Speedups of Deterministic Machines by Synchronous Parallel Machines*

PATRICK W. DYMOND

Department of Electrical Engineering and Computer Sciences,
Mail Code C-014, University of California at San Diego,
La Jolla, California 92093

AND

MARTIN TOMPA

Department of Computer Science, FR-35,
University of Washington, Seattle, Washington 98195

Received May 1983; revised January 1984; accepted July 16, 1984

This paper presents the new speedups $\text{DTIME}(T) \subseteq \text{ATIME}(T/\log T)$ and $\text{DTIME}(T) \subseteq \text{PRAM-time}(\sqrt{T})$. These improve the results of Hopcroft, Paul, and Valiant (J. Assoc. Comput. Mach. 24 (1977), 332–337) that $\text{DTIME}(T) \subseteq \text{DSPACE}(T/\log T)$, and of Paul and Reichuk (Acta Inform. 14 (1980), 391–403) that $\text{DTIME}(T) \subseteq \text{ATIME}((T \log \log T)/\log T)$. The new approach unifies not only these two previous results, but also the result of Paterson and Valiant (Theoret. Comput. Sci. 2 (1976), 397–400) that $\text{Size}(T) \subseteq \text{Depth}(O(T/\log T))$.

1. SYNCHRONOUS PARALLEL MACHINES

This work is concerned with the amount of time that can be saved by using synchronous parallel machines in place of sequential ones. Cook [3] has classified the synchronous parallel models according to whether the interconnection among processors during a computation is fixed or variable. The fixed structure models include uniform Boolean circuits [1, 19], aggregates [5], conglomerates [7], and alternating Turing machines [2]. The variable structure models include PRAMs [6], SIMDAGs [7], and hardware modification machines [5]. Time complexities on these models differ at most by a quadratic:

$$\text{Fixed-time}(T) \subseteq \text{DSPACE}(T) \subseteq \text{Variable-time}(T) \subseteq \text{Fixed-time}(T^2),$$

where $\text{Fixed-time}(T)$ can represent the class of languages accepted in time $T$ by any

* This material is based upon work supported by the National Science Foundation under Grants MCS-8111098 and MCS-8110089.
one of the fixed structure models cited, and similarly Variable-time(T) the class of languages accepted in time T by any one of the variable structure models cited.

The best general speedup known of multitape deterministic Turing machines by any of the fixed structure machines is due to Paul and Reischuk [13], namely DTIME(T) \subseteq ATIME((T \log \log T)/\log T). The best speedup by variable structure machines is hardly better: from the result of Hopcroft, Paul, and Valiant [9] that DTIME(T) \subseteq DSPACE(T/\log T) it follows that DTIME(T) \subseteq Variable-time(T/\log T). (Better speedups are possible if the simulated machine is restricted to have only one tape: Paul, Prauβ, and Reischuk [12] have shown that time T on such a machine can be simulated in time O(\sqrt{T}) on an alternating Turing machine, based on an earlier result of Paterson [10].)

This paper presents two new speedups of deterministic Turing machines by synchronous parallel machines. The first is a speedup by fixed structure machines, namely DTIME(T) \subseteq ATIME(T/\log T). This improves not only the speedup of Paul and Reischuk, but also the aforementioned result of Hopcroft, Paul, and Valiant that DTIME(T) \subseteq DSPACE(T/\log T). Our method is similar to those used in the two results it subsumes, but hinges on a new 2-person pebble game that models alternating computations. As a consequence of studying this new game, we also get an alternative proof of the result of Paterson and Valiant [11] that Size(T) \subseteq Depth(O(T/\log T)) for (nonuniform) Boolean circuits.

Our second speedup is by variable structure machines, namely DTIME(T) \subseteq PRAM-time(\sqrt{T}). (The same speedup holds for SIMDAGs, which are at least as fast as PRAMs.) This improved speedup reflects the presumed quadratic advantage of variable structure machines over fixed structure machines.

2. A Pebble Game that Models Alternating Computations

This section presents the rule of a new 2-person pebble game, which was investigated initially in [21]. The main result of this section concerns an optimal strategy for the game, and is applied in the next section to prove the promised speedup of deterministic machines by alternating machines.

The ordinary pebble game is played by 1 person (see Pippenger [16] for a survey). The 2-person pebble game used in this paper is a game played by 2 adversaries, called Pebbler and Challenger. Like the 1-person version, this game is played on an acyclic directed graph G. At all times during the game there is one vertex designated by Challenger and called the "challenged" vertex. Challenger moves first by choosing any vertex to challenge. Pebbler's turn consists of placing 1 pebble on each of any number of vertices, with no restriction on which vertices may be pebbled. Challenger's turn consists of choosing a new vertex to challenge which, from this point on, must be either the current challenged vertex or any vertex pebbled in Pebbler's most recent move. The players alternate in this fashion until, at the beginning of Pebbler's move, all immediate predecessors of the current challenged vertex
w are already pebbled.\footnote{A vertex \( u \) is an \textit{immediate predecessor} of a vertex \( v \) if \((u, v)\) is an edge. The \textit{predecessor} relation is the transitive (but irreflexive) closure of the immediate predecessor relation.} The game ends at this point, and we say that Challenger loses in \( G \) at \( w \). (For the purposes of this paper, it suffices to make the simplifying assumption that a pebble placed on a vertex remains there until Challenger loses.)

If \( G \) is thought of as a circuit computing some function, then a play of this 2-person game corresponds to an alternating implementation of that circuit, in the following sense. A pebble placed on vertex \( v \) by Pebbler corresponds to existentially guessing the value of the subexpression computed at \( v \). A move of Challenger corresponds to universally verifying each of those guesses, plus the fact that those guesses lead to the correct value computed at the current challenged vertex.

If, even against best defense by Challenger, Pebbler can always win the game using at most \( t \) pebble placements, then we say that \( G \) can be \textit{2-person pebbled in time} \( t \).

The main lemma used by Hopcroft, Paul, and Valiant \cite{9} to prove that \( \text{DTIME}(T) \subseteq \text{DSPACE}(T/\log T) \) is that any graph with \( n \) vertices and bounded indegree can be (1-person) pebbled using \( O(n/\log n) \) pebbles. The main result of this section is the analogue for time in the 2-person game:

\textbf{Lemma 1.} \textit{Let} \( G = (V, E) \) \textit{be an acyclic directed graph with} \( n \) \textit{vertices and bounded indegree. Then} \( G \) \textit{can be 2-person pebbled in time} \( O(n/\log n) \).

\textit{Proof.} Pebbler's strategy and its analysis are the alternating analogues of the "best pebble" strategy of Paul, Tarjan, and Celoni \cite{14, 15}. Let \( d \) be the maximum indegree of any vertex in \( G \), and \( m \) be the number of edges in \( G \). Suppose Challenger's first move is to challenge vertex \( u \). Pebbler's strategy is described below as a recursive procedure that, given \( G \) and the challenged vertex \( u \) as inputs, pebbles \( G \) and returns the vertex in \( G \) at which Challenger lost.

(I) \text{If} \( m \leq k \), \text{where} \( k \) \text{is the constant specified in Theorem 5 of [14]}, \text{then Pebbler's first (and only) move is to place a pebble on every vertex other than} \( u \) \text{in the weakly connected component of} \( G \) \text{that contains} \( u \). \text{Challenger loses in} \( G \) \text{at the next vertex challenged.}

(II) \text{If} \( m > k \), \text{partition} \( V \) \text{into blocks} \( V_1 \) \text{and} \( V_2 \) \text{such that}

(i) \text{there are no edges from any vertex in} \( V_2 \) \text{to any vertex in} \( V_1 \), \text{and}

(ii) \text{the total indegree of all vertices in} \( V_2 \) \text{satisfies}

\[
m/2 + m/\log m - d \leq |(V \times V_2) \cap E| \leq m/2 + m/\log m.
\]

\text{Let} \( G_1 \) \text{and} \( G_2 \) \text{be the subgraphs induced by} \( V_1 \) \text{and} \( V_2 \), \text{respectively.}

(A) \text{If} \( v \in V_1 \), \text{Pebbler applies the strategy recursively to the subgraph} \( G_1 \) \text{and challenged vertex} \( v \). \text{Suppose Challenger loses in} \( G_1 \) \text{at} \( w \). \text{Then Challenger also loses in} \( G \) \text{at} \( w \), \text{since all of} \( w \)'s immediate predecessors are in \( G_1 \).
(B) Otherwise assume \( v \in V_2 \). Let \( C \) be the subset of \( V_1 \) with immediate successors in \( V_2 \), that is,

\[
C = \{ u \in V_1 \mid \text{for some } w \in V_2, (u, w) \in E \}.
\]

(1) If \(|C| \leq 2m/\log_2 m\), Pebbler's first move is to pebble each vertex in \( C \).

(a) If Challenger next challenges a vertex \( u \) in \( C \), Pebbler applies the strategy recursively to \( G_1 \) with challenged vertex \( u \). If Challenger loses in \( G_1 \) at \( w \), Challenger also loses in \( G \) at \( w \).

(b) If Challenger rechallenges \( v \), Pebbler applies the strategy recursively to \( G_2 \) with challenged vertex \( v \). Suppose Challenger loses in \( G_2 \) at vertex \( w \). Then Challenger also loses in \( G \) at \( w \), since every immediate predecessor of \( w \) in \( G_1 \) is in \( C \), and is hence already pebbled.

(2) If \(|C| > 2m/\log_2 m\), Pebbler applies the strategy recursively to \( G_2 \) with challenged vertex \( v \), ignoring all edges from \( G_1 \) to \( G_2 \). Suppose Challenger loses in \( G_2 \) at vertex \( w \). At the next move Pebbler pebbles those vertices in \( V_1 \) that are immediate predecessors of \( w \).

(a) If Challenger persists in challenging \( w \), Challenger immediately loses in \( G \) at \( w \).

(b) If Challenger challenges a vertex \( u \) in \( V_1 \), Pebbler applies the strategy recursively to \( G_1 \) with challenged vertex \( u \). If Challenger loses in \( G_1 \) at \( w \), Challenger also loses in \( G \) at \( w \).

Let \( q(m) \) be the maximum number of pebble placements used by this strategy on any graph with \( m \) edges and indegree at most \( d \). Then

\[
q(m) \leq m, \quad \text{if } m \leq k.
\]

\[
\leq \max \{ q(m/2 + m/\log_2 m) + 2m/\log_2 m, \quad \text{if } m > k.
\]

\[
2q(m/2 - m/\log_2 m + d) + d \},
\]

(The constant \( k \) satisfies the property that

\[
m/2 - m/\log_2 m + d \leq m/2 + m/\log_2 m, \quad \text{if } m > k \ [14].
\]

This recurrence is identical to the one that Paul, Tarjan, and Celoni solve \[15\], except for the presence of the last \(+d\) term. Minor changes to their proof by induction show that this recurrence has a solution that satisfies

\[
q(m) \leq ((d + 1) \log_2 k)m/\log_2 m - d.
\]

The stated result follows from the fact that \( m \leq dn \).
Lemma 1 is optimal to within a constant factor, since

1. there exist graphs with \( n \) vertices and indegree 2 whose (1-person) pebbling requires \( \Omega(n/\log n) \) pebbles [14], and

2. any graph that cannot be 1-person pebbled with \( p \) pebbles cannot be 2-person pebbled in time \( p - 1 \) [21].

3. THE SPEEDUP OF DETERMINISTIC MACHINES BY ALTERNATING MACHINES

This section employs Lemma 1 to prove the stated speedup of deterministic machines by alternating machines. The key ideas are adapted from Hopcroft, Paul, and Valiant [9] and Paul and Reischuk [13].

**Theorem 2.** \( \text{DTIME}(T(n)) \subseteq \text{ATIME}(T(n)/\log T(n)) \), for any \( T(n) \geq n \).

(See [2] for a discussion of sublinear time-bounded alternating Turing machines.)

**Construction**

Let \( D \) be a deterministic Turing machine with \( k \) worktapes that accepts in time \( T(n) \), and assume without loss of generality that \( D \) loops in an accepting state if it ever reaches one. Let \( x \) be an input of length \( n \). Consider the computation of \( D \) on input \( x \) to be divided into \( B \) time intervals each of length \( T(n)/B \), and the \( k \) worktapes to be divided into \( B \) blocks each of length \( T(n)/B \), the value of \( B \) to be determined later. A block \( b \) on worktape \( j \) is said to be accessible at time \( t \) if worktape head \( j \) is either in \( b \) or in a block adjacent to \( b \) at time \( t \). Thus, at most \( 3k \) blocks are accessible at any time. Associate an acyclic directed graph \( G_{D,x} = (V, E) \) with the computation as follows:

- \( V = \{0, 1, 2, \ldots, B\} \); vertex \( i \) should be thought of as associated with time \( iT(n)/B \) of the computation of \( D \) on \( x \).
- \( E = \{(i, j) \mid i < j, \text{ and some worktape block of } D \text{ is accessible at times } iT(n)/B \text{ and } jT(n)/B, \text{ but not at any time } hT(n)/B, \text{ where } h \text{ is an integer satisfying } i < h < j \} \).

\( G_{D,x} \) has \( B + 1 \) vertices and indegree at most \( k + 1 \). (The edge \((j - 1, j)\) accounts for at least \( 2k \) of the blocks accessible at time \( jT(n)/B \).)

We now describe an alternating Turing machine \( A \) that simulates \( D \). On input \( x \) of length \( n \), \( A \) guesses \( T(n) \) and \( B \), and guesses and records the positions of each of the \( k + 1 \) tape heads of \( D \) at each of the times \( 0, T(n)/B, 2T(n)/B, \ldots, T(n) \). From these \( A \) can construct and record the graph \( G_{D,x} \).

The main portion of the construction of \( A \) is a simulation of a 2-person pebbling of \( G_{D,x} \). Information about the game configuration is recorded using 3 additional
tapes, one for each of the current challenged vertex, vertices pebbled in Pebbler's
most recent turn, and vertices pebbled in earlier turns. The information associated
with each challenged and pebbled vertex on these tapes consists of the vertex num-
ber $i$, plus the (guessed) state and contents of all accessible worktape blocks at time
$iT(n)/B$. Until the game begins these tapes are blank.

Corresponding to Challenger's first move, $A$ records the vertex number $B$ on the
"challenged vertex" tape. It guesses and records on this same tape an accepting
state of $D$ and the contents of the accessible blocks of $D$ at time $T(n)$. The
simulation of the pebble game then proceeds as follows. If it is Pebbler's turn, $A$
guesses whether or not Challenger has just lost. If so, $A$ accepts if and only if

1. all immediate predecessors of the challenged vertex $j$ are pebbled, and
2. the head positions, state, and contents of the accessible blocks associated
   with vertex $j$ are consistent with the input and with the head positions, state, and
   contents of the accessible blocks associated with vertex $j-1$, and
3. the contents of those accessible blocks associated with vertex $j$ that are
   not accessible blocks associated with vertex $j-1$ are consistent with the block con-
   tents associated with vertex $j$'s other immediate predecessors.

Part (2) is verified by direct simulation of $D$ for $T(n)/B$ steps. Part (3) requires
no simulation at all, since any block that is accessible at time $(j-1)T(n)/B$ but not at
time $iT(n)/B$ could not have been altered since the last time $iT(n)/B$ that it
was accessible. (If it was never before accessible, it must be blank.)

If, on the other hand, it is Pebbler's turn but $A$ guessed that Challenger has not
yet lost, $A$ guesses how many vertices Pebbler will pebble on this turn, and guesses
and records on the "most recently pebbled" tape the vertex number, state, and con-
tents of the accessible blocks for each of these vertices. If it is Challenger's turn and
Pebbler has pebbled $p$ vertices in the most recent turn, $A$ uses universal states to try
each of these $p$ vertices plus the current challenged vertex as the new challenged
vertex. It does this by overwriting the "challenged vertex" tape with the information
 corresponding to the new challenged vertex, appending all the information from the
"most recently pebbled" tape onto the "previously pebbled" tape, and erasing the
"most recently pebbled" tape.

Correctness

It is easy to see that if $D$ accepts an input $x$, $A$ does as well. For the converse, the
following stronger claim will be established:

Suppose that, at the beginning of Pebbler's turn, $A$ is in a configuration
that leads to acceptance. Then if the guessed head positions, state, and con-
tents of the accessible blocks are correct for every pebbled predecessor of
the challenged vertex, they are also correct for the challenged vertex.

(Note that the term "predecessor" in this claim does not necessarily mean
"immediate predecessor.") The correctness of $A$'s construction follows from this
claim, since if $A$ accepts $x$, the accepting state of $D$ guessed at Challenger's first turn is in fact the state $D$ is in at time $T(n)$ given input $x$. The claim itself is established by induction on the number $h$ of alternations required to end the game, starting from the hypothesized Pebbler's turn.

**Basis** ($h = 0$). Then $A$ accepts if and only if the information associated with the challenged vertex is consistent with the information associated with each of its immediate predecessors, which are all pebbled. Since, by hypothesis, the latter pieces of information are all correct, so are the former.

**Induction** ($h > 0$). Suppose it is the beginning of Pebbler's turn, $A$ is in a configuration that leads to acceptance in at most $h$ alternations, $v$ is the challenged vertex, $P$ is the set of pebbled vertices, and the information associated with each predecessor of $v$ in $P$ is correct. Then there is some set $R$ of vertices that $A$ guesses to pebble this move such that, no matter which vertex in $R \cup \{v\}$ is next challenged, $A$ will be in a configuration that leads to acceptance in at most $h - 2$ alternations. Without loss of generality, assume that every vertex in $R$ is a predecessor of $v$. By the induction hypothesis, for any $v' \in R \cup \{v\}$, if the information associated with each predecessor of $v'$ in $P \cup R$ is correct, the information associated with $v'$ is also correct. By considering the vertices of $R \cup \{v\}$ one at a time in topological order, this statement implies that the information associated with each vertex in $R \cup \{v\}$ is correct. In particular, the information associated with $v$ is correct.

**Analysis**

All that remains is to show that $A$ runs in the stated time. $O(B \log T(n))$ time suffices for the tasks done by $A$ before the pebbling, namely guessing $(k + 1)(B + 1)$ head positions of $D$, and constructing $G_{D,v}$, using a fixed number of alternations to guess and verify the edges. $O(\log n + T(n)/B)$ time suffices for the task done after the pebbling, namely simulating $D$ for $T(n)/B$ steps to verify that the information associated with the vertex at which Challenger loses is consistent with the information associated with each of its immediate predecessors. ($A$'s index tape can be used to copy onto a worktape that portion of the input within radius $T(n)/B$ of $D$'s input head, after which the worktape is used in place of the input tape in the direct simulation.)

By Lemma 1, Pebbler needs only $O(B/\log B)$ pebble placements on $G_{D,v}$, no matter how Challenger plays. For each such placement, $A$ requires time $O(\log B + T(n)/B)$ to guess the vertex number, state, and contents of the accessible blocks, and later to copy that information onto the "challenged vertex" and "previously pebbled" tapes. Finally, the time to retrieve the information required to set up the direct simulation of $T(n)/B$ steps of $D$ is proportional to the length of the "previously pebbled" tape, which certainly cannot exceed the running time of $A$ up to this point. This task is performed only once.
Therefore, the total amount of time used by $A$ is

$$O(B \log T(n) + (\log n + T(n)/B) + (B/\log B)(\log B + T(n)/B))$$

$$= O(B \log T(n) + T(n)/\log B).$$

This time bound is $O(T(n)/\log T(n))$ if $B$ is chosen to be $\sqrt{T(n)}$.

It is interesting to note that the number of alternations in this simulation is $O(B/\log B)$. Hence, any deterministic Turing machine running in time $T(n)$ can be simulated by an alternating Turing machine using $O(T(n)/\log B(n))$ time and only $O(B(n))$ alternations, for any $B(n)$. In particular, $O((T(n))')$ alternations suffice to achieve the log $T(n)$ speedup of Theorem 2, for any fixed $\varepsilon > 0$.

4. The Speedup of Size by Depth

Paterson and Valiant [11] showed that circuits of size $T$ could be simulated by circuits of depth $O(T/\log T)$. This section demonstrates that their result, like Theorem 2, is a consequence of Lemma 1.

Define $\text{Depth}(T)$ ($\text{Size}(T)$) to be the set of languages that can be recognized by a Boolean circuit with maximum path length (resp. number of gates) $T$.

**Theorem 3.** Let $G$ be a Boolean circuit that recognizes a language $L$. If $G$ can be 2-person pebbled in time $t$, then $L \in \text{Depth}(2t + 1)$.

**Construction**

The construction is motivated by Ruzzo's simulation of alternating time by circuit depth [19]. The idea behind the construction is very much like that of Theorem 2, namely, the simulating circuit uses $\lor$-gates to "guess" the values computed at pebbled gates of $G$, and $\land$-gates to "verify" those guesses.

The Boolean circuit of depth $2t + 1$ is a tree $T$, which will be described recursively. Suppose that $v$ is the challenged vertex and $P$ the set of pebbled vertices in $G$ at the beginning of a Pebbler's turn. Associated with this point in the game are subtrees $T(P, v, I)$, one for every possible interpretation $I: P \cup \{v\} \rightarrow \{0, 1\}$. If all immediate predecessors of $v$ are in $P$, then $T(P, v, I)$ is either a constant gate, an input gate, or the negation of an input gate, as follows:

$T(P, v, I) = x_i$, if $v$ is the input $x_i$, and $I(v) = 1$.

$= \lnot x_i$, if $v$ is the input $x_i$, and $I(v) = 0$.

$= 1$, if $v$ is an "$\lor$" gate with inputs $a$ and $b$, and $I(v) = I(a) \lor I(b)$.

$= 0$, if $v$ is an "$\land$" gate with inputs $a$ and $b$, and $I(v) \neq I(a) \land I(b)$.

(In this definition, "$\lor$" can represent any Boolean operator that occurs in the circuit $G$.)
Otherwise suppose Pebbler pebbles a set \( R \) of vertices in this turn. Then \( T(P, v, I) \) is a complete binary tree of \( \lor \)-gates having \( 2^{|R|} \) leaves, one for every possible extension \( I' \) of the interpretation \( I \) to domain \( P \cup R \cup \{v\} \). Each of these leaves is, in turn, the root of a complete binary tree of \( \land \)-gates having \( |R \cup \{v\}| \) leaves, one for every possible vertex \( v' \) eligible for Challenger's next challenge. The leaf so reached is the root of \( T(P \cup R, v', I') \), which is constructed recursively. Finally, \( T \) itself is simply \( T(\phi, r, I) \), where \( r \) is the output gate of \( G \) and \( I(r) = 1 \).

**Correctness**

Fix some arbitrary input \( x \). For a particular distinguished subtree \( T(P, v, I) \) of \( T \), \( I \) is said to be correct on some \( u \in P \cup \{v\} \) if and only if the gate \( u \) in \( G \) evaluates to \( I(u) \) on the fixed input \( x \). Henceforth, let \( r \) denote the output gate of \( G \). To demonstrate correctness of the construction of \( T \), it must be shown that, on input \( x \), \( r \) evaluates to 1 if and only if the root of \( T \) evaluates to 1.

"Only if" clause. For this direction, the following stronger claim will be established:

If \( I \) is correct on every vertex in \( P \cup \{v\} \), then the root of \( T(P, v, I) \) evaluates to 1.

The desired conclusion follows from this claim by considering \( T(\phi, r, I) \), where \( I(r) = 1 \), for if \( r \) evaluates to 1, then \( I \) is correct on \( r \), so the root of \( T = T(\phi, r, I) \) evaluates to 1. The claim itself is established by induction on the height \( h \) of \( T(P, v, I) \).

**Basis** \( (h \leq 1) \). If \( T(P, v, I) \) is an input \( x_i \) and \( I(v) = 1 \) is correct, then \( x_i = 1 \). If \( T(P, v, I) \) is \( \sim x_i \) and \( I(v) = 0 \) is correct, then \( \sim x_i = 1 \). Finally, if \( T(P, v, I) \) is a constant, and \( v \) is \( a \circ b \), and \( I \) is correct on \( a, b, \) and \( v \), then \( I(v) = I(a) \circ I(b) \), so the constant is 1.

**Induction** \( (h > 1) \). Consider the subtree \( T(P, v, I) \), and assume that \( I \) is correct on every vertex in \( P \cup \{v\} \). Let \( R \) be the set of vertices pebbled by Pebbler in this turn. Let \( I' \) be the extension of \( I \) to domain \( P \cup R \cup \{v\} \) that is correct on every vertex in \( R \). By the induction hypothesis, the root of \( T(P \cup R, v', I') \) evaluates to 1 for every possible vertex \( v' \in R \cup \{v\} \). Hence the root of \( T(P, v, I) \) evaluates to 1, by its construction.

"If" clause. This direction is similar to the correctness proof of Theorem 2. The following claim will be established:

If the root of \( T(P, v, I) \) evaluates to 1 and \( I \) is correct on every predecessor of \( v \) in \( P \), then \( I \) is correct on \( v \).

(Note that the term "predecessor" in this claim does not necessarily mean "immediate predecessor.") The desired conclusion follows from this claim by again considering \( T = T(\phi, r, I) \), where \( I(r) = 1 \), for if the root of \( T \) evaluates to 1, then
I(r) = 1 is correct (I being vacuously correct on every predecessor of r in \( \phi \)), and so r evaluates to 1. The claim itself is proved by induction on the height \( h \) of \( T(P, v, I) \).

**Basis** \((h \leq 1)\). If \( T(P, v, I) \) is an input \( x_i \) and \( x_i = 1 \), then \( I(v) = 1 \) is correct. If \( T(P, v, I) \) is \( \sim x_i \) and \( \sim x_i = 1 \), then \( I(v) = 0 \) is correct. Finally, if \( T(P, v, I) \) is the constant 1, and \( v = a \circ b \), and I is correct on a and b, then \( I(v) = I(a) \circ I(b) \) is also correct.

**Induction** \((h > 1)\). Suppose the root of \( T(P, v, I) \) evaluates to 1 and I is correct on every predecessor of v in P. Let \( R \) be the set of vertices pebbled by Pebbler in this turn, and assume without loss of generality that every vertex in \( R \) is a predecessor of v. By the construction of \( T(P, v, I) \), there must be some extension \( I' \) of I to domain \( P \cup R \cup \{v\} \) such that, for every \( v' \in R \cup \{v\} \), the root of \( T(P \cup R, v', I') \) evaluates to 1. Notice that any predecessor of \( v' \) in \( P \cup R \) is either in \( R \), or is a predecessor of \( v \) in \( P \); \( I' \), and hence \( I' \), is correct on the latter by hypothesis. By considering the vertices of \( R \cup \{v\} \) one at a time in topological order, \(|R \cup \{v\}| \) applications of the induction hypothesis show that \( I' \) is correct on every vertex in \( R \cup \{v\} \). But \( I' \) agrees with I on v, so I is correct on v.

**Analysis**

The subtrees arising in the basis of the construction have height either 0 or 1. The height added to the tree corresponding to a move in which \( p \) vertices are pebbled is \( p + \lceil \log_2(p + 1) \rceil \) which, for \( p \geq 1 \), is at most \( 2p \). Hence, if \( t \) vertices are pebbled in total, the height of \( T \) is at most \( 2t + 1 \).

**Corollary 4** (Paterson and Valiant [11]). \( \text{Size}(T(n)) \leq \text{Depth}(O(T(n)/\log T(n))) \), for all \( T(n) \geq n \).

**Proof.** This follows directly from Lemma 1 and Theorem 3.

**Theorem 5.** \( \text{DTIME}(T(n)) \subseteq \text{PRAM-time}(\sqrt{T(n)}) \), for any \( T(n) \geq n \).

**Proof.** We use a parallel implementation of the technique used by Hopcroft, Paul, and Valiant to speed up deterministic Turing machines by (ordinary) RAMs [8, Sect. 4]. Let \( M \) be a \( T(n) \) time-bounded deterministic Turing machine with \( k \) tapes (including the input tape), and consider \( M \)'s computation to be divided into blocks of size \( t = \sqrt{T(n)} \), as in Theorem 2. Given the value of t, the simulating PRAM \( P \) will use its first \( kt \) registers as an array corresponding to the \( kt \) blocks of \( M \)'s tapes. The content of each block is represented by an integer of \( O(t) \) bits, which \( P \) stores in the register corresponding to that block. Initially all the registers contain the integer representing a block of blank tape, except for the registers corresponding to the blocks of \( M \)'s input tape, which must be initialized appropriately to represent the symbols of the input. A *local configuration* \( C \) for \( M \) is a vector of integers consisting of an integer representing the current state and, for
each of the $k$ tapes, an integer representing the contents of that block containing the tape head, an integer representing the position of the tape head within that block, and integers representing the contents of the two neighboring blocks.

Given the array data structure described above, and using $k$ base registers to record the number of the block where each tape head resides, $P$ can obtain the current local configuration in a constant number of steps. In $t$ steps by $M$, only those tape squares in blocks in the local configuration can be read or altered, so to update the array data structure to reflect the tape contents after $t$ more steps of $M$, $P$ need only compute $\text{res}(C)$, the result of $t$ steps beginning from the local configuration $C$. ($\text{res}(C)$ consists of integers representing the new state, revised block contents and head positions, as well as information about which heads have moved out of their blocks and, if so, in which direction.) To compute $\text{res}(C)$ would require time $O(t)$ by a direct simulation, but can be done in constant time if $\text{res}(C)$ is available in a multidimensional table indexed by the components of $C$. Thus, prior to the start of the simulation phase described above, $P$ creates the table by initializing a separate processor for every possible value of $C$. (There are $\exp(O(t))$ possible values for $C$; this many processors can be initialized by a PRAM in time $O(t)$ in such a way that each gets a distinct value of $C$ [6].) Each processor then computes $\text{res}(C)$ by direct simulation of $t$ steps of $M$, and stores the result in the table in global memory.

Following initialization of the table and the array described above, $P$ performs $t$ updates of the array, each update revising the array to reflect $t$ more steps of $M$, and each update requiring constant time to perform. We have assumed that the value of $t$ is known to $P$ at the beginning of the algorithm, but if it is not $P$ can try successively larger powers of 2 as the value of $t$ until a successful value is found; in this case the time requirement is a geometric series whose growth rate is determined by its last term.  

It is worthwhile to consider the simulation of machines other than Turing machines. In $t$ steps of a Turing machine, only tape squares at distance $t$ or less from the initial positions of the heads can be modified. The technique used in the proof of the theorem is applicable whenever the simulated machine satisfies such a "locality of reference" property. Based on an earlier version of Theorem 5, Reif [18] has shown that a similar result can be obtained for the simulation of log cost RAMs. Variations in the simulating parallel machine are also possible: Ruzzo [personal communication] has shown that the simulation can be carried out by a vector machine [17]. However, a fan-in argument makes it clear that a circuit of constant depth could not perform the update step of the simulation above, which requires selecting one of the $\exp(O(t))$ table entries.

6. NONDETERMINISTIC SPEEDUPS

We have thus far examined speedups only for deterministic machines, and so we now briefly consider the situation with respect to nondeterminism. The techniques
of Theorem 2 do not suffice for simulating nondeterministic sequential machines by parallel machines, since concurrent processes in the simulating machine may choose differing sequences of guesses when performing the direct simulation of the nondeterministic machine starting from a particular configuration. If the simulated machine used its nondeterminism in such a restricted way that a "choice history" could be recorded in alternating space \( T/\log T \) (e.g., by only making a nondeterministic move every \( \log T \) steps), then the simulation could be carried through.

In the case of nondeterministic parallel machines, existing results can be classified into two categories: in the first, using nondeterminism affects the power of the machine by at most a polynomial factor; in the second, the nondeterministic parallel machine is apparently exponentially faster. Results of the first type were obtained by Pratt and Stockmeyer \[17\], where it was shown that time on both nondeterministic and deterministic vector machines is polynomially related to sequential space. Fortune and Wyllie \[6\] and Savitch \[20\] first obtained results of the second type by showing that time \( T \) on a nondeterministic PRAM is equivalent to time \( \exp(T) \) on a nondeterministic Turing machine. This difference in the effect of adding nondeterminism to parallel machines can be traced to the amount of nondeterminism available at each step of the computation. In \[4\], for example, a nondeterministic version of hardware modification machines \[5\] in which only one processor may make nondeterministic choices is considered. Such machines are shown to be capable of speeding up deterministic hardware modification machines by only a constant factor in time. (In fact even alternation does not help by more than a constant factor.) So this kind of nondeterminism is of no help in obtaining speedups (although it could reduce the amount of hardware used). The second type of result offers an exponential speedup of nondeterministic Turing machines by a version of nondeterministic hardware modification machine in which all the processors may make independent nondeterministic choices. Such an exponential speedup seems unlikely in the case in which the simulating parallel machine is deterministic.

7. Conclusion

We have improved the known speedups of deterministic multitape Turing machines by both fixed and variable structure parallel machines. In these results we have not restricted the number of processors used, which is unfortunately exponential in the time bound. (Because of the relationship between alternating Turing machines and uniform Boolean circuits, it is appropriate to take the processor bound of an alternating Turing machine to be the total number of configurations \[19\].) It would be interesting to know what speedups can be achieved using a number of processors that is only polynomial in the time bound.

It is worthwhile to note that even the most powerful variable structure model, the SIMDAG \[7\], can be simulated with only a square loss in time by alternating Turing machines. Thus, any improvement to Theorem 5 by a factor of \( \omega(\sqrt{\log T(n)}) \) in
the simulating time, even if the simulating parallel machine were a SIMDAG, would improve Theorem 2 and its corollaries.

ACKNOWLEDGMENTS

We thank Larry Ruzzo and Walter Savitch for enlightening discussions.

REFERENCES

21. M. TOMPA, A pebble game that models alternation, unpublished manuscript.