INFORMATION AND CONTROL 67, 195–211 (1985)

Lower Time Bounds for Solving Linear Diophantine Equations on Several Parallel Computational Models

FRIEDHELM MEYER AUF DER HEIDE*

Fachbereich Informatik,
Johann Wolfgang Goethe-Universität,
6000 Frankfurt a.M., West Germany

We consider parallel random access machines (PRAM's) with p processors and distributed systems of random access machines (DRAM's) with p processors being partially joint by wires according to a communication graph. For these computational models we prove lower bounds for testing the solvability of linear Diophantine equations and related problems including the knapsack problem. These bounds are achieved by generalizing and simplifying a lower bound for parallel computation trees due to Yao, introducing a new type of computation trees which models computations of DRAM's, and by generalizing a technique used by Paul and Simon and Klein and Meyer auf der Heide to carry over lower bounds from computation trees to RAM's. Thereby we prove that for many problems, p processors cannot speed up a computation by a factor $O(p)$ but only by a factor $O(\log(p+1))$ and in the case of DRAM's whose communication network has degree c by a factor $O(\log(c+1))$ only.

I. INTRODUCTION

In this paper we prove lower bounds for parallel random access machines (PRAM's) and distributed systems of random access machines (DRAM's). PRAM's are often dealt with in literature (see, e.g., Shiloach and Vishkin (1981), Fortune and Willie (1978), and Reischuk (1984)) as a very powerful model of parallel computation in order to design fast parallel algorithms or to prove lower bounds. DRAM's consist of a finite set of RAM's ($\hat{=}$ PRAM's with one processor) which are partially joined by wires according to their communication network. This computational model is, for example, considered in Galil and Paul (1983), Meyer auf der Heide (1983) or Meyer auf der Heide (1985) in order to obtain simulation results.

* Part of this work was done while the author was visiting SFB 124 of the University of the Saarland.

0019-9958/85 $3.00

The lower bounds we want to prove in this paper are applied to recognizing the languages $L_n = \{(\bar{a}, b),\ \bar{a} \in R^n,\ b \in R,\ \exists \bar{x} \in N^n,\ \bar{x} \cdot \bar{a} = b\}$ and $L_{n,k} = \{(\bar{a}, b),\ \bar{a} \in R^n,\ b \in R,\ \exists \bar{x} \in \{0,\ldots, k\}^n,\ \bar{x} \cdot \bar{a} = b\}$. $\bigcup_{n \in N} L_n$ is the language associated to the problem of deciding the solvability of Diophantine equations, $\bigcup_{n \in N} L_{n,1}$ is that language associated with the knapsack problem. Both languages are well known to be NP-complete (see Garey and Johnson (1979).

In this paper we apply a version of the "component counting argument" in order to prove lower bounds. In literature this argument is applied to prove lower bounds on (sequential) computation trees (CT's). These are computations which work up n real inputs, may execute operations from some subset $S \subset \{+, -, *, /, \leqslant\}$, and which only may execute direct, but no indirect storage access. In Dobkin and Lipton (1975), the component counting argument is applied to CT's with operation set $\{+, -, \leqslant\}$ (they are often called linear search programs or linear decision trees in literature). They prove an $\Omega(\log(q))$ lower bound for recognizing a language $L \subset R^n$ consisting of q connected components.

In Ben Or (1983), this result is generalized to CT's with operation set $\{+, -, *, /, \leqslant\}$. A further application of this argument is shown by Yao (1981). He proves an $\Omega(\sqrt{\log(q)/n - \log(p+1)})$ lower bound for a parallel version of CT's (PCT's) with operation set $\{+, -, *, /, \leqslant\}$ and p processors.

In Dobkin and Lipton (1975), the number of connected components of $R^{n+1} \setminus L_{n,1}$ is proved to be $2^{\Omega(n^2)}$. In Meyer auf der Heide (1985), he generalizes this result to $2^{\Omega(n^2 \log(k+1))}$ for $L_{n,k}$. This implies lower bounds $\Omega(n^2 \log(k+1))$ resp. $\Omega(\sqrt{n \log(k+1) - \log(p+1)})$ for $L_{n,k}$ on the computational models above.

In this paper we improve the bound from Yao (1981) to

$$\Omega((\log(q)/n \log(p+1))),$$

if only operations from $\{+, -, \leqslant\}$ are allowed. Furthermore we improve and simplify this bound and the one from Yao (1981) for the case that L is the complement of a union of m hyperplanes. For example, $L_{n,k}$ is such a language. In this case we obtain $\Omega(\sqrt{\log(m) - \log(p)})$ resp. $\Omega((\log(m)/\log(p+1)))$ lower bounds, if operations from $\{+, -, *, /, \leqslant\}$ resp. $\{+, -, \leqslant\}$ are allowed. These bounds are never weaker than those mentioned above, because a language $L = R^n \setminus \{m \text{ hyperplanes}\}$ has at most $(m+1)^n$ connected components as shown in (Klein and Meyer auf der Heide (1983)). These bounds have several advantages: We need not compute the number of connected components of the language, which often can be very difficult, and we get better bounds for the case that L has only $(m+1)^{o(n)}$ connected components.

We then introduce a new model of computation trees, distributed com-

putation trees (DCT's), with communication graph G and p processors which correspond to DRAM's. For a language L as above we prove an $\Omega((\log(m)/\log(c+1))$ lower bound, if c is the degree of G. This bound holds independently of the number of processors. All the above bounds are contained in the third section.

In the second section we introduce the computational models and prove that the above lower bounds for PCT's and DCT's with operation set $\{+, -, \leqslant\}$ are tight for a large class of problems, especially if $L \subset R$.

In the fourth section we generalize a method used already in (Meyer auf der Heide, 1985; Klein and Meyer auf der Heide, 1983; or Paul and Simon, 1980) for carrying over lower bounds from CT's to RAM's to our parallel computational models.

In the fifth section we finally apply this result and the lower bounds from the third section in order to obtain lower bounds for PRAM's and DRAM's. We furthermore show that the bounds already hold for very small input sets. This enables us to prove lower bounds for L_n, too, although L_n is defined by an infinite set of hyperplanes. Therefore L_n cannot be handled by PCT's or DCT's. The bounds we get for L_n depend on the input size. An appropriate lower bound for (sequential) RAM's is shown in Meyer auf der Heide, 1985. Altogether we achieve the following lower bounds for inputs from $\{0,\ldots,(k+1)^{2^n}\}^n$, if p processors are available in order to recognize $L_{n,k}$ or L_n:

- $\Omega(n \log(k+1)/\log(p+1))$ for PRAM's with operations $\{+, -, \leqslant\}$.
- $\Omega(\sqrt{n \log(k+1)} - \log(p))$ for PRAM's with operations $\{+, -, *, \leqslant\}$.
- $\Omega(n \log(k+1)/\log(c+1))$ for DRAM's with operations $\{+, -, \leqslant\}$ and communication graph with degree c.

As we show in the second section, the first and the last bound is often tight. Thus these results show that in many cases p processors cannot speed up a computation by a factor $O(p)$ but only by a factor $O(\log(p+1))$ resp. $O(\log(c+1))$. Especially, if c is a constant, DRAM's are not faster than sequential RAM's for many problems.

II. Computational Models and Upper Bounds

In this chapter we define the models of computation we want to deal with in this paper, namely parallel random access machines (PRAM's), distributed systems of random access machines (DRAM's), parallel computation trees (PCT's), and distributed systems of computation trees

(DCT's). At the end of this chapter we describe some upper bounds for the computational models. As we later want to prove lower bounds, we need detailed definitions of these models.

PRAM's are considered in many papers (Shiloach and Vishkin, 1981; Fortune and Willie, 1978; Reischuk, 1984). Our definition generalizes the usual one for (sequential) RAM's which can be found in Aho, Hopcroft, and Ullman (1974) or, tailored to our application, in Klein and Meyer auf der Heide (1983).

A PRAM with p processors $P_1,..., P_p$ and operation set $S \subset \{+, -, *\}$ consists of an infinite set of registers addressed with positive integers and p special registers, the accumulators of $P_1,..., P_p$. $\langle 0 \rangle_1,..., \langle 0 \rangle_p$ denote their current contents. Each register can store one nonnegative integer. The content of register k is denoted by $\langle k \rangle$. Furthermore to each processor P_i a program is attached. It consists of a finite sequence $I_1,..., I_q$ of instructions from the set below:

(1)	c-load (k)	$\langle 0 \rangle_i \leftarrow k$
(2)	load (k)	$\langle 0 \rangle_i \leftarrow \langle k \rangle$
(3)	i-load (k)	$\langle 0 \rangle_i \leftarrow \langle\langle k \rangle\rangle$
(4)	store (k)	$\langle k \rangle \leftarrow \langle 0 \rangle_i$
(5)	i-store (k)	$\langle\langle k \rangle\rangle \leftarrow \langle 0 \rangle_i$
(6)	"o" (k)	$\langle 0 \rangle_i \leftarrow \langle 0 \rangle_i \, o \langle k \rangle, o \in S$
(7)	if $\langle 0 \rangle_i > 0$	then goto $I_j, j \in \{1,..., q\}$
(8)	stop	M stops.

The first instruction executed is always I_1. It is always assumed that only nonnegative integers are computed. If a PRAM M is started with $\bar{x} = (x_1,..., x_n) \in N^n$, then we assume that initially $\langle i \rangle = x_i$, $i = 1,..., n$, and all other registers have stored 0. M is synchronized, i.e., one step consists of the execution of one instruction by each processor. M recognizes $L \subset N^n$, if M started with $\bar{x} \in N^n$ stops with $\langle 0 \rangle_1 = 1$, if $\bar{x} \in L$, and with $\langle 0 \rangle_1 = 0$ else. There is a problem arising out of the fact that several processors may manipulate the same storage location in one step by a store instruction. In literature, there are many solutions described for this conflict (see Shiloach and Vishkin (1981), Fortune and Willie (1978), and Reischuk (1984)). In this paper we shall assume the most powerful of them, which says that if several processors want to store something in the same register, the one with the largest address succeeds. The usual (sequential) RAM can be considered as a PRAM with one processor.

DRAM's are considered, for example, in Galil and Paul (1983), Meyer auf der Heide (1983), and Meyer auf der Heide (1986). Additionally, for example, sorting networks (Batcher (1968)), permutation networks

(Waksman (1968)), or the cube-connected cycles network (Preparata and Vuillemin (1981)) can be looked upon as DRAM's.

A DRAM M with p processors $P_1,..., P_p$, operation set $S \subset \{+, -, *\}$, and communication graph G consists of p RAM's $P_1,..., P_p$ with operation set S, which are partially joint by wires. The processors and the wires form the communication graph G.

Each processor P_i has the additional capability of executing an instruction from

- read (j) $\quad \langle o \rangle_i \leftarrow \langle o \rangle_j$
- i-read (k) $\quad \langle o \rangle_i \leftarrow \langle o \rangle_{\langle k \rangle_i}$
- write (j) $\quad \langle o \rangle_j \leftarrow \langle o \rangle_i$
- i-write (k) $\quad \langle o \rangle_{\langle k \rangle_i} \leftarrow \langle o \rangle_i$,

where $\langle l \rangle_j$ denotes the content of register l of P_j. If P_j or $P_{\langle k \rangle_i}$ are not neighbors of P_i in G, then M stops with an error message. M is synchronized.

An input $\bar{x} \in N^n$ for M is stored in the first n processors of each P_i, the output (0 or 1) appears in the accumulator of P_1. Thereby the recognized language is defined as for RAM's. Write conflicts are solved as for PRAM's.

The computational models we define now, PCT's, and DCT's, are—as we see later—abstractions of PRAM's and DRAM's in which no indirect storage access (i-store (k), i-load (k), i-read (k), i-write (k)) is allowed and which can work up inputs from R^n.

PCT's are introduced in Yao (1981) and a sequential version in Ben Or (1983), and for operation set $\{+, -\}$ in Dobkin and Lipton (1975).

A PCT T for R^n with p processors and operation set $S \subset \{+, -, *, /, :\}$ is a finite tree in which each inner node v is labelled with a tupel $(J_1,..., J_p)$. J_i may be a function $L: R^n \to R$ which is of the form $L = L_1 o L_2$ with $o \in S$ and L_1 (resp. L_2) may be previously computed on the path to v or a constant or one of the input variables $x_1,..., x_n$. J_i may also be a question "$L(x_1,..., x_n) > 0$" for some previously computed function L. If $(J_1,..., J_p)$ contains $p' \leqslant p$ questions, then v has $2^{p'}$ sons, one for each of the $2^{p'}$ possible outcomes of the questions. The leaves are labelled with accept or reject. Thereby to each input $\bar{x} \in R^n$ a unique path to a leaf is associated in the obvious way. The set of inputs passing through a node v is called $c(v)$.

The union over all sets $c(v)$ for the accepting leaves v of T is the language recognized by T, the depth of a deepest node v with $c(v) \neq \emptyset$ is its complexity. A PCT with one processor is a (sequential) computation tree (CT) as introduced in Ben Or (1983).

The following definition of a DCT is new. A DCT T for R^n with p processors $P_1,..., P_p$, operation set $S \subset \{+, -, *, /, :\}$ and communication graph G with vertices $P_1,..., P_p$ consists of p trees $T^1,..., T^p$, each similar to a

CT. For $i \in \{1,..., p\}$, T^i describes the computation of P_i. The computation consists of internal computation steps, executed by nodes in even depth, and communication steps, executed by nodes in odd depth.

We shall attach a function $L_v: R^n \to R$ to each node v of T. Nodes in even depth get sons and these sons get functions as described for CT's. Now let $i \in \{1,..., p\}$ and v' be a node in odd depth l in T^i. Let $P_{i_1},..., P_{i_c}$ be the neighbors of P_i in G. Then for each combination $v_1,..., v_c$ of nodes in depth l of $T^{i_1},..., T^{i_c}$, v' gets a son v. The edge (v', v) is chosen by inputs from $\bigcap_{j=1}^{c} c(v_j)$ and $L_v \in \{L_{v'}, L_{v_1},..., L_{v_c}\}$. The leaves of T^1 are labelled with accept or reject. The set of inputs arriving at an accepting leaf of T^1 form the recognized language of T. The depth of a deepest node v in T with $c(v) \neq \varnothing$ is the complexity of T.

We now shall describe a PRAM (and a DRAM) with p processors (whose communication graph is a balanced, c-ary tree) which recognize a language $L = \{x_1,..., x_q\} \subset N$. We assume that $x_1 < \cdots < x_q$. For simplicity of description we assume that $q = p^r$ for some $r \in N$. Then the following, recursively defined PRAM M with p processors $p_1,..., p_p$ recognizes L. Let $x \in N$ be an input for M.

If $r = 0$, (i.e., $q = 1$) then P_1 tests whether $x = x_1$. If $r > 0$ then partition L in p sets $L_1,..., L_p$, $L_i = \{x_{(i-1)p^{r-1}+1},..., x_{ip^{r-1}}\}$:

(a) The P_i's test in parallel, whether $x_{(i-1)p^{r-1}} < x \leqslant x_{ip^{r-1}}$ holds.

(b) The (unique) P_i for which the above holds stores its address i into register 1.

(c) Each processor reads $\langle 1 \rangle$.

(d) M recognizes recursively $L_{\langle 1 \rangle}$.

Obviously, the above algorithm is correct. Let $T(q)$ be its run time for a language with q elements. Then $T(1) \leqslant d$ and for $r > 0$ $T(p^r) \leqslant T(p^{r-1}) + d$ for some constant $d > 0$. Thus $T(q) = O(\log(q)/\log(p)))$.

A similar algorithm can be executed on a DRAM with p processors whose communication graph is a c-ary, balanced tree. Here we have to modify step (b) of the above algorithm. Now the processor P_i as in (b) communicates its address i to all other processors by sending it along the wires in the communication graph G. This can obviously be done by read and write instructions in time $O(\text{longest path in } G) = O(\log(p)/\log(c))$. Then we can go on recursively as described above in (c) and obtain an analogous recursion for the run time $T'(q)$ for recognizing a language with q elements, $q = p^r$: $T'(1) \leqslant d'$ and for $r > 0$, $T'(p^r) \leqslant T'(p^{r-1}) + d' \cdot (\log(p)/\log(c))$ for some constant $d' > 0$. Thus $T'(q) = O(\log(q)/\log(c)))$.

These algorithms can easily be generalized to arbitrary values of q. We only need the operation "$-$" because, in our models, a comparison is executed as "$x_i - x_j > 0$." Thus we have shown

Remark 1. A language $L=\{x_1,\ldots,x_q\}\subset N$ can be recognized by a PRAM (DRAM) with p processors (whose communication graph is a balanced c-ary tree) and operation set $\{-\}$ in

$$O(\log(q)/\log(p))) \ (O(\log(q)/\log(c))) \text{ steps.}$$

We shall see later that these bounds are tight within a constant factor, also if we allow the operation set $\{+,-\}$ and arbitrary communication graphs with degree $(c+1)$. Especially this shows that by a DRAM with bounded degree (independent of p and q), a language L as above cannot be recognized faster than by a RAM.

III. Lower Bounds for Parallel and Distributed Computation Trees

In this chapter we shall prove lower bounds for PCT's and DCT's by applying the "component counting argument" as it is, for example, used in Dobkin and Lipton (1975), Ben Or (1983), and Yao (1981). We first state a lemma from Meyer auf der Heide (1985) which we will often use in the sequel.

Lemma 1 (Klein, Meyer auf der Heide). *Let $H_1,\ldots,H_m$ be hyperplanes in R^n, then $R^n\setminus\bigcup_{i=1}^m H_i$ consists of at most $(m+1)^n$ connected components.*

The following theorem is due to Yao (1981) and makes use of a fairly deep result from algebraic topology.

Theorem 1 (Yao). *Let $L\subset R^n$ consist of q connected components. Then each* PCT *for R^n with p processors and operation set $\{+,-,*,/\}$ which recognizes L has complexity at least $\sqrt{\log(q)}/\sqrt{n}-\frac{1}{2}\log(p)$.*

In Corollary 1 we present an elementary proof of this theorem for the case that L is defined by hyperplanes. This result also applies to the knapsack problem considered by Yao (1981).

We now improve the above result for the case that only the operation set $\{+,-\}$ is allowed.

Theorem 2. *Let $L\subset R^n$ consist of q connected components. Then each PCT T for R^n with p processors and operation set $\{+,-\}$ which recognizes L has complexity at least $\log(q)/(n\log(p+1))$.*

Proof. Here we first note that the functions attached to a PCT T as above are affine, i.e., are of the form $L(x_1,\ldots,x_n)=\sum_{i=1}^n a_i x_i-b$ for some constants $a_1,\ldots,a_n,b$. Now let v be a node of T in which $p'\leqslant p$ if-questions

are executed. Let $H_1,\ldots, H_{p'}$ be the hyperplanes defined by the p' affine functions defining the if-questions. Then the inputs which choose an edge from v' to a node v form a connected component of $R^n \setminus \bigcup_{i=1}^{p'} H_i$. As their number is at most $(p'+1)^n \leqslant (p+1)^n$ by Lemma 1, each node in T has degree at most $(p+1)^n$, if we remove all nodes v with $c(v)=\varnothing$ from T. Thus if T has complexity t it has at most $(p+1)^{nt}$ leaves. As each set $c(v)$ is the solution set of a system of linear inequalities, it is a convex polytope and therefore connected. Thus if v is an accepting leaf of T, $c(v)$ is contained in one connected component of L, because otherwise it would also contain elements from $R^n \setminus L$. Thus $(p+1)^{nt} \geqslant q$ which proves the theorem. ∎

We now shall present elementary proofs and improvements of the above two theorems for the case that L is defined by m hyperplanes, i.e., if $L = R^n \setminus \bigcup_{i=1}^{m} H_i$, where $H_1,\ldots, H_m$ are hyperplanes in R^n.

COROLLARY 1. *Let $L \subset R^n$ be defined by m hyperplanes. If L has q connected components, then each PCT in R^n with p processors and operation set $\{+, -, *, /.\}$ ($\{+, -\}$) accepting L has complexity at least*

$$\sqrt{\log(m+1)} - \tfrac{1}{2}\log(p)\,(\log(m+1)/\log(p+1)).$$

We first note that by Lemma 1 these results are never worse than those from the two preceeding theorems.

As shown in Remark 1 the bound for the operation set $\{+, -\}$ is tight within a constant factor for $n=1$.

Proof of corollary 1. By elementary arguments from linear algebra one knows that there is a straight line g in R^n which intersects the m hyperplanes defining L in m different points. As recognizing $L \cap g$ for inputs from $g(\hat{=} R)$ is not harder than recognizing L for inputs from R^n and as $L \cap g$ has $m+1$ connected components we obtain the corollary by applying Theorems 1 and 2 for $q = m+1$ and $n=1$. ∎

This trick of reducing the problem in R^n to recognizing a language in R shall allow us later to prove lower bounds for PRAM's over $\{+, -, *\}$, too, which seems to be very hard if inputs from N^n are considered.

We now shall present a lower bound for DCT's.

THEOREM 3. *Let $L \subset R^n$ be defined by m hyperplanes. Then each DCT in R^n with degree c and operation set $\{+, -\}$ which recognizes L has complexity at least* $\log(m+1)/\log(c+1)$ (*independent of the number p of processors*).

Also this result is shown to be tight within a constant factor in

Remark 1. It further shows that if $n=1$ and c is a constant independent of p and q, then a DCT cannot be faster than a (sequential) CT.

Proof of Theorem 3. Applying the trick from above of choosing an appropriate straight line in R^n, we know that it suffices to prove the theorem for $n=1$. In this case L consists of $m+1$ intervals.

Now let T be a DCT as in the theorem which recognizes L in t steps. We assume again that all nodes v with $c(v)=\varnothing$ are removed from T. Now let $h(l)$ be the maximum number of nodes in depth l of some of the T^v's T consists of. We claim that the following property holds:

$$\text{For each } l\in\{1,\dots,t\},\ h(l)\leqslant h(l-1)\cdot(c+1),\ h(0)=1. \tag{1}$$

Before we prove (1), we conclude the theorem from it.

By (1) we know that T^1 has at most $(c+1)^t$ leaves. For each leaf v, $c(v)$ is an interval, thus it is connected. The argument from the proof of Theorem 2 now guarantees that $(c+1)^t\geqslant m+1$ which proves the theorem.

It remains to prove the proposition (1). For $l=0$, (1) is clearly true. Let $l>0$. Then we know as above, that for each node v in depth $l-1$ of T, $c(v)$ is connected. Thus for each $j\in\{1,\dots,p\}$, the sets $c(v)$ for nodes v of T^j in depth $(l-1)$ form a disjoint partition A_j of R in at most $h(l-1)$ connected sets.

Now let $c'\leqslant c$ and $i,j_1,\dots,j_{c'}\in\{1,\dots,p\}$ be chosen such that $P_{j_1},\dots,P_{j_{c'}}$ are the neighbors of P_i in G. Let A be the partition of R in sets of the form $c(v)\cap\bigcap_{h=1}^{c'}c(v_h)$ for nodes v, $v_1,\dots,v_{c'}$ in depth $(l-1)$ of T^i, $T^{j_1},\dots,T^{j_{c'}}$. Then obviously A consists of at most $h(l-1)\cdot(c'+1)\leqslant h(l-1)\cdot(c+1)$ connected sets. Thus T^i has at most $h(l-1)\cdot(c+1)$ nodes in depth l. ∎

Now we apply our results to prove lower bounds for recognizing $L_{n,k}$. Recall that $L_{n,k}=\{(\bar a,b),\ \bar a\in R^n,\ b\in R,\ \exists\bar x\in\{0,\dots,k\}^n,\ \bar x\cdot\bar a=b\}$. $L_{n,k}$ can be expressed as $\bigcup_{\bar x\in\{0,\dots,k\}^n,\bar x\neq\bar 0}H_{\bar x}$, where $H_{\bar x}$ is the hyperplane $\{(\bar a,b),\ \bar a\in R^n,\ b\in R,\ \bar x\cdot\bar a=b\}$ in R^{n+1}. Thus $L_{n,k}$ is defined by $(k+1)^n-1$ hyperplanes. Therefore we obtain the following theorem by applying Corollary 1 and Theorem 3.

THEOREM 4. *The following lower bounds hold for recognizing $L_{n,k}$ in R^{n+1} with p processors. For* PCT*'s with operation set* $\{+,-,*,/\}$:

$$\Omega\left(\sqrt{n\log(k+1)}-\log(p)\right).$$

For PCT*'s with operation set* $\{+,-\}$:

$$\Omega(n\log(k+1)/\log(p+1)).$$

For DCT*'s with operation set* $\{+, -\}$ *and degree* c:

$$\Omega(n \log(k+1)/\log(c+1)).$$

The first lower bound is already shown by Yao (1981) for $k=1$. We note here that PCT's or DCT's cannot recognize L_n, because L_n is defined by an infinite number of hyperplanes. We shall later prove lower bounds for L_n on PRAM's and DRAM's which are expressed in terms of the input size.

We finally state a technical remark which is proved implicitly in the Theorems 1, 2, and 3.

Remark 2. The following upper bounds hold for the number of leaves of PCT's or DCT's in R with p processors (and degree c) with complexity t.

—For PCT's with operation set $+, -, *, /, \}$: $(p+1)^t \cdot 2^{\binom{t+1}{2}}$.

—For PCT's with operation set $\{+, -\}$: $(p+1)^t$.

—For a tree T^t of a DCT with operation set $\{+, -\}$ and degree c: $(c+1)^t$.

IV. Simulating PRAM's and DRAM's by PCT's and DCT's

In this chapter we shall describe a method how to simulate a PRAM or DRAM by a PCT or DCT. This method is described for RAM's and CT's already in Klein, Meyer auf der Heide (1983) or Paul and Simon (1980)). We shall see that a PRAM or DRAM without indirect storage access is almost a PCT or DCT and that we can simulate indirect storage access by direct storage access without loss of time, if we exclude certain subsets of the input set. These sets will be of the following form. Let M be a parallel straight line program, i.e., a PCT without branchings, with p processors, complexity t, and operation set $S \subset \{+, -, *\}$ in which the functions $\{f_i: R^n \to R,\ i=1,\ldots,r\}$ are computed. We assume that the f_i's are pairwise different. Let $f_{r+1},\ldots,f_{r+s}$ be constant functions. Then the set $\{\bar{x} \in R^n, \exists i,j \in \{1,\ldots,r+s\}\ i \neq j,\ f_i(\bar{x}) - f_j(\bar{x}) = 0\}$ is called a (p,t,s)-set in R^n over S. If $S = \{+, -\}$, then the sets $\{\bar{x} \in R^n,\ f_i(\bar{x}) - f_j(\bar{x}) = 0\}$ are hyperplanes or empty, if $S = \{+, -, *\}$, they are solution sets of polynomial equations with degree at most 2^t, because a straight line program can, also if p processors are available, only compute polynomials of degree at most 2^t in t steps.

Remark 3. *A* (p,t,s)-set in R^n over $\{+, -\}$ is the union of at most $(p \cdot t \cdot (p \cdot t + s))$ hyperplanes in R^n, *A* (p,t,s)-set in R^n over $\{+, -, *\}$ is the union of solution sets of at most $(p \cdot t(p \cdot t + s))$ polynomial equations with n variables and degree at most 2^t.

In this chapter we will prove the following theorem about simulating PRAM's and DRAM's by PCT's and DCT's.

THEOREM 5. *Let M be a* PRAM (DRAM) *over $S \subset \{+, -, *\}$ with p processors (and communication graph G) recognizing some $L \subset N^n$ in t steps. Then there is a* PCT (DCT) *T_M over S with p processors (and communication graph G) with complexity t and the following property: Let v be a leaf of $T_M(T_M^1)$. Then there is a $(p, t, p \cdot t)$-set $((p, t, p \cdot (t+1))$-set$)$ F_v in R^n over S—the forbidden inputs for v—such that T_M simulates M for all inputs from $(c(v) \setminus F_v) \cap N^n$.*

Proof. We first prove the theorem for a PRAM as above. This proof is similar to that one shown in Klein and Meyer auf der Heide (1983) for sequential RAM's. Let $P_1, \ldots, P_p$ be the processors of M and let the jth instruction in the program for P_i be $I_{i,j}$.

We now first describe a tree of depth t and label the nodes with p-tuples $(I_{1,j_1}, \ldots, I_{p,j_p})$ of instructions of $P_1, \ldots, P_p$ such that the paths in the tree describe computations of M.

The root v_0 gets no label but one son labelled with $(I_{1,1}, \ldots, I_{p,1})$.

Now suppose that a path from v_0 to a node v is defined and labelled, v let be labelled by $(I_{1,j_1}, \ldots, I_{p,j_p})$. Furthermore let $p' \leqslant p$ I_{q,j_q}'s be *if*-questions, w.l.o.g. the first p' ones. Then v gets $2^{p'}$ sons $v_{\bar{\alpha}}$, $\bar{\alpha} \in \{0, 1\}^{p'}$. For each $q > p'$ and each q with $\alpha_q = 0$ (which means that the else-branch of the if-question and each q with $\alpha_q = 0$ (which means that the else-branch of the if-question is chosen), the qth instruction in the label for $v_{\bar{\alpha}}$ is I_{q,j_q+1}, if $\alpha_q = 1$ then this instruction id $I_{q,\beta}$, where $I_{q,j_q} \hat{=}$ if $\langle 0 \rangle_q$ then goto I_β.

Now we shall attach functions $L_i^v: R^n \to R$ to each node v of the tree, one for each P_i, $i = 1, \ldots, p$. For $\bar{x} \in N^n$, $L_i^v(\bar{x})$ is intended to be $l_i^v(\bar{x})$, the content of the accumulator of P_i, if M started with $\bar{x}$ executes the computation associated to the path to v. Furthermore, if the instruction I_{i,j_i} belonging to the label of v is i-store(k), then an additional function $A_i^v: R^n \to R$ is attached to v. $A_i^v(\bar{x})$ then is intended to be $a_i^v(\bar{x})$, the address used for the indirect storage access. M started with $\bar{x} \in N^n$ is simulated correctly up to v, if for each node w on the path to v and for each $j \in \{1, \ldots, p\}$, $L_j^w(\bar{x}) = l_j^w(\bar{x})$, and if $A_j^w(\bar{x}) = a_j^w(\bar{x})$ for the case that A_j^w is defined.

The root v_0 gets $L_i^v \equiv 0$, $i = 1, \ldots, p$. Up to now we obviously have simulated correctly. Now let v' be a node such that the attachments of functions to all nodes on the path to v' (including v') are done. Let v' be labelled with $(I_{1,j_1}, \ldots, I_{p,j_p})$ and let v be a son of v'.

We now define L_i^v for some $i \in \{1, \ldots, p\}$ and check for which inputs $L_i^v(\bar{x}) = l_i^v(\bar{x})$ and, if defined $A_i^v(\bar{x}) = a_i^v(\bar{x})$ holds. Let $I = I_{i,i_j}$ and let M started with $\bar{y} \in N^n$ be simulated correctly up to v'.

(a) If $I \in \{$store(k), if $\langle 0 \rangle_i > 0$ then goto $I_\beta\}$, then $L_i^v = L_i^{v'}$, if $I = c$-load(k), then $L_i^v \equiv k$. Obviously, in this case $L_i^v(\bar{y}) = l_i^v(\bar{y})$.

(b) If $I = i\text{-store}(k)$, then $L_i^v = L_i^{v'}$. In order to define A_i^v we search for the last vertex w on the path to w and the maximum $j \in \{1,...,p\}$ such that the jth instruction of the label of w is store(k). If w and j exist and L_j^w is not constant, then $A_i^v = L_j^w$. If $L_j^w \equiv k'$, then I is replaced by store(k'). If w, j do not exist and $k \in \{1,...,n\}$, $A_i^v(\bar{x}) = x_k$, else $A_i^v(\bar{x}) = 0$.

One easily checks that $L_i^v(\bar{y}) = l_i^v(\bar{y})$ holds. $A_i^v(y) \neq a_i^v(\bar{y})$ only happens, if on the path from w to v, the address, i.e., the content of register k, is changed by an i-store(k') instruction. In this case the following property holds:

There is some node w'' on the path to v where i-store(k') is executed in some P_j and $A_j^{w''}(\bar{y}) = k$ but $A_j^{w''} \not\equiv k$. (2)

(c) If $I = \text{"}o\text{"}(k)$ for some $o \in S$ ($I = \text{load}(k)$), then we search for the same w and j as in b). If they exist, then $L_i^v = L_i^{v'} o L_j^w$ ($L_i^v = L_j^w$); if they do not exist and $k \in \{1,...,n\}$ then $L_i^v = L_i^{v'} o x_k$ ($L_i^v = x_k$), else $L_i^v = L_i^{v'} o 0$ ($L_i^v \equiv 0$). Again $L_i^v(\bar{y}) = l_i^v(\bar{y})$, if (2) does not hold.

(d) If $I = i\text{-load}(k)$, then we again search for w, j as in (b). If they exist, let w' be the last vertex on the path from w to v and $j' \in \{1,...,p\}$ be maximum such that the j'th instruction in the label of w' is i-store(k') and $A_{j'}^{w'} = L_j^w$. If w', j' exist, $L_i^v = L_{j'}^{w'}$, $L_i^v \equiv 0$ else. In this case, $A_{j'}^{w'}(\bar{y})$ is the correct address used after the computation of M started with $\bar{y}$ up to v, if (2) does not hold. Furthermore $L_i^v(\bar{y}) = l_i^v(\bar{y})$, if the content of the register $A_{j'}^{w'}(\bar{y})$ is never changed by an i-store(k') instruction during the computation between w' and v. That means, $L_i^v(\bar{y})$ is correct, if (2) and the following property do not hold:

There is some node w'' on the path to v and some $j'' \in \{1,...,p\}$ such that $A_{j''}^{w''}(\bar{y}) = A_{j'}^{w'}(\bar{y})$ but $A_{j''}^{w''} \not\equiv A_{j'}^{w'}$. (3)

(e) If $I = \text{stop}$, then $L_i^v = L_i^{v'}$. In this case v is a leaf. If we assume w.l.o.g. that for $i = 1$, the last instruction executed before is c-load(0) or c-load(1), $L_i^v \equiv 0$ or $L_1^v \equiv 1$. In the first case v is rejecting, else accepting. Hereby we have described a PCT T_M with complexity t. Let v_1 be a leaf of T_M, $\bar{y} \in c(v_1) \cap N^n$. By the remarks above we know that M started with $\bar{y}$ is simulated correctly, if for no node on the path to v_1, (2) or (3) holds for $\bar{y}$. The set of inputs which fulfill (2) or (3) for some node v can be characterized as follows: Consider the straight line program with p processors over S described by the path to v_1. Let $f_1,...,f_r$ be computed in this program, and let $f_{r+1},...,f_s$ be the constants used as direct addresses (as k in add(k), i-store(k) etc.) on the path to v. Then $s \leqslant p \cdot t$ and as each

function A_i^v is some f_i, the $(p, t, p \cdot t)$-set in R^n over S associated to $\{f_1,\ldots,f_{r+s}\}$ contains all $\bar{y}$ which fulfill (2) or (3) for some v from the path to v_1. Thus T_M simulates M correctly for each input $\bar{y}$ from $c(v) \cap N^n$, which does not belong to this $(p, t, p \cdot t)$-set. This proves the theorem for PRAM's.

We now prove the analogous result for a DRAM M as described in the theorem. For this purpose we almost copy the above construction for $p = 1$ for each of the processors of M, because each processor is a sequential RAM, as long as it does not execute a read(j), i-read(k), write(j), or i-write(k) instruction. We first modify M in such a way that it executes alternately communication steps, i.e., each processor executes an instruction from {read(j), i-read(k), write(j), i-write(j)} and computation steps, i.e., each processor executes an instruction from the set defined for PRAM's. Communication steps are executed after an odd number of steps, computation steps else.

Analagously to the proof for PRAM's we now first define $T^1,\ldots, T^p$, the p trees of a DCT and label its nodes with instructions. The roots get no label but one son labelled with the first instructions of the programs for the P_i's. Now suppose that the trees are defined and labelled up to depth $(l-1)$. If $(l-1)$ is odd, then a computation step has to be executed, and the nodes in depth l are defined and labelled as described for PRAM's as above when $p = 1$ is assumed.

If $(l-1)$ is even, then let v' be a node in depth $(l-1)$ of T^i for some $i \in \{1,\ldots, p\}$. v' gets a son v for each combination of nodes $v_1,\ldots, v_s$ in depth $(l-1)$ in $T^{j_1},\ldots, T^{j_s}$, where $P_{j_1},\ldots, P_{j_s}$ are the neighbors of P_i in G. If v' is labelled with the qth instruction of the program for P_i, then v is labelled with the $(q+1)$th one. The edge (v', v) is chosen by inputs from $\bigcap_{j=1}^{s} c(v_j) \cap c(v)$. Now let L_i^v, A_i^v, l_i^v, a_i^v be defined analagously to the proof for PRAM's. We want to attach functions L_i^v and, if defined, A_i^v to the nodes v of each T^i.

Additionally we define functions $B_i^v: R^n \to R$ and $b_i^v: R^n \to R$ for nodes belonging to a communication step. $B_i^v(\bar{x})$ is intended to be $b_i^v(\bar{x})$, the actually computed address of a neighbor, P_i wants to write to or to read from when M started with $\bar{x}$ has executed the computation up to v in P_i. Now we suppose that the functions above are attached to all nodes of $T^1,\ldots, T^p$ up to depth $(l-1)$. If $(l-1)$ is odd, then the nodes in depth l in each T^i get functions as described for PRAM's when $p = 1$ is assumed. Now let $(l-1)$ be even, $i \in \{1,\ldots, p\}$, and v' be a node of T^i in depth $(l-1)$ labelled with instruction I. We define B_i^v for a son v of v'. If $I \in$ {read(j), write(j)}, then $B_i^v \equiv j$. This attachment obviously is always correct, i.e., $B_i^v = b_i^v$.

If $I \in \{i\text{-read}(k), i\text{-write}(k)\}$, then we search for a node w in T^i as described in (b) in the above proof for PRAM's. (We need not search for a

j as in (b), because $p=1$.) If $L_i^w \equiv j$ and P_j is a neighbor of P_i, then $B_i^v \equiv j$. In all other cases, B_i^v may be arbitrary, we shall assume $B_i^v \equiv i$.

This attachment only can be incorrect, if the contents of the register k of T^i is changed by an i-store(k') on the path from w to v, i.e., if the property (2) from the proof for PRAM's holds or if some input $\bar{y} \in N^n$, for which M started with $\bar{y}$ passes through v, fulfills

$$L_i^w \text{ is not constant, but } L_i^w(\bar{y}) \in \{j_1,\ldots,j_c\}, \text{ where } p_{j_1},\ldots,p_{j_c} \text{ are the neighbors of } P_i. \tag{4}$$

Now we define L_i^v. For this purpose we assume that v is a son of v' belonging to the nodes $v_1,\ldots,v_c$ of $T^{i_1},\ldots,T^{i_c}$ in depth $(l-1)$ as described in the definition of T. Let w.l.o.g. $v_1,\ldots,v_{c'}$, $c' \leqslant c$ be those nodes labelled with write(i) or i-write(k) and $B_{j_h}^{v_h} \equiv i$, $h=1,\ldots,c'$. Let $j_{c'}$ be maximum among $j_1,\ldots,j_{c'}$. If now I, the label of v', is read(j) or i-read(j) and

$$B_i^{v'}(\bar{y}) \geqslant j_{c'}, \text{ then } L_i^v = L_i^{v'}, \text{ else } L_i^v = L_{j_c}^{v_c}.$$

This attachment obviously is correct for inputs $\bar{y}$ which do not fulfill (4) and which are simulated correctly up to depth $(l-1)$. The leaves of T^1 are accepting or rejecting as described for PRAM's in (e).

Now let v be a leaf of T^1. Let $v_2,\ldots,v_p$ be the deepest nodes in $T^2,\ldots,T^p$ such that $c(v) \subset c(v_2),\ldots,c(v) \subset c(v_p)$ holds. Then, as by a communication step in a node v after which $L_j^{v'}$ for some node v' from some T^j is stored, $c(v)$ becomes restricted to a subset of $c(v')$, only the computations in $T^2,\ldots,T^p$ up to $v_2,\ldots,v_p$ affect the computation to v in T^1. Thus for $\bar{y} \in c(v) \cap N^n$, M started with $\bar{y}$ is simulated correctly, if $\bar{y}$ does not fulfill (2), (3), or (4) for some node on the paths to v or $v_2,\ldots,v_p$ in $T^1,\ldots,T^p$.

Let H be the straight line program in R^n over S with p processors and length t, which consists of executing the p computations to v and the v_i's in parallel. Let this program compute $\{f_1,\ldots,f_r\}$. Let $\{f_{r+1},\ldots,f_{r+s}\}$ be the constants used in this program as direct addresses. Then $s \leqslant p \cdot t$ and the associated $(p, t, p \cdot t)$-set contains all inputs which fulfill (2) or (3). If we add the contants $1,\ldots,p$ to this set, the associated $(p, t, p \cdot (t+1))$-set contains all inputs which fulfill (2), (3), or (4). Thus M started with $\bar{y} \in c(v) \cap N^n$ is simulated correctly, if it does not belong to the above $(p, t, p \cdot (t+1))$-set, which proves the theorem. ∎

V. Lower Bounds for PRAM's and DRAM's

In this chapter we apply the results about PCT's and DCT's from the third section and the simulation result from the fourth section for proving lower bounds for PRAM's and DRAM's recognizing $L_{n,k}$ and L_n. For this

purpose we again apply the trick of choosing a straight line g in R^n and of considering only inputs belonging to this line. We denote the line segment between two points $\bar{x}, \bar{y} \in R^n$ by $(\bar{x}, \bar{y})$. ($\bar{x}$ and $\bar{y}$ do not belong to $(\bar{x}, \bar{y})$.)

LEMMA 2. *Let $L = R^n \setminus \bigcup_{i=1}^{m} H_i$ be defined by m hyperplanes in R^n. Let $B \subset N^n$ and g a straight line in R^n intersecting $H_1, \ldots, H_m$ in m different points $\bar{x}_1, \ldots, \bar{x}_m \in B$, and for $i \neq j$, $\#(x_i, x_j) \cap B \geq \frac{1}{2}m + 1$. Let M_1 be a* PRAM *with operation set $\{+, -, *\}$, M_2 a* PRAM *with operation set $\{+, -\}$, and M_3 a* DRAM *with operation set $\{+, -\}$ and degree c. If M_1, M_2 and M_3 have p processors and recognize L for inputs from B, then the following lower bounds hold:*

(i) *M_1 needs at least $\sqrt{\log(m)/2 - 1} - \frac{1}{2}\log(p)$ steps.*

(ii) *M_2 needs at least $\frac{1}{2}(\log(m) - \frac{1}{2}\log(p+1))$ steps.*

(iii) *M_3 needs at least $\frac{1}{2}(\log(m) - \frac{1}{2}\log(c+1))$ steps.*

In order to prove the lemma we first consider the PCT's and the DCT T_1, T_2, and T_3 attached to M_1, M_2, and M_3 in Theorem 5. Let v_1, v_2, v_3 be leaves of T_1, T_2, or T_3^1 and $F_{v_1}, F_{v_2}, F_{v_3}$ be the sets of forbidden inputs for v_1, v_2, v_3. Then by Remark 3 and Theorem 5,

(a) $\#(F_{v_1} \cap N^n \cap g) \leq 3(pt)^2 \cdot 2^t$

(b) $\#(F_{v_2} \cap N^n \cap g) \leq 3(pt)^2$

(c) $\#(F_{v_3} \cap N^n \cap g) \leq 3(pt)^2$,

where t denotes the complexity of T_1, T_2, T_3.

Applying Remark 2 we furthermore know that

(α) T_1 has at most $3 \cdot p^t \cdot p^{t(t+1)} \cdot (pt)^2 \cdot 2^t \leq 2^{2t^2} p^{2t}$,

(β) T_2 has at most $p^t 3(pt)^2$, and

(γ) T_3 has at most $(c+1)^t\, 3(pt)^2$

forbidden inputs at all on g.

Now we consider first M_1 and suppose that

$$2^{2t^2} \cdot p^{2t} > \tfrac{1}{2}m. \tag{5}$$

Then it follows directly that

$$t > \sqrt{\log(m)/2 - 1} - \tfrac{1}{2}\log(p).$$

If (5) does not hold, then by (1), $B^* = B \setminus \{\text{forbidden inputs for } M_1\}$ contains at least $\frac{1}{2}m$ elements from $\{x_1, \ldots, x_m\}$, w.l.o.g. $x_1, \ldots, x_{m'}$, $m' \geq \frac{1}{2}m$. Let w.l.o.g. $x_1, \ldots, x_{m'}$ be ordered consecutively on g. Then for $i \in \{1, \ldots, m'-1\}$ $(x_i, x_{i+1}) \cap B^*$ contains at least one element because at most $\frac{1}{2}m$ of the at

least $\frac{1}{2}m+1$ elements of $(x_i, x_{i+1}) \cap B$ can be forbidden inputs because of (α) and (5). Thus the language $L' \subset R^n$ recognized by T_1 fulfills that $L' \cap g$ has at least $m' \geqslant \frac{1}{2m}$ connected components on g thus (i) follows from Corollary 1.

Analogously (ii) and (iii) follow by (β) and Corollary 1 resp. (γ) and Theorem 3. ∎

Now we are able to prove lower bounds for $L_{n,k}$ and L_n.

THEOREM 6. *Let M_1 be a* PRAM *with operation set $\{+, -, *\}$, M_2 a* PRAM *with operation set $\{+, -\}$, and M_3 a* DRAM *with operation set $\{+, -\}$ and degree c. Let M_1, M_2, and M_3 have p processors. If they recognize $L_{n,k}$ or L_n for inputs from $\{0,..., (k+1)^{2n}\}^n$, then*

(i) *M_1 needs at least $\sqrt{n \log(k+1)/2 - 1} - \frac{1}{2}\log(p)$ steps.*

(ii) *M_2 needs at least $(\frac{1}{2}n \log(k+1) - 1)/\log(p+1)$ steps.*

(iii) *M_3 needs at least $(\frac{1}{2}n \log(k+1) - 1)/\log(c+1)$ steps.*

Proof. For the bounds for $L_{n,k}$ we have to find a set B and a straight line g in R^n with the properties demanded in Lemma 2. For this purpose, for $h \in N$, let g_h denote the straight line $\{(h, h(k+1), h(k+1)^2,..., h(k+1)^{n-1}, x), x \in R\}$ in R^{n+1}. Recall that for $\bar{\alpha} \in \{0,..., k\}^n \setminus \{\bar{0}\}$ the hyperplane $H_{\bar{\alpha}}$ in R^{n+1} is defined by the equation $\sum_{i=1}^{n} \alpha_i x_i - x_{n+1} = 0$ and that $L_{n,k}$ is the union of all such $H_{\bar{\alpha}}$'s.

For $\bar{\alpha} \in \{0,..., k\}^n \setminus \{\bar{0}\}$ let $\bar{x}_{\bar{\alpha}}$ be the intersection point of $H_{\bar{\alpha}}$ and g_h. Then its only variable component, the $(n+1)$th one, is a multiple of h, and the $\bar{x}_{\bar{\alpha}}$'s are pairwise different. Thus choosing $g = g_h$ with $h = \frac{1}{2}(k+1)$ and $B = \{0,..., (k+1)^{2n}\}^n$ we may conclude the lower bounds for $L_{n,k}$ from Lemma 2.

In order to prove the lower bounds for L_n, we apply a trick from Meyer auf der Heide (1985), where a lower bound for L_n on (sequential) RAM's is shown.

Let w.l.o.g. $(k+1)$ be even and

$$B^* = (\{0, 2, 4,..., (k+1)^{2n}\}^{n-1} \times \{1, 3, 5,..., (k+1)^{2n} - 1\}) \cup (L_{n,k} \cap \{0,..., (k+1)^{2n}\}^n).$$

Then the above lower bounds for $L_{n,k}$ also hold for inputs from B^*. But for $(\bar{a}, b) \in B^*$ it holds that $(\bar{a}, b) \in L_{n,k} \Leftrightarrow (\bar{a}, b) \in L_n$.

"$\Rightarrow$" holds because $L_{n,k} \subset L_n$.

"$\Leftarrow$" holds because, if $(\bar{a}, b) \notin L_{n,k}$, then by definition of B^*, $\bar{a} \cdot \bar{x}$ is even for each $\bar{x} \in N^n$ and b is odd. Thus $\bar{a} \cdot \bar{x} \neq b$ for each $\bar{x} \in N^n$ which implies $(\bar{a}, b) \notin L_n$. Thus the lower bounds also hold for L_n. ∎

RECEIVED: July, 1984; ACCEPTED Sept. 2, 1985

REFERENCES

SHILOACH, Y. AND VISHKIN, U. (1981), Finding the maximum, merging, and sorting in a parallel computation model, *J. Algorithms* **2**, 88–102.

FORTUNE, S., AND WILLIE, J. (1978), Parallelism in random access machines, *in* "Proc. of the 10-th ACM-Sympos. Theory of Comput." pp. 114–118.

REISCHUK, R. (1984), Simultaneous writes of parallel random access machines do not help to compute simple arithmetic functions, *J. Assoc. Comput. Mach.*, in press.

GALIL, Z., AND PAUL, W. J. (1983), A general purpose parallel computer, *J. Assoc. Comput. Mach.* **30**, No. 2, 360–387.

MEYER AUF DER HEIDE, F. (1983), Efficiency of universal parallel computers, *Acta Inform.* **19**, 269–296.

MEYER AUF DER HEIDE, F. (1986), Efficient simulations among several models of parallel computers, *SIAM J. Comput.*

GAREY, M. R., AND JOHNSON, D. S. (1979) "Computers and Intractability, a Guide to the Theory of NP-Completeness," Freeman San Francisco.

DOBKIN, D., AND LIPTON, R. (1975), A lower bound of $\frac{1}{5}n^2$ on linear search programs for the knapsack problem, *J. Comput. System Sci.* **16**, 417–421.

BEN OR, M. (1983), Lower bounds for algebraic computation trees, *in* "Proc. of the 15-th ACM Sympos. Theory of Comput.", pp. 80–86.

YAO, A. C. C. (1981), On the parallel computation of the knapsack problem, *in* "Proc. of the 13-th ACM Sympos. Theory of Comput., pp. 123–127.

MEYER AUF DER HEIDE, F. (1985), Lower bound for solving linear Diophantine equations on random access machines, *J. Assoc. Comput. Mach.* **32**, 929–937.

KLEIN, P., AND MEYER AUF DER HEIDE, F. (1983), A lower time bound for the knapsack problem on random access machines, *Acta Inform.* **19**, 385–395.

PAUL, W. J., AND SIMON, J. (1980), Decision trees and random access machines, *in* "Symposium über Logik und Algorithmik," Zürich.

AHO, A., HOPCROFT, J. E., AND ULLMAN, J. D. (1974), "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass.

BATCHER, K. (1968), Sorting networks and their applications, AFIPS Spring Joint Comput. Conf. **32**, 307–314.

WAKSMAN, A. (1968), A permutation network, *J. Assoc. Comput. Mach.* **15**, No. 1, 159–163.

PREPARATA, F. P., AND VUILLEMIN, J. (1981), The cube-connected cycles, a versatile network for parallel computation, *Comm. ACM* **24**, 300–310.