Efficient Simulations among Several Models of Parallel Computers (Extended Abstract)

Friedhelm Meyer auf der Heide
Fachbereich Informatik
Johann Wolfgang Goethe-Universität
6000 Frankfurt am Main
Fed. Rep. of Germany

Abstract: We consider parallel computers (PC's) with fixed communication network and deal with the question, how fast can we simulate PC's with n processors whose communication network has unbounded degree (unfair PC's) by PC's where this degree is bounded (fair PC's). An important class of unfair PC's is thatone of PC's with predictable communication (pred.com.). In such a PC each processor can compute in $0(t)$ steps the sequence of processors it wants to communicate with during the next t steps. A famous example of such PC's are cubes initialized with ascend-and descend programs as introduced by Preparata and Vuillemin in [5]. They could simulate such unfair PC's with pred.com. with constant time loss using only as many processors as the cube has. We generalize this result by presenting a fair PC which can simulate each unfair PC with pred.com., n processors, and $0(\log(n))$ storage locations per processor with constant time loss using $0(n^{1+\varepsilon})$ processors for an arbitrary $\varepsilon>0$.

I. 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 usual sequential capabilities. They are partially joint by wires. The graph defined by the processors and the wires is its communication network. In one step each processor is allowed to read an information from a (relative to the communication network) neighboring processor. We allow that several processors read from the same processors at the same time. We assume the PC is syncronized.

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 $O(\log(n))$ storagelocations, each for 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].

This unfair PC has $N=2^k$ 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 which N processors- the Cube-Connected Cycles- and constant time loss.

In this paper we shall present a fair PC which can simulate each unfair PC with pred.com. with n processors, a so-called n-simulator. This n-simulator can for an arbitrary $\varepsilon>0$ be constructed such that it has $O(n^{1+\varepsilon})$ processors and only a constant time loss. (For a more detailed construction see [10].)

In [10] the above method of simulating and ideas from [4] are combined for constructing a so-called n-universal PC. Such a fair PC can simulate each fair PC with n processors and fixed degree. 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 a time loss $O(\log\log(n))$.

The n-universal PC constructed in [10] has the same number of processors as above but a constant time loss only. This result even holds if we remove the restriction of the storage capacity of the processors.

In [10], it is also shown that we cannot hope for fast simulations if we want to build a general n-simulator. Such a fair PC is able to simulate each unfair PC, also those whose communication is not predictable. A lower bound of $\Omega(\log(n))$ is proved 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 results from [1], [6], [7] or [8] one can prove that this lower bound is tight.

II. The Construction of an n-Simulator.

In this chapter we shall construct an n-simulator. The construction of the n-simulator proceeds in three steps. First we describe for each integer t a fair PC T_t^* which can simulate t steps of each unfair PC with pred.com. in $O(t)$ steps. Afterwards we define what we mean by weak n-simulators. Based on such fair PC's we construct our n-simulators. Finally we quote from [10] some ways of constructing weak n-simulators which completes our construction.

First we describe a 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 initialized in an appropriate way. T_t^* consists of n exemplaries $T_t^1,\dots,T_t^n$ 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 $T_0,\dots,T_{t-1}$ and t new processors $P_0,\dots,P_{t-1}$. These processors form the root cycle of T_t by wires between P_p and $P_{(p+1)\mathrm{mod}(t)}$ for $p \in \{0,\dots,t-1\}$.

P_0 is the main root. Furthermore, for each $p \in \{0,\dots,t-1\}$, P_p is joint to the main root of T_p.

An example of this fair PC is shown in figure 1.

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.

<u>Lemma 1:</u> For $t\geq 1$, 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 $R_1,\dots,R_n$. A configuration $K=(K_1,\dots,K_n)$ of H consists of configurations K_i for each processor R_i of H, $i \in \{1,\dots,n\}$. Recall that each processor only has $O(\log(n))$ storage areas. 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 a string of integers of length

$O(\log(n))$. In the sequel we shall identify this string with the configuration.

Now suppose that H started with K executes p steps for some integer p. The resulting configuration $\bar{K}=(\bar{K}_1,\dots,\bar{K}_n)$ then is called the p'th successor configuration of K and for $i \in \{1,\dots,n\}$, $\bar{K}_i$ is the p'th successor configuration of K for R_i.

For an integer p and $i \in \{1,\dots,n\}$, $Com(K,R_i,p)$ denotes the string of addresses of processors R_i reads from during p steps of H started with K. For $q \in \{1,\dots,p\}$, $Com(K,R_i,p)_q$ denotes the q'th element of $Com(K,R_i,p)$. If for some such q, H doesn't read from another processor in the q'th step started with K, we assume that $Com(K,R_i,p)=i$.

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

If $t=0$ the T_t contains K_i.

Let $t>0$. Then each root processor contains K_i, and for each $p \in \{0,\dots,t-1\}$ the exemplary T_p joint to the p'th root processor is prepared for K and R_j for p steps, if $j=Com(K,R_i,t)_{p+1}$ and $j \neq i$. If $j=i$, T_p may be arbitrary.

T_t^* is prepared for K if for each $i \in \{1,\dots,n\}$ 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 t'th successor configuration of K for R_i in each root processor of T_t^*, $i \in \{1,\dots,n\}$.

Lemma 2: If T_t^* is prepared for K, it can simulate t steps of H started with K in O(t) steps.

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

For $p \in \{1,\dots,t\}$ we say that the root cycle of T_t^i is p-prepared if P_{p-1} and $P_{p \bmod (t)}$ contain the p'th successor configuration of K for R_i and for each $q \in \{1,\dots,p-1\}$, $P_{(p+q) \bmod (t)}$ contains the (p-q)'th successor configuration of K for R_i.

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 $q \in \{0,\dots,t-1\}$ the main root Q of the exemplary of T_q joint to P_q contains the q'th 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 $p \in \{0,\dots,t-2\}$, it becomes $(p+1)$-prepared by the following algorithm.

Part 1: For each $q \in \{1,\dots,p\}$, $P_{(p+q)\mathrm{mod}(t)}$ simulates the $(p-q+1)$'th step of R_i with the help of $P_{(p+q-1)\mathrm{mod}(t)}$.

Remark 1: As $P_{(p+q-1)\mathrm{mod}(t)}$ has already executed this step by 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 $q \in \{1,\dots,p+1\}$, $P_{(p+q)\mathrm{mod}(t)}$ simulates the $(p-q+2)$'th step of R_i with the help of $P_{(p+q-1)\mathrm{mod}(t)}$.

Remark 3: This works in constant time, because in step 1 resp. step 2, $P_{(p+q-1)\mathrm{mod}(t)}$ 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 $s' \cdot p +$(time for step 1) many steps that means it is available when it is required by P_p. Thus P_0 contains the t'th 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 $i \in \{1,\dots,n\}$ we have simulated t steps of M started with K in $(s'+s'')\cdot t$ steps.

Figure 2 shows the states of the p-prepared root cycle of T_8^i for some $i \in \{1,\dots,n\}$ and each $p \in \{1,\dots,8\}$. A number ℓ in the q'th column and p'th row, $q \in \{0,\dots,7\}$, $p \in \{1,\dots,8\}$ means:
If T_8 is p-prepared, P_q contains the ℓ'th successor configuration of K for R_i.

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 1. 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. Unfortunately such algorithms would need at least $\Omega(\log(n)^2)$ steps. 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 doesn't neede much more time than one preparation.

This initialization effects that afterwards d preparations can become executed, each in $O(\log(n))$ steps. This trick will guarantee the constant time loss. Let in the sequel $\varepsilon>0$ be fixed and $t:=\lfloor \varepsilon \log(n) \rfloor$. Then by lemma 1, 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. Explicit constructions of weak n-simulators can be found in [10].

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

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

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 $M^0,\dots,M^{r-1}$. For each $\ell \in \{0,\dots,r-1\}$, $i \in \{1,\dots,n\}$, each root processor of T_t^i in M^ℓ is joint to the corresponding processor in $M^{(\ell+1)\mathrm{mod}(r)}$.

<u>Theorem 1:</u> Let M be a weak n-simulator with initialization time d. Then M* is an n-simulator which can simulate ℓ steps pf some arbitrary unfair PC with pred.com. and n processors in $O(d+\ell)$ steps. If M has m processors, M* has $\lceil d/\ t \rceil \cdot$ m processors.

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

Assume that for each $i \in \{1,\dots,n\}$, $q \in \{0,\dots,r-1\}$, K_i^0 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 $q \in \{0,\dots,r-1\}$, $i \in \{1,\dots,n\}$, each root processor of T_t^i in M^q computes $Com(K^q,R_i,t)$.

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 $q \in \{0,\dots,r-1\}$, 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 $q=0,\dots,r-1$ 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 $i \in \{1,\dots,n\}$ each root processor of $T_t^i$
   in T' contains $K_i^{(q+1)}$.
c) For each $i \in \{1,\dots,n\}$, 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 $i \in \{1,\dots,n\}$, 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 $O(t)=O(\log(n))$ 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 $O(r \cdot t) = O(d')$ steps.

Part 4: For each $q \in \{0,\dots,r-1\}$, $i \in \{1,\dots,n\}$, 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 lengthes of the representations of configurations.

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

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

For this construction we define so-called (a,b)-distributors $D_{a,b}$ for integers a,b, $a \leq b$.
Such a fair PC has a+b distinguished processors, a input processors $A_1,\dots,A_a$ and b output processors $B_1,\dots,B_b$, and has the following property:

If each B_i, $i \in \{1,\dots,b\}$, has stored an integer $c_i \in \{1,\dots,a+1\}$, then $D_{a,b}$ can initialize itself such that afterwards the following holds:
If each A_j, $j \in \{1,\dots,a\}$, contains an integer string x_j of length $O(\log(n))$, then $D_{a,b}$ can distribute $(x_1,\dots,x_a)$ according to $(c_1,\dots,c_b)$, i.e. can transport each x_j, $j \in \{1,\dots,a\}$, to each B_i with $c_i=j$, $i \in \{1,\dots,a\}$, in $O(\log(b) + \log(n))$ steps.
The above initialization is called the initialization of $D_{a,b}$ for $(c_1,\dots,c_b)$, and the time it needs is the initialization time of $D_{a,b}$.
Now we shall construct weak n-simulators with the help of (a,b)-distributors.
Let for $j \in \{0,\dots,t-1\}$ L_j be the following subset of the set of processors of T_t^*.
L_0 is the set of root processors of $T_t^1,\dots,T_t^n$.
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 $\#L_j=:m_j$, $j \in \{0,\dots,t-1\}$.
Let for $j \in \{0,\dots,t-1\}$ D_j be a (n,m_j) distributor with initialization time d_j.

Then the fair PC M based on $D_0,\dots,D_{t-1}$ is defined as follows:

M consists of T_t^* and $D_1,\dots,D_{t-1}$ where for $j \in \{0,\dots,t-1\}$ L_j is the set of output processors of D_j and the j'th root processors of $T_t^1,\dots,T_t^n$ are its input processors.

<u>Lemma 3:</u> M is a weak n-simulator with initialization time $0(\log(n)^2 + \sum_{j=0}^{t-1} d_j)$.

<u>Proof:</u> Let $K=(K_1,\dots,K_n)$ be a configuration of H, and suppose that for each $i \in \{1,\dots,n\}$, each root processor of T_t^i has stored Com (K,R_i,t).

Let for each processor P of T_t^* $\ell(P)$ be the address of the processor of H being represented by P relative to K. Clearly, for each $i \in \{1,\dots,n\}$, $\ell(P)=i$ for each root processor P of T_t^i.

The following algorithm initializes M for K.

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

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

For $j \in \{0,\dots,t-1\}$, the j'th pass of the loop needs $d_j+0(\log(n))+0(t)=0(d_j+\log(n))$ steps. Thus the initialization time of M is $0(\log(n)^2 + \sum_{j=0}^{t-1} d_j)$. Now a preparation of T_t^* in M can be

executed in $O(\max_j \{\log(d_j) + \log(n)\}) = O(\log(n))$ steps
Using ideas from [4] one can construct an (a,b)-distributor with $O(b \log(b))$ processors ans initialization time $O(\log(b)^4)$ with the help of Waksman permutation networks [9]. (See [10]).

Applying lemma 3 and theorem 1 we obtain:

<u>Theorem 2:</u> 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 $O(\log(n)^5 + \ell)$ steps.

Finally we note 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 $O(\log(b))$. Call the associated weak n-simulator M_2.

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 initiatisation time $O(\log(b)^2)$ which allows distributions using $O(\log(b)+\log(n))$ steps with overwhelming probability.

The associated weak n-simulator let be called M_3. M_2 and M_3 both have $O(n^{1+\varepsilon}\log(n))$ processors.

Applying M_2 and M_3 to theorem 1 we obtain:

<u>Theorem 3:</u> M_2^* (M_3^*) is an n-simulator with $O(n^{1+\varepsilon}\cdot\log(n)^2)$ processors. M_2^* (M_3^*) can simulate ℓ steps of some arbitrary unfair PC with pred.com. and n processors in $O(\log(n)^2+\ell)$ steps (with overwhelming probability).

References:

[1] M.Ajtai, J.Komlos, E.Szemeredi:
An O(nlog(n)) Sorting Network, Proc. of the 15'th Annual ACM Symposium on Theory of Computing (1983), Boston, USA, 1-9.

[2] K.Batcher:
Sorting Networks and their Applications, AFIPS Spring Joint Comp. Conf. 32 (1968), 307-314.

[3] Z.Galil, W.J.Paul:
A General Purpose Parallel Computer, Journal of the ACM 30(2) (1983), 360-387.

[4] F.Meyer auf der Heide:
Efficiency of Universal Parallel Computers, Acta Informatica 19 (1983) 269-296.

[5] F.P.Preparata, J.Vuillemin:
The Cube-Connected Cycles: A Versatile Network for Parallel Computation, Communications of the ACM 24 (1981), 300-310.

[6] J.H.Reif, L.G.Valiant:
A Logarithmic Time Sort for Linear Size Networks, 15'th Annual ACM Symposium on Theory of Computing (1983), Boston, USA,10-16.

[7] E.Upfal:
Efficient Schemes for Parallel Communication, Proc. of the ACM Symposium on Principles of Distributed Computing (1982), Ottawa, Canada.

[8] L.G.Valiant, G.J.Brebner:
Universal Schemes for Parallel Communication, Proc. of the 13'th Annual ACM Symposium on Theory of Computing (1981), Milwaukee, USA, 263-267.

[9] A.Waksman:
A Permutation Network, Journal of the ACM, 15(1) (1968), 159-163.

[10]F.Meyer auf der Heide:
Efficient Simulations among Several Models of Parallel Computers, Interner Bericht des Fachbereichs Informatik der J.W.Goethe-Universität, Frankfurt, 2/83, to appear in SIAM J.on Comp.

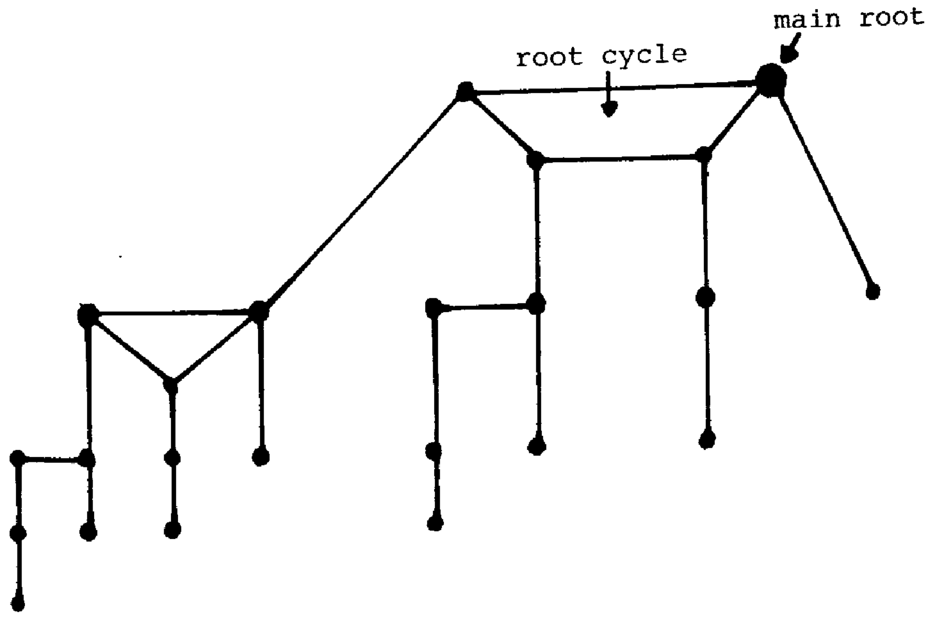

Figure 1: The fair PC T_4.

	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
1	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	$\underline{8}$

Figure 2: The design of a p-prepared root cycle.