanded in Lemma 11. If there are $i, j \in \{1, \cdots, n\}$, $i \neq j$, such that the simulation of the computation step which only has the edge (i, j) needs at least $\gamma \log(n)$ steps, then this is F. Otherwise, let F contain all those edges (i, j'), $i \in \{1, \cdots, n\}$, where j' is defined for i in the claim. Now if M simulates F faster than $\gamma \log(n)$, we know that for the sets of representatives $A_1', \cdots, A_n'$ after this simulation, $\# A_i' \leqq \# A_i / n^{\varepsilon}$ for each $i \in \{1, \cdots, n\}$. Thus $\sum_{i=1}^{n} \# A_i' \leqq 1/n^{\varepsilon}\cdot \sum_{i=1}^{n} \# A_i$, which proves the lemma. □

Proof of Theorem 10. Consider the computation $F_1, F_2, \cdots$ which is defined step by step by Lemma 11. Let l be an integer and for $t \in \{0, \cdots, l\}$ let h_t denote the number of representatives at time t. Let $k_1, \cdots, k_l$ be the t-time losses of the simulation of $F_1, \cdots, F_l$ and let $S \subset \{1, \cdots, l\}$ be the set of those indices t for which k_t is smaller than $\gamma \log(n)$, $s := \# S$. Then

$$(*) \qquad \sum_{t=1}^{l} k_t \geqq \sum_{t \in S}^{l} k_t \geqq (l - s)\gamma \log(n).$$

It remains now to bound s. We know that $h_0 = n$, $h_t \geqq n$ for each $t \in \{1, \cdots, l\}$. Furthermore we know by Lemma 11 that during the simulation the number of representatives is decreased s times by a factor of at least $1/n^{\varepsilon}$, namely during the simulation of each

tth step with $t \in S$. On the other hand it is at most $(l-s)$ times increased by a factor of at most c^{k_t+1} for each $t \in \{1, \cdots, l\} \backslash S$. Thus we may conclude

$$n \leqq h_l \leqq h_0 \cdot \left(\frac{1}{n^{\varepsilon}}\right)^s \cdot \prod_{\substack{t=1 \\ t \notin S}}^{l} c^{k_t+1} \leqq \frac{1}{n^{\varepsilon \cdot s-1}} \cdot c^{l+\sum_{t=1}^{l} k_t}.$$

If now $\sum_{t=1}^{l} k_t > l \cdot (\varepsilon \cdot \log_c (n)/2 - 1)$ the theorem is proved.

Otherwise $n \leqq n^{(\varepsilon \cdot l)/2} / n^{\varepsilon \cdot s-1}$ which implies $n^{\varepsilon \cdot s} \leqq n^{(\varepsilon \cdot l)/2}$. Thus $s \leqq l/2$.

Inserting this bound in the inequality (*) from above, we obtain $\sum_{t=1}^{l} k_t \geqq l \cdot (\gamma/2) \log (n)$, which proves the theorem. □

REFERENCES

[1] M. Ajtai, J. Komlos and E. Szemeredi, *An $O(n \log (n))$ sorting network*, Proc. 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 1-9.

[2] K. Batcher, *Sorting networks and their applications*, AFIPS Spring Joint Computing Conference, 32 1968, pp. 307-314.

[3] Z. Galil and W. J. Paul, *A general purpose parallel computer*, J. Assoc. Comput. Mach., 30 (1983), pp. 360-387.

[4] F. Meyer auf der Heide, *Efficiency of universal parallel computers*, Acta Informatica, 19 (1983), pp. 269-296.

[5] F. P. Preparata and J. Vuillemin, *The cube-connected cycles: a versatile network for parallel computation*, Comm. ACM, 24 (1981), pp. 300-310.

[6] J. H. Reif and L. G. Valiant, *A logarithmic time sort for linear size networks*, 15th Annual ACM Symposium on Theory of Computing, Boston, 1983, pp. 10-16.

[7] E. Upfal, *Efficient schemes for parallel communication*, Proc. ACM Symposium on Principles of Distributed Computing, Ottawa, 1982.

[8] L. G. Valiant and G. J. Brebner, *Universal schemes for parallel communication*, Proc. 13th Annual ACM Symposium on Theory of Computing, Milwaukee, WI, 1981, pp. 263-267.

[9] A. Waksman, *A permutation network*, J. Assoc Comput. Mach., 15 (1968), pp. 159-163.

[10] U. Vishkin, *A parallel-design-distributed-implementation* (PDDI) *general purpose computer*, Technical Report No. 96, Dept. Computer Science, New York Univ., New York, 1983.