INFORMATION AND COMPUTATION 101, 123–129 (1992)

k versus $k+1$ Index Registers and Modifiable versus Non-modifiable Programs*

K. MEHLHORN, W. J. PAUL, AND C. UHRIG

Fachbereich 14 Informatik, Universität des Saarlandes, 6600 Saarbrücken, Germany

We compare Random Access Machines with k or $k+1$ index registers and modifiable or non-modifiable programs and show for a simple problem of data transfer that the more prowerful versions are more efficient. © 1992 Academic Press, Inc.

I. INTRODUCTION

Complexity theory has addressed a number of architectural questions in the past: the power of an additional tape or head in Turing machines, the power of multidimensional versus one-dimensional tapes, the power of two-way versus one-way input, the power of queues and stacks versus tapes (Aanderaa, 1979; Li and Vitányi, 1988; Maass, 1987; Paul, 1982, 1984). In this note we consider more realistic machine models than Turing machines. More specifically, we show that an additional index register or the ability to modify its program strictly increases the power of Random Access Machines.

We now give the precise statements of the results. The proofs are contained in the next section.

A RAM is characterized by three parameters r, k, and n and is denoted $M_r(k, n)$. It consists of a CPU with r registers $R_1, ..., R_r$ and a memory $D[0, ..., 2^n-1]$ of 2^n words. Each register and memory cell can hold a value in $[0, ..., 2^n-1]$. Out of the r registers the first k can be used as index register.

Table 1 gives the instructions that are available to access the memory, that is, to load data from and store data into the memory. Besides these load and store instructions we allow an arbitrary number of instructions affecting only the processing unit, i.e., data and index registers and program counter. (Note that we do not allow instructions which access the memory and simultaneously change the CPU in a complex way, e.g., $R_i \leftarrow R_i + D[R_j]$. That is a limitation of our proof technique which we address in the final section.)

* This research was supported by the DFG, SFB 124, Projects B2 and D4.

TABLE 1

Load and Store Instructions

Instruction		Effect
LOAD & INC	i, j, c	$R_i \leftarrow D[R_j + c]$; $R_j \leftarrow R_j + 1$; $PC \leftarrow PC + 1$
LOAD	i, j, c	$R_i \leftarrow D[R_j + c]$; $PC \leftarrow PC + 1$
STORE & INC	i, j, c	$D[R_j + c] \leftarrow R_i$; $R_j \leftarrow R_j + 1$; $PC \leftarrow PC + 1$
STORE	i, j, c	$D[R_j + c] \leftarrow R_i$; $PC \leftarrow PC + 1$

Note. $0 \leqslant j \leqslant k$, $1 \leqslant i \leqslant r$. R_i is the ith register, D is the memory, and PC is the program counter. c is an integer. R_0 is a shorthand for 0.

We use the unit cost measure to compute the cost of the computation.

In an Aiken machine, the program is fixed throughout execution; i.e., program and data are stored in different memories and LOADs and STOREs only affect the data memory. In von Neumann machines there is no such restriction; more precisely, we assume that the instructions have numerical codes and that the additional instruction

	Effect
LOADINSTR i, j	$P[i] \leftarrow R_j$

loads the instruction whose numerical code is contained in R_j into the ith cell of the program memory P.

Programs for either machine type are written for fixed values of r and k, but must work for arbitrary values of n.

We consider the following simple h-block transfer problem.

Given integers N, $b_0, ..., b_h$ in $D[0], ..., D[h+1]$ such that $b_{i-1} + N \leqslant b_i$ and $b_h + N \leqslant 2^n$, move the contents of $D[b_0 + j]$ into $D[b_i + j]$ for $0 \leqslant j < N$ and $1 \leqslant i \leqslant h$. N is called the size of the problem.

THEOREM 1. (a) *Let c, k, and r be positive integers with $k + 3 \leqslant r$. Then there is an Aiken program A_0 for $M_r(k+1, *)$ which for all n solves any k-block transfer problem of size $N = c \cdot n$ in time $k + 3 + (k + 1 + 2/c) \cdot N$ on the Aiken machine $M_r(k+1, n)$.*

(b) *Let c and k be positive integers. Then there are a constant d and a von Neumann program V_0 for $M_2(1, *)$ which for all n solves any k-block transfer problem of size $N = c \cdot n$ in time $d \cdot c \cdot k + (k + 1 + 3/c) \cdot N$ on the von Neumann machine $M_2(1, n)$.*

THEOREM 2. *For all r there is a $\beta(r) > 0$ such that for all c and $k \leqslant r$ and any Aiken program A for $M_r(k, *)$ the following holds: for all sufficiently* *large n there is a k-block transfer problem of size $N = c \cdot n$ for which A on the Aiken machine $M_r(k, n)$ requires time at least $(k + 1 + \beta(r)) \cdot N$.*

Theorems 1 and 2 together separate Aiken and von Neumannn machines and also Aiken machines with different numbers of index registers. Let k and $r \geqslant k+3$ be fixed. Choose c such that $3/c < \beta(r)$. In this situation Theorem 2 implies for all large n the existence of a k-block transfer problem of size $c \cdot n$ which takes at least $(k+1+\beta(r)) \cdot N$ on Aiken machine $M_r(k, n)$. However, by Theorem 1, on the Aiken machine $M_r(k+1, n)$ this problem can be solved in time $O(k) + (k+1+2/c) \cdot N$ and on von Neumann machine $M_2(1, n)$ this problem can be solved in time $O(c \cdot k) + (k+1+3/c) \cdot N$, which in both cases is strictly less for n sufficiently large.

II. Proofs

Proof of Theorem 1. (a) The program is quite straightforward. We use $k+1$ index registers $R_1, ..., R_{k+1}$ and two additional registers R_{r-1} and R_r. We step through the $k+1$ blocks in lock-step fashion and use the ith index register to index the $(i-1)$th block, $1 \leqslant i \leqslant k+1$. In each iteration we first load register R_{r-1} from block B_0 and then store the register in all other blocks; i.e., each iteration takes $k+1$ steps. Register r is loaded with N initially and is decremented by c after c iterations. It is also tested for zero after c iterations. Thus c words are transferred in $c(k+1)+2$ time units. The details are as follows:

```
              (1)LOAD                  r, 0, 0}  R_r ← D[0](D[0] = N)
              (2)LOAD                  1, 0, 1 ⎫
                 ⋮                             ⎬ R_i ← D[i] for 1 ≤ i ≤ k + 1
          (k + 2)LOAD          k + 1, 0, k + 1 ⎭
          (k + 3)LOAD & INC          r − 1, 1, 0 ⎫
          (k + 4)STORE& INC          r − 1, 2, 0 ⎪
                 ⋮                               ⎬ move one word
         (2k + 3)STORE & INC     r − 1, k + 1, 0 ⎭
         (2k + 4)LOAD & INC          r − 1, 1, 0 ⎫
                 ⋮                               ⎬ move one word
                                                 ⎭
                 ⋮                                      ⋮
                 ⋮                               ⎫
(k + 2 + (k + 1) c)STORE & INC   r − 1, k + 1, 0 ⎬ move one word
(k + 3 + (k + 1) c)DEC                  r, c}  R_r ← R_r − c
                   JNZ              r, k + 3}  goto k + 3 if R_r > 0
                   STOP
```

The program above clearly runs in time $k + 3 + (k + 1 + 2/c) \cdot N$.

(b) V_0 uses a single index register R_1 and one additional register R_2. The main loop is very similar to the main loop of program A_0 of part (a). If R_1 has a value between 0 and $N-1$, say x, then one word can be moved by

```
LOAD          2, 1, $b_0$        $R_2 \leftarrow D[b_0+x]$
STORE         2, 1, $b_1$        $D[b_1+x] \leftarrow R_2$
                 ⋮                  ⋮
STORE         2, 1, $b_{k-1}$    $D[b_{k-1}+x] \leftarrow R_2$
STORE & INC   2, 1, $b_k$        $D[b_k+x] \leftarrow R_2; x \leftarrow x+1$
```

and c copies of this piece of code will move c consecutive words and also increment R_1 by c. There is a difficulty, however. The quantities $b_0, ..., b_k$ are not known when the program is written, but are only known when the program is started. The solution is, of course, that a von Neumann machine can use its ability to modify the program in order to generate the code listed above. Time $d \cdot c \cdot k$ for some constant d certainly suffices to do so. Having moves c words, we subtract $N(=D[0])$ from R_1, test for zero, restore the old value of R_1, and repeat. Thus c words can be transferred in $c \cdot (k+1)+3$ times units. ▮

Proof of Theorem 2. We first give an informal outline of the proof. Clearly, each word $D[b_0+j]$, $0 \leqslant j < N$, has to be loaded once and stored k times for a total of $(k+1) \cdot N$ time steps. We call these steps standard. In order to count non-standard steps we divide the computation into intervals of length $D := 1 + r(2k+1)$ and assume that the majority of them do not contain a non-standard step; in the other case, the lower bound follows immediately. We then identify for each such interval a cell $D[b_0+j]$ such that this cell is loaded in the interval, its contents are not stored in the CPU at the end of the interval, and yet they were not stored in all the k other blocks by the end of the interval. Thus the value $D[b_0+j]$ must be reestablished in the CPU at some later time and this is necessarily done by a non-standard step. In this way, we associate a non-standard step with every interval and obtain a lower bound. The details follow.

Let A be any Aiken program for $M_r(k, *)$ which solves the k-block transfer problem. Let M be the maximal constant that appears in A; let $b_0 \geqslant k+1$, $b_i \geqslant b_{i-1}+N+M$ for $i=1, ..., k$ and $b_k+N \leqslant 2^n$. We choose the contents of $D[b_0], ..., D[b_0+N-1]$ so that

$$D[b_0+j] \notin [b_i - M \cdots b_i + N] \qquad \text{for} \quad 0 \leqslant j \leqslant N-1, 0 \leqslant i \leqslant k,$$

and

$$D[b_0+j] \neq D[b_0+j'] \qquad \text{for} \quad 0 \leqslant j < j' \leqslant N-1.$$

Let B_0^j be the initial contents of $D[b_0+j]$, let $B_0=\{B_0^0, ..., B_0^{N-1}\}$, and let T be the computation time.

A step t, $1 \leqslant t \leqslant T$, is called *standard* if the instruction executed at time t is a load from $D[b_0+j]$ for the first time in the computation for some j, $0 \leqslant j < N$, or if it is the first store of the value B_0^j into $D[b_i+j]$ for some i, $1 \leqslant i \leqslant k$, and j, $0 \leqslant j < N$. There are clearly exactly $(k+1)\cdot N$ standard steps in the computation.

Our goal is to prove a lower bound on the number of non-standard steps. For this matter we divide the computation into intervals of length $D := 1 + r\cdot(2k+1)$. Let interval I_j comprise steps $(j-1)D+1$ to $\min(j\cdot D, T)$, where $1 \leqslant j \leqslant \rho := \lceil T/D \rceil$. Let d_j be the number of non-standard steps in I_j. Then

$$T \geqslant (k+1)\cdot N + \sum_j d_j.$$

An interval has type I if it contains at least one non-standard instruction and it has type II otherwise. We now distinguish cases according to the type of the majority of the intervals.

Case I. At least $\lceil \rho/2 \rceil$ intervals have type I. Then $\sum_j d_j \geqslant \rho/2$ and hence

$$\begin{aligned} T &\geqslant (k+1)\cdot N + T/2D \geqslant (k+1+(k+1)/(2D))\cdot N \\ &\geqslant (k+1+1/(4r))\cdot N. \end{aligned}$$

Case II. At least $\lceil \rho/2 \rceil$ intervals have type II. Then there are at least $\lceil \rho/2 \rceil - 1$ intervals of type II which have length D (since only the last interval may have length $< D$). Let I be any such interval.

LEMMA 1. *There is an index j, $0 \leqslant j < N-1$, such that*:

(1) *the standard load of B_0^j is contained in I*

(2) *the standard store into $D[b_i+j]$ is not contained in I for some i*

(3) *no register of the CPU contains B_0^j at the end of the interval I.*

Proof. Let i, $0 \leqslant i \leqslant k$, be such that no register $R_1, ..., R_k$ contains a value v with $v+m \in [b_i \cdots b_i+N-1]$ for some $m, 0 \leqslant m \leqslant M$, at the beginning of interval I.

We show first that I contains no standard step involving B_i. Assume otherwise. Then there must be a minimal $t \in I$ such that after step t some R_l, $1 \leqslant l \leqslant k$, contains a value v such that

$$v+m \in [b_i \cdots b_i+N-1] \qquad \text{for some} \quad m, 0 \leqslant m \leqslant M.$$

The tth step either loaded some B_0^p into R_l, i.e., $v = B_0^p$, or increased R_l by one, or did not change R_l. The first alternative contradicts our choice of the B_0^p's, and the third alternative contradicts the definition of t. In the second alternative we must have

$$v+m-1 \notin [b_i, ..., b_i+N-1]$$

for all m, $0 \leqslant m \leqslant M$ by the definition of t, and hence

$$v+M=b_i.$$

Also, since R_l was used as an index register in the tth instruction and hence

$$v+m'-1 \in [b_h, ..., b_h+N-1]$$

for some m', $0 \leqslant m' \leqslant M$, and some $h \neq i$, this is a contradiction to our choice of the b's. In either case we have derived a contradiction and hence there cannot be any standard step involving B_i.

We now turn to the existence of index j. For this matter we distinguish the cases $i=0$ and $i \neq 0$.

Assume first that $i=0$. Since all instructions in I are standard and none of them involves B_0, all instructions in I are (standard) store instructions. We say that a register R_j, $1 \leqslant j \leqslant k$, is used as an index register at step $t \in I$ if an instruction Inst $*, j, *$ is executed at step t. In this case the value of R_j lies in

$$[b_l - M, ..., b_l + N]$$

for some l, $1 \leqslant l \leqslant k$, before and after the tth instruction, and hence not in B_0. Thus if I contains an instruction STORE $j, *, *$ or STORE & INC $j, *, *$, then R_j cannot be used as an index register in I and hence the value of R_j does not change during I. Thus I can contain at most $r \cdot k < D$ instructions, a contradiction.

Assume next that $i > 0$. Let m be the number of LOAD instructions in I. As in case $i=0$ we can argue that there are at most $(r+m) \cdot k$ STORE instructions in I. Since $m + (r+m) \cdot k \geqslant D$ we conclude $m \geqslant r+1$. Then one of the m words loaded from B_0 cannot be contained in the CPU at the end of I. The index of this word is the desired j. ∎

For each but the last interval I of type II let $j(I)$ be an index which satisfies the properties of Lemma 1, and let $t(I) > \max I$ be a step such that $B_0^{j(I)}$ is contained in a register of the CPU after step $t(I)$, but not before step $t(I)$. The step $t(I)$ clearly exists, since not all standard stores of $B_0^{j(I)}$ have taken place by the end of interval I and since no register of the CPU contains the value $B_0^{j(I)}$ at the end of interval I. Also the step $t(I)$ must be a non-standard step (by an argument similar to the proof of Lemma 1) and there can be at most r distinct intervals of type II which have the same value $t(I)$, since the CPU has only r registers. (It is tempting to claim that there can be at most one interval. This is fallacious, however, since we allowed instructions which change all registers of the CPU and hence it is possible that r values appear at the same step.) Thus there are at least $(\lceil \rho/2 \rceil - 1)/r$ non-standard steps. For $\rho > 2$, this is $\geqslant \lceil \rho/4 \rceil / r$ and hence

$$T \geqslant (k+1+(k+1)/(4Dr)) \cdot N \geqslant (k+1+1/(8r^2)) \cdot N.$$

With $\beta(r) = 1/(8r^2)$ this completes the proof of Theorem 2.

III. Discussion

A preliminary version of this paper was presented at ICALP 89 (Mehlhorn and Paul, 1989). In that paper only the cases of one and two registers were dealt with. Even for that case the proof given was considerably more complex than the present proof, and hence the present paper may be considered progress. However, we have to admit that our result is still quite weak since our proof makes essential use of the fact that a LOAD or STORE affects the CPU in a very limited way. In particular, our proof technique cannot handle the situation where a LOAD may affect the entire CPU, i.e., the content of the CPU after a LOAD is a function of the previous content and the word loaded. In this general situation the problem remains open.

Received April 17, 1989; final manuscript received December 18, 1990

References

Aanderaa, S. O. (1974), On k-tape versus $(k-1)$-tape real-time computation, *in* "Complexity of Computation," pp. 75–96.

Li., M., and Vitányi, P. M. B. (1988), Tape versus queue and stacks: The lower bounds, *Inform. and Comput.* **78**, 56–85.

Maass, W., Schnitger, G., and Szemeredi, E. ((1987), Two tapes are better than one for off-line Turing machines, *in* "Proc. 19th ACM Symposium on Theory of Computing," pp. 94–100.

Mehlhorn, K., and Paul, W. J. (1989), Two versus one index register and modifiable versus non-modifiable programs, *in* "Proc. 16th International Colloquium on Automata, Languages and Programming, LNCS 372," pp. 603–609.

Paul, W. J. (1982), On-line simulation of $k+1$ tapes by k tapes requires non-linear time, *Inform. Contr.* **53**, 1–8.

Paul, W. J. (1980), On heads versus tapes, *Theoret. Comput. Sci.* **28**, 1–12.

Printed in Belgium
Uitgever: Academic Press, Inc.
Verantwoordelijke uitgever voor België:
Hubert Van Maele
Altenastraat 20, B-8310 Sint-Kruis