EFFICIENT SIMULATIONS OF MULTICOUNTER MACHINES*)

(Preliminary version)

Paul M.B. Vitányi
Mathematisch Centrum
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

ABSTRACT

An oblivious 1-tape Turing machine can on-line simulate a multicounter machine in linear time and logarithmic space. This leads to a linear cost combinational logic network implementing the first n steps of a multicounter machine and also to a linear time/logarithmic space on-line simulation by an oblivious logarithmic cost RAM. An oblivious $\log^* n$-head tape unit can simulate the first n steps of a multicounter machine in real-time, which leads to a linear cost combinational logic network with a constant data rate.

1. INTRODUCTION

In many computations it is necessary to maintain several counts such that, at all times, an instant signal indicates which counts are zero. Keeping k counts in tally notation, where a count is incremented/decremented by at most 1 in each step, governed by the input and the set of currently zero counts, is formalized in the notion of a k-counter machine [2]. Multicounter machines have been studied extensively, because of their numerous connections with both theoretical issues and more or less practical applications. The purpose of this paper is to investigate the dependence of the required time and storage, to maintain counts, on storage structure and organization and the cost required by a combinational logic network. To do this, we use a notion of auxiliary interest: that of an oblivious Turing machine. An oblivious Turing machine is one whose head movements are fixed functions of time, independent of the inputs to the machine. The main result obtained here shows that an oblivious Turing machine with only *one* storage tape can simulate a k-counter machine on-line in linear time and in storage logarithmic in the maximal possible count. These bounds are optimal, up to order of magnitude, also for on-line simulation by nonoblivious machines.

It is obvious that, for any time function T(n), given a k-counter machine, or a k-pushdown store machine, which operate in time T(n), we can find a time equivalent k-tape Turing machine. However, such a Turing machine will, apart from using k tapes, also use $O(T(n))$ storage. In [7] it was shown that for the pushdown store, of which

---

*) Registered at the Mathematical Centre as Report.

the contents can not be appreciably compacted, the best we can do for on-line simulation by an oblivious Turing machine is 2 storage tapes, $\Theta(T(n) \log T(n))$ time and $\Theta(T(n))$ storage. For the multicounter machine, [2] demonstrated a linear time/logarithimic space simulation by a 1-tape Turing machine. [9, Corollary 2] showed how to simulate on-line a T(n) time-, S(n) storage-bounded multitape Turing machine by an oblivious 2-tape Turing machine in time $O(T(n) \log S(n))$ and storage $O(S(n))$. Combining the compacting of counts in [2] and the method of [9] we achieve the best previously known on-line simulation of a k-counter machine by an oblivious Turing machine: 2 tapes, $O(T(n) \log \log T(n))$ running time and $O(\log T(n))$ storage. It is somewhat surprising to see that we can restrict a Turing machine for on-line simulation of a k-counter machine to 1 storage tape, logarithmic storage, oblivious head movements and still retain a linear running time.

In Section 2 this result is derived and connected with a linear cost combinational network for doing the same job. This network processes the inputs in sequence and may incur a time delay of $\Theta(\log n)$ between processing and input and producing the corresponding output followed by the processing of the next input. Since we would like to obtain a constant data rate, i.e., a constant time delay between processing the i-th input at the i-th input port and producing the i-th output at the i-th output port, $1 \leq i \leq n$, we show in Section 3 how to real-time simulate n steps of a multicounter machine by an oblivious $\log^* n$-head tape unit and use this to obtain a linear cost combinational network with such a fast response time. It is not our purpose here to introduce an odd machine model with a variable number of access pointers. One should rather think of it as an expedient intermediate step to derive the desired result for fixed n. Subsequently we note that cyclic networks (or VLSI where the length of the wires adds to the cost) can real-time simulate a multicounter machine in logarithmic (area) cost.

In Section 5 we analyse the cost of on-line simulation of a multicounter machine by a logarithmic cost RAM. This turns out to be $O(n)$ time and $O(\log n)$ space on the oblivious version, which is optimal, also for nonoblivious RAMs. For the relevant definitions of multicounter machines [1,2], multitape Turing machines [8], combinational logic networks [7], real-time and linear time on-line simulation [7] and oblivious computations [7,9,10] we direct the reader to these references. The present paper is a preliminary draft; the results in Sections 2 and 4 appeared in Techn. Report IW 167, Mathematical Centre, Amsterdam, May 1981.

2. LINEAR-TIME ON-LINE SIMULATION BY AN OBLIVIOUS ONE-HEAD TAPE UNIT WITH AN APPLICATION TO COMBINATIONAL LOGIC NETWORKS

We first point out one of the salient features of the problem of simulating k-CM's on-line by efficient oblivious Turing machines. Suppose we can simulate some abstract storage device S on-line by an efficient oblivious Turing machine M. Then we can also simulate a collection of k such devices $S_1, S_2, \ldots, S_k$, interacting through

a common finite control, by dividing all tapes of M into k tracks, each of which is a duplicate of the corresponding former tape. Now the same head movements do the same job on k collections of tracks as formerly on the tapes of M, so the time and storage complexity of the extended M are the same as those of the original. While the problem of, say, simulating a k-counter machine in linear time by a k'-tape Turing machine $k' < k$, stems precisely from the fact that k' is less than k, the problem of simulating a k-counter machine by a k'-tape oblivious Turing machine in linear time is the same problem as that of simulating a 1-counter machine in linear time by a k'-tape oblivious Turing machine. Hence, for a proof of feasibility it suffices to look for the simulation of 1 counter only. (For a proof of infeasibility we would have the advantage of knowing that the head movements are fixed, and are the same for all input streams. Besides, we could assume that we needed to simulate an arbitrary, albeit fixed, number of counters.)

In [2] it was shown that a 1-TM can simulate a k-CM on-line in linear time. This simulation uses $O(\log n)$ storage, for n steps by the k-CM, which is clearly optimal. It is a priori by no means obvious that an oblivious multitape TM can simulate one counter in linear time. We shall show that the result of [2] can be extended to hold for oblivious Turing machines.

In our investigation we noted that head-reversals are not necessary to maintain counters. We did not succeed in getting the idea below to work in an oblivious environment, and include it here as a curiosity, possibly folklore, item.

Suppose we want to simulate a k-CM C with counts $x_1, x_2, \ldots, x_k$ represented by the variables n_1 through n_k. The number of simulated steps of C is contained in the variable n. For $i = 1,2,\ldots,k$ if count x_i is incremented by $\delta \in \{-1,0,+1\}$ then

$$\begin{aligned} n_i &\leftarrow n_i + 2 \quad \text{for } \delta = +1 \\ n_i &\leftarrow n_i + 1 \quad \text{for } \delta = 0 \\ n_i &\leftarrow n_i \quad \text{for } \delta = -1 \end{aligned}$$

Let, for $i = 1,2,\ldots,k$, $\hat{x}_i$ denote the current count on the i-th counter of C.

PROPOSITION 1. *For* $i = 1,2,\ldots,k$, $\hat{x}_i = 0$ *iff* $n_i = n$.

PROOF. Let n be the number of steps performed by C, p_i be the number of +1's, r_i be the number of 0's, and q_i be the number of -1's, added to the i-th counter, $1 \le i \le k$, during these n steps. Hence $p_i+q_i+r_i = n$ for all i, $1 \le i \le k$. By definition we have $n_i = 2p_i + r_i$. Suppose $n_i = n$. Then it follows that $p_i = q_i$ and therefore $p_i - q_i = \hat{x}_i = 0$. Conversely, let $\hat{x}_i = p_i - q_i = 0$. Then $p_i = q_i$ and $n_i = p_i + q_i + r_i = n$. □

Hence we obtain:

COROLLARY. *A one-way* k-CM C *can be simulated in real-time by a* (k+2)*-head one-way non-writing finite automaton* F *of which the heads can detect coincidence. Hence, four heads without head reversals suffice to accept all recursively enumerable sets.*

(Hint: 1 head reads the input from left to right, 1 head keeps the count of n by its distance to the origin, and the remaining k heads so keep the counts n_1 through n_k. It was shown in [4] that 2-CMs can accept all recursively enumerable sets. We assume that the tape is unbounded, whatever the input may be.)

After this digression we show:

THEOREM 2. *If* C *is a* k*-counter machine, then we can find an oblivious* 1*-tape Turing machine* M *that simulates* C *on-line in time* $O(n)$ *and storage* $O(\log n)$ *for* n *steps by* C.

Following [7], we note that in the above theorem "machine" can be replaced by "transducer" and the proof below will still hold.

PROOF. It shall follow from the method used, and is also more generally the case for simulation by oblivious Turing machines (cf. above), that if the theorem holds for 1-CM's then it also holds for k-CM's, $k \geq 1$. Let C be a 1-CM. The simulating oblivious 1-TM M will have one storage tape divided into 3 cannels, called the *n-channel*, the *y-channel*, and the *z-channel*. If, in the current step of C its count c is modified to $c+\delta$, $\delta \in \{-1,0,+1\}$, then:

$$\begin{aligned}
\delta = +1 &\Rightarrow n \leftarrow n+1; \quad y \leftarrow y+1; \quad z \leftarrow z,\\
\delta = \ 0 &\Rightarrow n \leftarrow n+1; \quad y \leftarrow y \ \ ; \quad z \leftarrow z,\\
\delta = -1 &\Rightarrow n \leftarrow n+1; \quad y \leftarrow y \ \ ; \quad z \leftarrow z+1,
\end{aligned}$$

where n is the count contained on the n-channel, y is the count contained on the y-channel and z is the count contained on the z-channel. Hence, always (1) $c = y-z$, and (2) $y+z \leq n$. The count n on the n-channel is recorded in the usual binary notation, with the low order digit on the start square and the high order digit on the right, see Figure 1. At the start of the cycle simulating the i-th step of C, $i = p.2^j$ and p is odd, squares 0 through j-1 on the n-channel contain 1's and square j contains a 0. So in this cycle, M's head, starting from square 0, travels right to square j and deposits a 1 there. It turns all 1's on squares 0 through j-1 into 0's during this pass. The head then returns to square 0. This maintenance of the count n completely fixes M's head movement, so M is oblivious. The representation of y and z is in a redundant binary notation. If y is denoted by $y_0y_1\cdots y_i$, y_j in square j of the y-channel, then $y_j \in \{0,1,2\}$, $0 \leq j \leq i$, and $y = \Sigma_{j=0}^{i}\, y_j 2^j$. Similarly for the count z. So the representation of y[z] over {0,1,2} is not unique. Finally, the head covers 2 squares on the tape, and shifts 1 square in 1 step of M, like a mask covering 2 tape-squares. So it has a look-ahead of 1. See Figure 1.

We now explain the operation of M. The intuitive idea behind a 2 in square j of the y[z]-channel is an, as yet unprocessed, carry from the j-th to (j+1)-th position of the binary representation of y[z]. During the left-to-right sweeps of its head, governed by the moves indicated for the updating of n, M maintains invariants (1) and (2). During the corresponding right-to-left sweeps back to the start square, M

maintains also invariant (3): if $y_j[z_j] > 0$ is the contents of square j on the y[z] channel then z_{j-1}, z_j, z_{j+1} $[y_{j-1}, y_j, y_{j+1}]$ are 0 or blank. Moreover, every square right of a blank square, on that channel, contains blanks and no square containing a 0 has a blank right neighbour in that channel. This latter condition gets rid of leading 0's.

The validity of the simulation is now ensured if we can show the following assertions to hold at the end of M's cycle to simulate the i-th step of C, $i \geq 0$.

(a) For all i, $i \geq 0$, M can always add 1 to either channel y or z in the cycle simulating step i+1 of C.

(b) M can maintain invariants (1), (2) and (3) to hold at the end of each simulation cycle.

(c) The fact that (1), (2) and (3) hold at the end of the i-th simulation cycle of M ensures that the count of C is 0 subsequent to C's i-th step iff both the y-channel and z-channel contain blanks on all squares subsequent to the completion by M of simulating C's i-th step.

CLAIM 1. Assertion (a) holds at the start of each simulation cycle.

PROOF SKETCH. In the process of simulating the i-th step of C, M takes care of (a) during its left-to-right sweeps by propagating all unprocessed carries on squares 0,1,...,j on both the y-channel and z-channel to the right, leaving 0's or 1's on squares 0,1,...,j and depositing a digit d, $0 \leq d \leq 2$, on square j+1 of the channel concerned, for $i = p.2^j$ and p is odd. Assuming that M has adopted this strategy, we prove the claim by induction on the number of steps of C, equivalently, number of simulation cycles of M. □□

CLAIM 2. Assertion (b) holds at the start of each simulation cycle.

PROOF SKETCH. As we saw in the proof of claim 1, assertion (a) is implemented during the left-to-right sweeps. During the right-to-left sweeps assertion (b) is implemented.

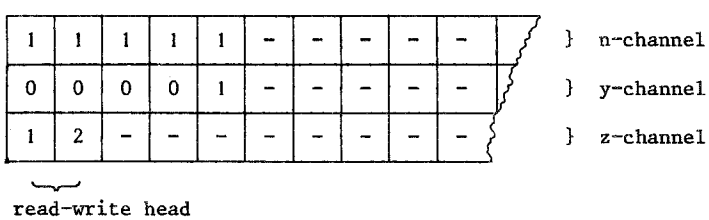

Figure 1. The configuration on M's tape after it has simulated 31 steps of C, consisting of, consecutively, 16 "add 1"'s, 11 "add 0"'s, and 5 "add -1"'s. The head has returned to the start position.

Clearly, assertion (b) holds at the start of the 1-th cycle. During its right-to-left sweeps, at each step M subtracts the 2-digit numbers covered on the y- and z-channel from each other, leaving the covered positions on at least one channel containing only 0's. M also changes (by marking the most significant digits) leading 0's on either channel into blanks during its right-to-left sweeps. Suppose the claim holds at the start of simulation cycles 1,2,...,i. We show that it then also holds at the start of simulation cycle i+1. It is obvious that M's strategy outlined above maintains invariants (1) and (2). It is left to show that it also maintains invariant (3). Again this is done by induction on the number of simulation cycles of M. □□

CLAIM 3. Assertion (c) holds at the start of each simulation cycle.

PROOF OF CLAIM. That a square on a channel can only contain a blank if all squares right of it, on that channel, contain blanks, and that the representations of y and z have no leading 0's, at the start of each simulation cycle, is a consequence of the proof of claim 2. That y-z = c at the conclusion of the i-th simulation cycle of M, where c is the count of C after i steps, follows because in the left-to-right sweep we add the correct amount to a channel according to claim 1, and in the right-to-left sweep we subtract equal amounts from either channel. It remains to show that as a consequence of the maintainence of condition (3) assertion (c) holds under these conditions.

Suppose that, at the end of the i-th simulation cycle of M, not both the y- and z-channel contain but blanks and that, by way of contradiction, y-z = 0. Then there is one channel, say y, which has a leading digit in position j, j > 0, while the digits on the positions j and j-1 on the z-channel are blank. So the count represented by y is greater or equal to 2^j while the count on z is smaller or equal to $2\sum_{j=0}^{j-2} 2^i = 2^j-2$. So y-z ≥ 2 which contradicts the assumption. (For j = 0, y-z ≥ 1.)

It remains to show that if c ≠ 0 then not both channels y and z contain only blanks. Since always, at the start of a cycle, c = y-z holds, if c ≠ 0 then y ≠ z; so in that case at least one of the y-channel and z-channel must contain a count ≠ 0. Hence there must be a square which contains a digit d > 0 on one of these channels. □□

By claims 1, 2 and 3 the on-line simulation of C by M is correct as outlined. It is easy to see that the simulation uses $O(\log n)$ storage for simulating n steps by C. We now estimate the time required for simulating n steps by C. In the i-th simulation cycle M needs to travel to square j, for $i = p.2^j$ and p is odd. Therefore, M needs 2j steps for this cycle. For $i = p.2^j$ and p is even, i.e., i is even, M needs 1 step. Hence, for simulating 2^{h+1} steps by C, M needs all in all:

$$T(2^{h+1}) = \Sigma_{j=1}^{h} 2^{h-j}.2j + 2^h = 2^{h+1}.\Sigma_{j=1}^{h} j.2^{-j} + 2^h < 2^{h+1}.\Sigma_{j=1}^{\infty} j.2^{-j} + 2^h$$

$$\leq 2.2^{h+1} + 2^h = 5.2^h.$$

Now, given n, choose $h = \lfloor \log n \rfloor$ so that $2^h \leq n < 2^{h+1}$. Then $T(n) \leq T(2^{h+1}) \leq 5.2^h \leq 5n$.

Since the movement of M's head has nothing to do with the actual counts y and z, but only with the number of steps passed since the start of C, we observe that a k-CM can be simulated on-line by an oblivious 1-tape TM M_k, which is just like M, but equipped with y_i- and z_i-channels, $1 \leq i \leq k$, and therefore with a total of 2k+1 channels. Just like M, M_k uses $\Theta(\log n)$ storage and $T(n) \leq 5n$ steps to simulate n steps of C_k, the simulated k-CM, which proves the Theorem.

The covering of 2 or 3 tape squares by the head of M can be simulated easily by cutting out 1 or 2 squares of the storage tape and buffering it in the finite control. The swapping to and fro, from tape to buffer, according to the storage head movement, is easily handled in the finite control, of which the size is blown up a bit. This is similar to the way to achieve the speed-up in [3]. □

It is well-known that oblivious Turing machine computations correspond to those of *combinational logic networks* [7,9]. The networks we consider are acyclic interconnections of *gates* by means of *wires* that carry signals. It will be assumed that there are finitely many different types of gates available and that these form a "universal" basis, so that any input-output function can be implemented by a suitable network. Each type of a gate has a *cost*, which is a positive real number, say 1 for each. The *cost* of a network is the sum of the costs of its gates. The method used above can be used to construct a combinational logic network that implements the first n steps of the computation by a k-CM. Such a network will have n inputs carrying suitable encodings of the symbols read from the input terminal and n outputs carrying encodings of the symbols written on the output terminal, where we assume, for technical reasons, that the k-CM is a transducer. If the input- and output-alphabets have more than two symbols, the inputs and outputs of the network will be "cables" of wires carrying binary signals. Using standard techniques, [7,9], it is easy to show, by imitation of the oblivious Turing machine constructed in the proof of Theorem 2, that:

COROLLARY. *If* C *is a* k-CM *transducer, then we can construct a combinational logic network implementing* n *steps of* C *with cost* $O(kn)$.

3. REAL-TIME SIMULATION BY AN OBLIVIOUS $\log^*n$-HEAD TAPE UNIT AND A CORRESPONDING COMBINATIONAL LOGIC NETWORK

In the simulations of the previous section we may incur a time delay of $\Theta(\log n)$ between the processing of an input and the production of the corresponding output. For the combinational logic network with n input ports and n output ports this is interpreted as follows. The (i+1)-th input port is enabled by a signal of the i-th output port. Between this enabling and the production of the (i+1)-th output $\Theta(\log n)$ time may pass. Note that we can only process the (i+1)-th input after the i-th output is produced, since the set of zero counts at step i influences the translation of the j-th input to incrementing/decrementing the various counters for $j > i$. To eliminate the unbounded time delay we construct as an intermediate step, for each n, a real-time simulation by an oblivious $\log^*n$-head tape unit. While this doesn't solve the problem of simulating an arbitrary multicounter machine in real-time by a Turing machine with a fixed number of tapes [1,2], it turns out that with respect to the resulting combinational logic network this gives as good a result as could be expected from simulating an arbitrary multicounter machine in real-time by an oblivious Turing machine with a fixed number of tapes. In the sequel we call a combinational network with $\Theta(1)$ time delay, between enabling the i-th input port and the production of the i-th output, a *constant data rate* network.

For the $\log^*n$-head simulation we use basically that of the previous section with the tape divided into $\log^*n$ blocks of increasing sizes, each with a resident head. The size of the 0-th block is $x = s(0)$ for some constant x, of block 1, $s(1) = 2^{x-1}$ and of block i, $i > 1$, $s(i) = 2^{s(i-1)}$. Since we need $\Theta(\log n)$ length tape to simulate n steps, we need less than $\log^* n$ blocks, where $\log^*n$ is the number of consecutive iterations of taking the logarithm to get a number less or equal to 1 when we start from n. The 0-th block is maintained in the finite control and, assuming the blocks are marked, all heads can travel around on local information alone. Only the head on block 1 needs to be connected with the finite control to exchange information regarding the counts. See Figure 2.

Each head covers four squares, like a window, and is said to be scanning the leftmost square it covers. Each head, on information which is put in the first square of its block by the head on the previous block, makes a sweep from left-to-right over its block until it scans the end cell and then back from right-to-left until it scans the first cell. There it waits until the next sweep is due. Hence such a complete sweep over block i by the resident head takes 2s(i) steps. We maintain three invariants.

At all times $t > 0$ holds:

(1) $y+z \leq t$

(2) $y-z$ = current count

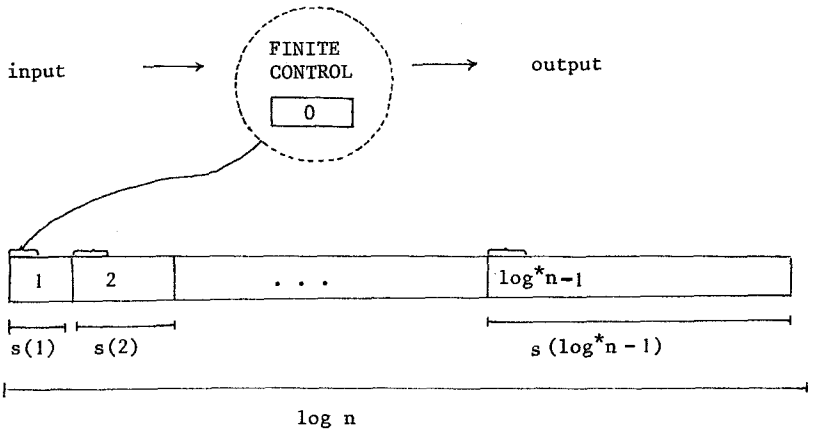

Figure 2.

(3) for all positions j on blocks 0 through $\log^* n$:

$$y_j > 0 \Rightarrow z_{j-1}, z_j, z_{j+1} \in \{0,-\} \;\&$$
$$z_j > 0 \Rightarrow y_{j-1}, y_j, y_{j+1} \in \{0,-\} \;\&$$
$$(y_j = - \iff z_j = -) \;\&\; \neg(y_j = z_j = 0 \;\&\; y_{j+1} = z_{j+1} = -).$$

(For j = 0 the obvious allowances are made.) The movements of the heads are governed by the count on the n-channel. Here this count may contain 2's representing unprocessed carries. This does not occur on the segment of n maintained on block 0, which is incremented by 1 in each step. When that count reaches 0 again (modulo 2^x steps) a carry is sent to the head on block 1 which then resides on the first square. Upon receiving a carry from block 0, the head on block 1 makes a full sweep over block 1 processing the carry and returning to the first square. Since this takes $2 \cdot s(1) = 2^x$ steps, it is in position to receive the next carry. When the segment of the n count on block 1 reaches 0 again (modulo $2^{s(1)}$ sweeps), at the right extreme of this last sweep a carry is propagated to the first square of block 2, starting a sweep of the resident head. In general, each cycle of $2^{s(i)}$ sweeps over block i produces a carry to the first square of block i+1 starting a sweep by the resident head. Since this sweep takes $2 \cdot s(i+1)$ steps, and a carry is produced each cycle of $T(i) \geq 2 \cdot s(i) \cdot 2^{s(i)}$ steps, the head on block i+1 is in position to start its sweep upon receiving the carry if

(*) $$2 \cdot s(i+1) \leq 2 \cdot s(i) \cdot 2^{s(i)}, \quad \text{for } i \geq 1.$$

Block 0 is instantly updated, and therefore we need $2\ s(1) \leq 2^{s(0)}$. Since the

inequalities are satisfied by the chosen block sizes, each propagated carry to a block is processed immediately. Having fixed the oblivious head movements, by starting a sweep over block i+1 each time a carry arrives from block i on the n channel, it remains to prove that invariants (1) - (3) can be maintained at all times during the real-time simulation. (Before proceeding, we remark that it is not necessary to assume that the blocks are delimited on the tape initially. Using four extra counters we can, as soon as we have the size of block i on one of them, determine s(i+1) before the first sweep over block i+1 is due. Determining the size of block 1 by the finite control, we can *bootstrap* the simulation of these four counters in the main simulation itself, which will be able to simulate an arbitrary number of counters, and so successively determine the blocks as they are needed. However, for the present objective of eventually producing a combinational logic network, there is no advantage in amplifying on this construction.)

We have to show:

(a) Each block can always receive incoming carries on the first square of its y-[z-] channel, and, in particular, block 0 receiving the inputs never overflows. I.e., (1) and (2) are maintained at all times.

(b) Invariant (3) holds at all times.

From (a) and (b) it follows, by the same reasoning as in the last section, that the current count y-z = 0 iff both y = z = 0 iff both y- and z-channel currently contain blanks only. The finite control, containing block 0, therefore knows instantly when the count is zero.

CLAIM 1. (a) can be maintained.

PROOF SKETCH. By induction on the consecutive blocks i.

Base case. A sweep over block 1 takes $2\ s(1) = 2^{s(0)}$ steps. Since a channel y, z on block 0 can accomodate a count of $2\cdot(2^{s(0)}-1)$, subsequent to propagation of a carry to block 1 (signifying a count of $2^{s(0)}$) block 0 contains at most $2^{s(0)} - 1$ on either channel. In the next $2^{s(0)} - 1$ steps the count may rise to $2\cdot(2^{s(0)}-1)$, but at the $2^{s(0)}$-th step a new carry is propagated to block 1, resulting from the current count on the channel plus the current input to that channel, restoring a count of at most $2^{s(0)} - 1$.

Induction. During its left-to-right sweeps, the head on block i, $i > 0$, processes a 2 deposited in the first square of the y,z-channels by propagating it as far as possible on the left two squares covered. So a 2 in the first square of a channel of block i may increment the contents of the first square of that channel on block i+1 by 1. Assume that the first square of a channel on block j, $1 \leq j \leq i$, is not incremented by more than 1 in between the starts of two consecutive sweeps over that block. Identifying 0's and blanks, and considering only one channel, let block i contain 00...0 or 10...0 at the start of the t_1-th sweep. By assumption, if block i contains

211...1 at the start of the t_2-th sweep, then $t_2 - t_1 \geq 2^{s(i)} - 1$. So sweep t_2 causes an increment of 1 on the first square of block i+1, by propagating the 2 right leaving 0's. Also by assumption, at the start of the $(t_2 - t_1 + 1)$-th sweep block i contains 00...0 or 10...0 again. Since block i contains only blanks initially, and $t_2 - t_1 + 1 \geq 2^{s(i)}$, while a sweep over block i+1 takes less time than $2^{s(i)}$ sweeps over block i, the assumption holds for block i+1. The assumption holds for block 1 by the base case.

So no channel on a block i, i > 0, ever contains more than $2^{s(i)} + 1$ which, together with the base case, proves the claim. □□

CLAIM 2. (b) can be maintained.

PROOF SKETCH. Contrary to the simulation in the previous section, we preserve invariant (3) while going from left-to-right on a block in propagating a carry. Going from right-to-left nothing is changed, so invariant (3) will hold at all times. We do so by subtracting the 3 bit pieces of the y- and z-count, covered by the left three positions of the head while going from left to right. If a nonzero digit replaces a 0 or a blank on a channel this is in the middle position of the three positions covered and the three positions covered on the other channel are replaced by 0's (or blanks). This still allows us to propagate a 2 as far as the central position of the 3 covered, so to the first square on the next block at the right extreme of the sweep. From the proof of the previous claim we have seen that a carry to the first square of the next block was sufficient. The rightmost (fourth) square covered by the head serves to detect adjacent blanks so as to return created leading 0's to blanks immediately. Due to the fact that invariant (3) holds and 2's occur only on the first square of a block and underneath a head, only one new leading 0 can be created per channel in a sweep on the rightmost nonblank block. □□

Hence we have:

THEOREM 3. *We can simulate the first* n *steps of a multicounter machine by an oblivious* $\log^*$n*-head tape unit in real-time and logarithmic space. (Similarly we can directly construct an oblivious* $\log^*$n*-tape Turing machine for the same job.)*

Just as argued in the previous section, we can construct a corresponding combinational logic network. Since only squares which are being rewritten need to be represented by logic components, and the time to make a sweep on block i+1 is $2 \cdot s(i+1)$ while there is only one such sweep in each cycle T(i), $T(i) \geq 2 \cdot s(i) \cdot 2^{s(i)} = 2 \cdot s(i) \cdot s(i+1)$ steps, the cost of this network is reduced from the expected $O(n \log^* n)$ by not representing squares covered by a head which does no rewriting.

THEOREM 4. *We can implement the first* n *steps of a* k*-counter machine on an* $O(kn)$ *cost combinational logic network with constant data rate.*

PROOF. The network has a constant data rate, i.e. a time interval $O(1)$ between enabling the i-th input port by the (i-1)-th output and producing the i-th output, $1 \le i \le n$, since it is derived from a real-time simulation. Each piece of logic circuitry, representing four squares covered by a head which is moving, has cost c(k), depending only on the number k of counters simulated but not on the number of steps n. The state of the finite control (containing block 0) is represented by cost d(k) pieces of logic connected to the input ports. In each cycle $T(i) \ge 2\ s(i) \cdot 2^{s(i)}$ steps, the head on block i+1 is active for only $2 \cdot 2^{s(i)}$ steps. Hence such a head is active for only $O(n/s(i))$ steps out of n, $1 \le i < \log^* n$. Summing this for all blocks i, $1 \le i \le \log^* n$, and adding the cost for the blocks 0 connected to the input ports we obtain a total cost C(k,n):

$$C(k,n) = \left(\left(\sum_{i=1}^{\log^* n - 1} n \cdot c(k)/s(i)\right) + n(c(k) + d(k))\right)$$

$$= O(n \cdot k). \qquad \square$$

4. SIMULATION BY CYCLIC NETWORKS (AND VLSI)

When we are not restricted to acyclic logic networks, but are allowed cyclic logic networks, or work in the framework of the VLSI model of computation recently advanced in [5], it is not difficult to see that:

THEOREM 5. *If* C *is a* k-CM *transducer, then we can construct*
(i) *a cyclic logic network simulating* n *steps of* C *with cost* $O(k \log n)$ *in real-time;*
(ii) *a* VLSI *simulating* n *steps of* C *in real-time with area* $O(k \log n)$.

PROOF. We prove (ii), and (ii) clearly implies (i). The VLSI circuit realizing the claimed behaviour could look as follows:

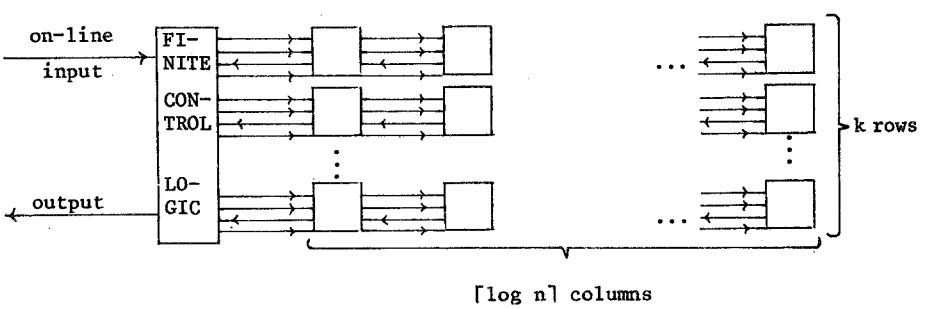

Figure 3. VLSI circuit simulating k-CM.

Each row stores a count in ordinary binary notation, with the low digit contained in the left block. Each block stores two bits: one for the binary digit of the count, and one to indicate whether the count digit contained is the most significant bit of that count. Carries are propagated along the top wire of each row, borrows along the bottom wire. The middle wires of each row transport information concerning the most significant bit in that row. Each block contains the necessary logic to process and transmit correctly carries, borrows and information concerning the most significant bit. The finite-control-logic rectangle processes the input signals and the information from the first blocks of each row, whether they contain a most significant bit 0 of the corresponding count, to issue carries or borrows to the first block of each row and to compute the output signal. We leave it to the reader to confirm that, subsequent to receiving the input signal, the corresponding output signal can be computed in time $O(\log k)$, which corresponds to the bit length of an input signal for driving k counters. Hence the VLSI circuit simulates the k-CM in real-time. Since the area occupied by the wires emanating from each block can be kept to the same size as the area occupied by the block itself, the blocks take $O(k \log n)$ area. The finite control logic structure contains some trees of depth log k, so its area can be kept to $O(k \log k)$. Under the assumption that $k \in O(n)$ this yields the required result. □

To fit a long thin rectangle in a square, as often is necessary to implement the structure on chip, we can fold it without increasing the surface area significantly. Note that the structure contains no long wires, and that it does not have to be overall synchronized: local synchronization is all we need. Hence it is a practicable design.

5. SIMULATION BY RAMs

For simulation with a uniform cost RAM it is clear that we can simulate a multi-counter on-line with constant delay and constant storage. Constant delay is the RAM analogue for real-time, i.e. if T(n) is the time for simulating n steps by the multi-counter then the RAM simulates on-line with constant delay if $T(n+1) - T(n) < c$ for some constant c and all n. It is easy to see, that a logarithmic cost RAM cannot simulate a counter machine on-line with constant delay, since it can only address registers of bounded index and bounded contents.

At first glance it seems that we can do no better than $O(n \log n)$ time for simulation of a countermachine by a logarithmic cost RAM. If we simulate with a tally mark in each register, we have to use indirect addressing to maintain the top of the counter requiring $O(n \log n)$ time and $O(n)$ storage to simulate n steps. Using a binary count we need only k registers for a k-counter machine, but need again $O(n \log n)$ time and $O(\log n)$ storage. Define an *oblivious RAM* as one in which the sequence of executed instructions, as well as the sequence of accessed storage locations, is a function of time alone. Due to the usual restrictions of the arithmetic operations of RAMs to +

and -, as well as to the needed translation of input commands with respect to the set of currently zero counters into counter instructions, we need to augment the RAM with some constant bit length boolean/arithmetic instructions in order not to be artificially precluded from obtaining the following result by imitation of the simulation in Section 2. (If we do not add these extra operations the Theorem below might only hold for nonoblivious RAMs by purely irrelevant definitional reasons.) Since we view the RAM as an abstract storage device performing a transduction we also assume it is connected to the input and an output terminal and dispense with the usual 'accept' instruction. Using the simulation in Section 2 we obtain:

THEOREM 6. *We can simulate a* k*-counter machine on-line by an oblivious logarithmic cost RAM in* $O(k \cdot n)$ *time and* $O(k \log n)$ *storage.*

PROOF. Do the simulation of Section 2 with the RAM, storing the head position of the 1-tape Turing machine in register 1 and the j-th square contents in register j+1. Then the sequence of executed instructions in the RAM program, and the sequence of accessed registers can be made a function of time alone. So the RAM is oblivious. The time for simulating sweeps of length j on the RAM is $O(k \sum_{i=2}^{j+1} \log i) = O(kj \log j)$. So if $T(2^{h+1})$ is the time needed to execute the first 2^{h+1} steps of the multicounter we obtain:

$$T(2^{h+1}) \in O\left(\sum_{j=1}^{h} k \cdot 2^{h-j} \cdot j \log j + k \cdot 2^{h}\right)$$

$$= O(k \cdot 2^{h+1}).$$

So $T(n) \in O(kn)$ and the storage used is $O(k \log n)$. □

This simulation is optimal in both space and time, even for nonoblivious RAMs.

6. FINAL REMARKS

Comparing our solution of the linear time simulation of a k-CM with the nonoblivious one in [2], the reader will notice that our average time complexity is the same as the worst case time complexity in [2]. So in actual fact, the solution in [2] runs faster in most cases than the one presented here. In [1] it was shown that the Origin Crossing Problem: "report when all k counts simultaneously reach 0" admits a real-time one-tape Turing machine solution. Contrary to the linear time simulation of [2], the method in [1] seems to contain inherently nonoblivious features, preventing us from turning it into an oblivious version. It has been a classic question [1,2], whether or not the Axis Crossing Problem: "report when one out of k counters reaches 0" or more generally "on-line simulate a k-counter machine" can be done in real-time by a (nonoblivious) k'-tape Turing machine for $k' < k$. A reasonable approach may seem to show that, anyway, a real-time simulation of multicounter machines by *oblivious*

one-head tape units is impossible. In the event, intuition is wrong. We have noticed, cf. Section 2, that if we restrict the simulating device to its oblivious counterpart we have the advantage that if 1 counter is simulatable then k counters can be simulated in just the same way. This key observation has led us in the meantime, by augmenting the ideas presented here with an involved tape manipulation technique, to a real-time simulation of multicounter machines by *oblivious* one-head tape units, thus solving the above problem with a considerable margin [11]. Although superficially it would seem that this farther reaching result obviates the present ones we like to point out that:

- The present results are far simpler to derive and will suffice for many applications, as will some of the distinctive techniques.
- To derive the linear cost constant datarate combinational logic network the present route by way of a $\log^{*}$n-head tape unit suffices.
- The RAM simulation result seems difficult to derive, if at all, from the simulation in [11] without regressing to the simulation given here.

REFERENCES

[1] FISCHER, M.J. & A.L. ROSENBERG, *Real-time solutions of the origin-crossing problem*, Math. Systems Theory 2 (1968), 257-264.

[2] FISCHER, P.C., A.R. MEYER & A.L. ROSENBERG, *Counter machines and counter languages*, Math. Systems Theory 2 (1968), 265-283.

[3] HARTMANIS, J. & R.E. STEARNS, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc. 117 (1965), 285-306.

[4] MINSKY, M., *Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines*, Ann. of Math. 74 (1961), 437-455.

[5] MEAD, C.A. & L.A. CONWAY, *Introduction to VLSI Systems*, Addison-Wesley, New York, 1980.

[6] PATERSON, M.S., M.J. FISCHER & A.R. MEYER, *An improved overlap argument for on-line multiplication*, SIAM-AMS Proceedings, Vol. 7, (Complexity of Computation) 1974, 97-112.

[7] PIPPENGER, N. & M.J. FISCHER, *Relations among complexity measures*, Journal ACM, 26 (1979), 361-384.

[8] ROSENBERG, A.L., *Real-time definable languages*, Journal ACM 14 (1967), 645-662.

[9] SCHNORR, C.P., *The network complexity and Turing machine complexity of finite functions*, Acta Informatica 7, (1976), 95-107

[10] VITÁNYI, P.M.B., *Relativized Obliviousness*, in *Lecture Notes in Computer Science* 88 (1980), 665-672, Springer Verlag, New York. (Proc. MFCS '80).

[11] VITÁNYI, P.M.B., *Real-time simulation of multicounters by oblivious one-tape Turing machines*, Proceedings 14th ACM Symp. on Theory of Computing, 1982.