September 1993

UILU-ENG-93-2236
ACT-128

Applied Computation Theory

ARE PARALLEL MACHINES ALWAYS FASTER THAN SEQUENTIAL MACHINES?

Louis Mak

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

NSF

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-93-2236 ACT-128	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Laboratory University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION National Science Foundation
6c. ADDRESS (City, State, and ZIP Code) 1308 West Main Street Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Washington, DC 20050
8a. NAME OF FUNDING / SPONSORING ORGANIZATION National Science Foundation 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Washington, DC 20050	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Are Parallel Machines Always Faster Than Sequential Machines?

12. PERSONAL AUTHOR(S)
Mak, Louis

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	September 1993	27

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	computational complexity, time complexity, Turing machine, random access machine, tree Turing machine, multidimensional Turing machine, alternation, parallel machine

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

It is shown that parallel machines are always faster than sequential machines for a wide range of machine models, including the tree Turing machine, the multidimensional Turing machine, and the log-cost RAM (random access machine). More precisely, it is shown that every sequential machine M (in the above list) that runs in time T can be sped up by a parallel version M' of M that runs in time $o(T)$. All previous speedup results either rely on the severe limitation on the storage structure of M (e.g., M is a Turing machine with linear tapes) or require that M' has a more versatile storage structure than M (e.g., M' is a PRAM (parallel RAM), and M is a Turing machine with linear tapes). It is unclear whether it is the parallelism, or the restriction on the storage structures, or the combination of both that realizes such speedup. This paper removes all the above restrictions on storage structures in previous results. This paper presents speedup theorems where both M and M' use the same kind of storage medium, which is not linear tapes. Thus, parallelism alone suffices to achieve the speedup.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Are Parallel Machines Always Faster Than Sequential Machines?

Louis Mak[1]
Department of Computer Science and
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801
email: mak@grinch.csl.uiuc.edu

July 2, 1993

Abstract

It is shown that parallel machines are always faster than sequential machines for a wide range of machine models, including the tree Turing machine, the multidimensional Turing machine, and the log-cost RAM (random access machine). More precisely, it is shown that every sequential machine M (in the above list) that runs in time T can be sped up by a parallel version M' of M that runs in time $o(T)$. All previous speedup results either rely on the severe limitation on the storage structure of M (e.g., M is a Turing machine with linear tapes) or require that M' has a more versatile storage structure than M (e.g., M' is a PRAM (parallel RAM), and M is a Turing machine with linear tapes). It is unclear whether it is the parallelism, or the restriction on the storage structures, or the combination of both that realizes such speedup. This paper removes all the above restrictions on storage structures in previous results. This paper presents speedup theorems where both M and M' use the same kind of storage medium, which is not linear tapes. Thus, parallelism alone suffices to achieve the speedup.

Key words: computational complexity, time complexity, Turing machine, random access machine, tree Turing machine, multidimensional Turing machine, alternation, parallel machine, simulation.

AMS(MOS) subject classifications: 68Q05, 03D10, 68Q10, 68Q15, 03D15.

[1]Supported by the National Science Foundation under Grant CCR-8922008

1 Introduction and Motivation

For many problems, there exist parallel algorithms which run asymptotically faster than the best known sequential algorithms. Sorting, minimum spanning tree, and connected components are but a few examples [1, 10]. But is it always the case? In other words, given a sequential algorithm for a problem, does there always exist a parallel algorithm for the same problem that runs asymptotically faster? In more formal terms, this question can be phrased as follows. Consider a class of sequential machines $\mathcal{S}$ and their "parallel" counterparts $\mathcal{P}$. Is it true that each machine in the class $\mathcal{S}$ that runs in time T can be simulated by a machine in the class $\mathcal{P}$ that runs in time $o(T)$? In order to discuss this question rigorously, we need to fix $\mathcal{S}$ and $\mathcal{P}$. To facilitate our discussion, we call the simulating machine the *host* and the machine being simulated the *guest.*

The deterministic Turing machine is one of the standard models of sequential computation. Kozen [12], and independently Chandra and Stockmeyer [3], introduced a parallel version of the Turing machine, which has become known as the alternating Turing machine. Paul and Reischuk [17] established that every deterministic Turing machine running in time T can be simulated by an alternating Turing machine that runs in time $O(T \log\log T/\log T)$. Using a two-person pebble game, Dymond and Tompa [5] improved the simulation time to $O(T/\log T)$. Together with the alternating time hierarchy theorem [3], this result implies that the class of languages accepted by alternating Turing machines in time $O(T)$ strictly includes the class of languages accepted by deterministic Turing machines in time $O(T)$. Thus, parallel time is strictly more powerful than deterministic time for Turing machines.

However, it is unclear whether this relationship between parallel time and deterministic time is just an "artifact" of the Turing machine, because the proofs of both Paul and Reischuk [17] and Dymond and Tompa [5] rely heavily on the linearity of the tapes of the Turing machine. We

demonstrate that this relationship between parallel time and deterministic time also holds for Turing machines with tree-structured tapes and multidimensional tapes and random access machines (RAM's) under the log-cost measure. In a separate paper [15], we show that this relationship also holds for RAM's under the unit-cost measure. Thus, the statement "Parallel time is strictly more powerful than deterministic time" is neither an "artifact" of the linearity of tapes nor an "idiosyncrasy" of the Turing type of machine models. Instead, we believe that it is an intrinsic property of the nature of computation.

Dymond and Tompa [5] gave further evidence to our belief by showing that every deterministic Turing machine that runs in time T can be simulated by a CREW PRAM in time $O(T^{1/2})$. Nevertheless, the random access memory of the PRAM is much more flexible than the linear tapes of the Turing machine. It is questionable whether it is the parallelism, or the more flexible storage mechanism of the host, or the combination of both that makes such speedup possible. We settle this question conclusively by establishing speedup theorems where the guest uses the same type of storage mechanism as the host. Hence, parallelism alone suffices to achieve the speedup.

We present our results on tree Turing machines and multidimensional Turing machines in Section 2. In Section 3, we prove a speedup theorem for log-cost RAM's. We discuss the significance of our results in Section 4. All logarithms are taken to base 2.

2 Variants of the Turing Machine

Two common variants of the Turing machine are the tree Turing machine and the multidimensional Turing machine. A tree Turing machine is similar to the standard multitape Turing machine except that the work "tapes" are infinite rooted complete binary trees. Each tree node is a cell and can store a symbol of a finite alphabet. There is one head for each tape. Initially, the head is at the

root. In one step, the head can move to either the left child, the right child, or the parent of cell currently under the head.

Similarly, a multidimensional Turing machine differs from the standard multitape Turing machine in its storage structure. A work "tape" of the d-dimensional Turing machine consists of the set of cells $C = \{(x_1, x_2, \ldots, x_d) \mid \text{each } x_i \text{ is an integer}\}$. Two cells $(x_1, x_2, \ldots, x_d)$ and $(y_1, y_2, \ldots, y_d)$ are neighbors if and only if $\sum_{1 \leq i \leq d} |x_i - y_i| = 1$. Initially, the tape head is at the cell $(0, 0, \ldots, 0)$. In one step, the head can move to a neighbor of the cell currently under the head. In particular, a 1-dimensional Turing machine is just an ordinary linear-tape Turing machine.

We now prove the main theorem in this section:

Theorem 1 *Every deterministic tree Turing machine running in time T can be simulated by an alternating tree Turing machine running in time $O(T/\log T + n)$, where n is the length of the input.*

2.1 Definitions

Let M be a deterministic tree Turing machine running in time $T = T(n)$. We will construct an alternating tree Turing machine M' that simulates M in time $O(T/\log T + n)$ for sufficiently large n, say $n > n_0$. Theorem 1 then follows, since we can modify M' to handle inputs of length at most n_0 by table lookup.

The *address* of a cell is a string in $\{L, R\}^*$ that encodes the path from the root to that cell. For example, the cell with address LR is the right (R) child of the left (L) child of the root. We identify a cell with its address. Henceforth, when we say the cell u, we mean the cell with address u. The *depth* of a cell is the length of its address. By the following lemma, we may assume that M uses only cells of depth $O(\log T)$.

Lemma 2 [18] *Every deterministic tree Turing machine running in time T can be simulated by*

another deterministic tree Turing machine that runs in time $O(T)$ and uses only cells of depth $O(\log T)$.

For clarity, we assume that M has only one work tape (which also serves as the input tape). It is straightforward to generalize our arguments to handle multiple tapes. Initially, the input $\alpha = a_1 a_2 \cdots a_n$ is stored in the leftmost branch of the tree tape, that is, cell L^{i-1} holds a_i for $1 \leq i \leq n$. Without loss of generality, we assume that when M accepts, it loops forever in a distinguished state Q_{accept}.

Consider the computation of M on α. Let U_{ad} denote the portion of the tree tape that is visited by the tape head between time a and time d. Evidently, U_{ad} is contiguous. In graph theoretic terms, U_{ad} is a subtree of the tree tape. The *position* of U_{ad} is the address of the cell in U_{ad} that is closest to the root. For $a \leq c \leq d$, define the *partial configuration* of M at time c with respect to the time period from a to d to be the state of M, head position, and contents of U_{ad} at time c. Denote this partial configuration by $PC(a, c, d)$. We represent the head position by the address of the cell under the tape head.

2.2 The Simulation

We now describe M'. The task of M' is to answer the following question: Given the initial configuration of M, is M in state Q_{accept} at time T? This task can be generalized as follows. Let q be a question about the partial configuration $PC(a, c, d)$. Given the partial configuration $PC(a, a, d)$, is q true at time c? More precisely, q has one of the following forms:

1. Is M in state Q?

2. Is the tape head scanning the cell u?

3. Is a the content of the cell u (in U_{ad})?

Note that these questions, together with their answers, fully specify the partial configuration $PC(a, c, d)$.

We employ a procedure Φ for this generalized task. Let q' be a question about $PC(a, a, d)$, and $\Psi_{ad}(q')$ be a procedure that accepts if and only if q' is true at time a. Thus, Ψ_{ad} specifies $PC(a, a, d)$. We will explain how to implement Ψ_{ad} later. Procedure $\Phi[\Psi_{ad}, c]$ uses a specification Ψ_{ad} of $PC(a, a, d)$ to answer a question q about $PC(a, c, d)$; $\Phi[\Psi_{ad}, c](q)$ accepts if and only if q is true at time c. Notice that $\Phi[\Psi_{ad}, c]$ is one way to implement Ψ_{cd}. Let $f(\Psi_{ad})$ be the maximum (over all questions q' about $PC(a, a, d)$) running time of Ψ_{ad}.

Let q_0 be the question: "Is M in state Q_{accept}?" To determine whether M accepts the input, M' simply calls $\Phi[\Psi_{0T}, T](q_0)$. The running time of M' is therefore at most $f(\Phi[\Psi_{0T}, T])$. Below we give three different strategies to implement procedure Φ. In some cases one strategy is faster than the others, but no single strategy is always faster than the other two. Procedure Φ existentially chooses which strategy to use. Hence, the running time of Φ is the minimum among the three strategies.

2.2.1 Strategy 1

Denote by $|U_{ac}|$ the number of cells in U_{ac}. For Strategy 1, $\Phi[\Psi_{ad}, c](q)$ does the following:

1. Guess the position and shape of U_{ac},

2. Guess the state of M, head position, and contents of U_{ac} at time a, that is, $PC(a, a, c)$.

3. Universally choose to do (3a) and (3b)

   (a) i. Starting from the guessed $PC(a, a, c)$, simulate M for $(c-a)$ steps to obtain $PC(a, c, c)$.

   ii. Accept if and only if q is true at time c.

   (b) Verify the guesses in Step (2) with the help of Ψ_{ad}.

In Step (3(a)i), Φ checks that during the simulation, the tape head traced out the same subtree as the guessed U_{ac}; this ensures the correctness of the guesses in Step (1). In Step (3(a)ii), Φ has to decide whether q is true at time c. There are two exhaustive cases:

I. The question q asks about the state of M, head position, or content of a cell in U_{ac}, that is, $PC(a, c, c)$. Then after simulating M for $(c - a)$ steps, Φ has all the information necessary to decide whether q is true at time c.

II. The question q asks about the content of a cell v not in U_{ac}. Observe that the content of v is not changed from time a to time c, since the tape head stays within U_{ac} during this period. Thus, q is true at time c if and only if q is true at time a. To answer q, Φ simply calls $\Psi_{ad}(q)$.

To perform Step (3b), Φ universally chooses a question q' and calls $\Psi_{ad}(q')$; each q' corresponds to one guess in Step (2). Lemma 2 guarantees that $O(\log T)$ time suffices to write down the head position and the address of a cell (and hence a question q').

Because U_{ac} is a tree, recording the shape and contents of U_{ac} takes $O(|U_{ac}|) = O(c-a)$ time. By Lemma 2, the position of U_{ac} and the head position require $O(\log T)$ time to write down. Therefore, the running time of Strategy 1 is

$$f_1(\Phi[\Psi_{ad}, c]) \leq A_1(\log T + (c - a)) + f(\Psi_{ad}) \tag{1}$$

for some constant A_1.

2.2.2 Strategy 2

Define b to be $\lfloor (a+c)/2 \rfloor$. Let $\widehat{\Psi}_{bd}$ be the procedure $\Phi[\Psi_{ad}, b]$. For Strategy 2, $\Phi[\Psi_{ad}, c](q)$ recursively calls $\Phi[\widehat{\Psi}_{bd}, c](q)$. The running time of Strategy 2 is

$$f_2(\Phi[\Psi_{ad}, c]) \leq A_2(\log T) + f(\Phi[\widehat{\Psi}_{bd}, c]) \tag{2}$$

for some constant A_2, since $O(\log T)$ time suffices to set up the parameters for the recursive call.

2.2.3 Strategy 3

For $b \leq t \leq c$, define V_{bt} to be $U_{bt} \cap U_{ab}$. Let V be a subtree of the tree tape that includes all cells in V_{bc}. For Strategy 3, $\Phi[\Psi_{ad}, c](q)$ does the following:

1. Guess the position and shape of V.
2. Guess the state of M, head position, and contents of V at time b.
3. Universally chooses to do (3a), (3b), and (3c).
   (a) Verify the guesses in Step (1).
   (b) Verify the guesses in Step (2).
   (c) Recursively call $\Phi[\widetilde{\Psi}_{bc}, c](q)$, where $\widetilde{\Psi}_{bc}$ is a particular implementation of Ψ_{bc}.

In Section 2.2.5, we explain how to implement $\widetilde{\Psi}_{bt}$ for $b \leq t \leq c$, and in Section 2.2.4 we establish the following bound on the running time of Step (3a).

$$\text{The time required to verify } V \text{ in Step (3a) is } O(\log T + |V|) + \max \left\{ \begin{array}{l} \max\limits_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t']) \\ \\ \max\limits_{b \leq t < c} f(\Phi[\widetilde{\Psi}_{bt}, t]) \end{array} \right\} \tag{3}$$

Both Step (1) and Step (2) take $O(\log T + |V|)$ time. In Step (3b), Φ universally chooses a question q' and recursively calls $\Phi[\Psi_{ad}, b](q')$; each q' corresponds to one guess in Step (2). Step (3b) takes $O(\log T) + f(\Phi[\Psi_{ad}, b])$ time, since $O(\log T)$ time suffices to set up the parameters for the recursive call. For the same reason, Step (3c) takes $O(\log T) + f(\Phi[\widetilde{\Psi}_{bc}, c])$ time. The running time of Step (3) is the maximum among Steps (3a), (3b), and (3c). Therefore, the running time of Strategy 3 is

$$f_3(\Phi[\Psi_{ad}, c]) \leq A_3(\log T + |V|) + \max \left\{ \begin{array}{l} \max\limits_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t']) \\ \\ \max\limits_{b \leq t \leq c} f(\Phi[\widetilde{\Psi}_{bt}, t]) \end{array} \right\} \tag{4}$$

for some constant A_3.

2.2.4 Verifying V

We now give the details of Step (3a) of Strategy 3 and prove (3). To check that V includes all cells in V_{bc}, Φ universally chooses t in the range $b \leq t \leq c$ and checks that V includes all cells in V_{bt}, assuming that V includes $V_{b(t-1)}$. For each t, it suffices to check that the head position at time t is either in V or not in U_{ab}.

If $t = b$, then Φ uses the information recorded in Steps (1) and (2) of Strategy 3 to check that V includes the head position at time b.

If $t > b$, then Φ does the following:

1. Guess the state, head position, and content of the cell under the tape head at time $t - 1$.

2. Universally choose to do (2a) and (2b).

   (a) Universally choose a question q' about the guesses in Step (1) and recursively call $\Phi[\widetilde{\Psi}_{b(t-1)}, t-1](q')$ to verify these guesses.

(b) Simulate the $(t-1)^{st}$ step of M to obtain the head position u at time t. Check that u is either in V or not in U_{ab}.

After we explain how to implement $\widetilde{\Psi}_{b(t-1)}$ in Section 2.2.5, it will be evident that assuming V includes $V_{b(t-1)}$ is necessary to guarantee the correctness of $\widetilde{\Psi}_{b(t-1)}$. Note that Φ requires $\widetilde{\Psi}_{b(t-1)}$ to check that V includes V_{bt}.

In Step (2b), Φ has to check that u is not in U_{ab}, that is, the tape head does not visit u between time a and b. To check this, Φ first chooses universally a time t' between a and b, inclusive, and guesses a cell $v \neq u$. Let q_v be the question: "Is the tape head scanning cell v?" Next, Φ recursively calls $\Phi[\Psi_{ad}, t'](q_v)$ to ensure that the head is at $v \neq u$ at time t'. This takes $O(\log T) + \max_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t'])$ time.

Now (3) follows from the following observations:

(i) Apart from Step (2), $O(\log T + |V|)$ time suffices to verify V.

(ii) The running time of Step (2) is the maximum of Steps (2a) and (2b).

(iii) Step (2a) takes $O(\log T) + f(\Phi[\widetilde{\Psi}_{b(t-1)}, t-1])$ time, where $b < t \leq c$.

(iv) Step (2b) takes $O(\log T) + \max_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t'])$ time.

2.2.5 Implementing $\widetilde{\Psi}_{bt}$

We describe the implementation of $\widetilde{\Psi}_{bt}$ for $b \leq t \leq c$, assuming V includes V_{bt}. Recall that the input $\tilde{q}$ to $\widetilde{\Psi}_{bt}$ is a question about $PC(b, b, t)$. Depending on $\tilde{q}$, there are two exhaustive cases.

I. The question $\tilde{q}$ asks about the content of a cell u in $U_{bt} - V$. Then u is not in U_{ab}, since V includes $V_{bt} = U_{bt} \cap U_{ab}$. This implies the content of u is not changed between time a and b. Thus, $\tilde{q}$ is true at time b if and only if $\tilde{q}$ is true at time a. To answer $\tilde{q}$, $\widetilde{\Psi}_{bt}$ calls $\Psi_{ad}(\tilde{q})$.

II. The question $\tilde{q}$ asks about the state of M, head position, or contents of V. Then $\widetilde{\Psi}_{bt}$ uses the information recorded in Steps (1) and (2) of Strategy 3 to decide whether $\tilde{q}$ is true at time b. If so, $\widetilde{\Psi}_{bt}$ accepts; otherwise, it rejects.

Cases (I) and (II) take $f(\Psi_{ad})$ and $O(\log T + |V|)$ time respectively. Consequently, the running time of $\widetilde{\Psi}_{bt}$ is

$$f(\widetilde{\Psi}_{bt}) \leq A_4(\log T + |V|) + f(\Psi_{ad}) \tag{5}$$

for some constant A_4.

Notice that in Strategy 3, any subtree V that includes V_{bc} will work. It is easy to show that V_{bc} itself is subtree. We may thus assume that $V = V_{bc}$ in the sequel. The curious reader may wonder why we do not simply define V to be V_{bc}. This is because it is easier to verify $V \supseteq V_{bc}$ than $V = V_{bc}$.

2.3 Time Analysis

Combining Strategies 1, 2, and 3, we have

$$f(\Phi[\Psi_{ad}, c]) = \min_{1 \leq i \leq 3} f_i(\Phi[\Psi_{ad}, c]) \tag{6}$$

In Sections 2.3.1 and 2.3.2, we solve the system of recurrence relations (1), (2), (4), (5), and (6) to show that for some constant n_0,

$$f(\Phi[\Psi_{0T}, T]) = O(T/\log T) + f(\Psi_{0T}) \text{ for } n > n_0 \tag{7}$$

To answer a question about the initial configuration of M requires nothing more than scanning the input. This implies $f(\Psi_{0T}) = O(n)$. The running time of M' is thus $O(T/\log T + n)$ for $n > n_0$.

This completes the proof of Theorem 1. □

Remarks:

1. If we allow random access into individual cells of a separate input tape [2], then we can reduce $f(\Psi_{0T})$ to $O(\log T)$. In this case, M' runs in $O(T/\log T)$ time.

2. In the simulation described in Section 2.2, we have assumed that M has only one tape. If M has k tapes, then all k tapes are treated analogously. In addition, we modify the questions about tape contents to the form: "Is a the content of cell u of tape i?", where $1 \leq i \leq k$. These modifications do not increase the asymptotic running time of M'.

2.3.1 Computation Graph

The *computation graph* of M on input α is a directed acyclic graph G with vertices $\{0, 1, \ldots, T\}$. There is an edge from vertex i to vertex j if and only if either $j = i + 1$ or a cell is visited at time i and at time $j > i$, but not at time t for all $i < t < j$. If M has k tapes, then the indegree and outdegree of each vertex of G is at most $D = k + 1$.

Let $V(i, j)$ be the set of vertices $\{i, i + 1, \ldots, j\}$, and $e(i, j)$ be the number of edges in the subgraph of G induced by $V(i, j)$. Consider $0 \leq a < b < c \leq T$, where $b = \lfloor (a + c)/2 \rfloor$. For notational convenience, denote $e(a, b)$, $e(b, c)$, and $e(a, c)$ by e_1, e_2, and e respectively. Let e_3 be the number of edges from $V(a, b - 1)$ to $V(b + 1, c)$. We derive upper and lower bounds on e_1 and e_2 in terms of e.

Lemma 3 $e/(2D + 1) \leq e_1, e_2 \leq 2De/(2D + 1)$

Proof Let $m = b - a$. Then $m \leq c - b \leq m + 1 \leq 2m$, since $m \geq 1$. Clearly, $b - a \leq e_1 \leq (b - a)D$, and $c - b \leq e_2 \leq (c - b)D$. Thus, $m \leq e_1, e_2 \leq 2mD$. Since $e \geq e_1 + e_2$, we have $e/e_1 \geq (e_1 + e_2)/e_1 = 1 + e_2/e_1 \geq 1 + m/2mD = (2D + 1)/2D$. The same argument yields $e/e_2 \geq (2D + 1)/2D$.

The number of edges from $V(a,c)$ to $V(b+1,c)$ is e_2+e_3. The total number of edges pointing to $V(b+1,c)$ is at most $D(c-b) \leq 2mD$. Hence, $e_2+e_3 \leq 2mD$, and $e/e_1 = (e_1+e_2+e_3)/e_1 = 1+(e_2+e_3)/e_1 \leq 1+2mD/m = 1+2D$. Similarly, considering the total number of edges originating from $V(a,b-1)$, we get $e_1+e_3 \leq (b-a)D \leq 2mD$, and $e/e_2 \geq 1+2D$. □

2.3.2 Solving the Recurrence

We now prove (7) by solving (1), (2), (4), (5), and (6). Let $e_0 = (2D+1)^{4D+1}\log T$, $A = 2\sum_{1\leq i\leq 4} A_i$, and $B = 4A(2D+1)^2\log(2D+1)$. As n tends to infinity, so does T. Thus, $T \geq e_0$ if n exceeds some constant n_0. Define $h(e)$ to be $e/\log T$. We show by induction on $e = e(a,c)$ that

$$f(\Phi[\Psi_{ad},c]) \leq Be/\log h(e) + f(\Psi_{ad}) \text{ for } e \geq e_0 \tag{8}$$

Setting $a = 0$ and $c = d = T$, we get $T \leq e = e(a,c) = e(0,T) \leq DT$, and (7) follows immediately from (8). We present three lemmas and then give the proof of (8).

Lemma 4 *For $x \geq 3$, $(\log x)/x$ decreases with x, and for $x \geq 3\log T$, $x/\log h(x)$ increases with x.*

Proof By elementary calculus, for $x \geq 3$, $(\log x)/x$ decreases with x, and $x/\log x$ increases with x. Let $x' = h(x)$. Then $x/\log h(x) = \log T(x'/\log x')$ increases with x' for $x' \geq 3$. Thus, $x/\log h(x)$ increases with x for $x \geq 3\log T$. □

Lemma 5 *For $y \geq 16$ and $z \geq 1$,* $\dfrac{\log y}{y} + \dfrac{1-2z/\log y}{1-z/\log y} \leq 1.$

Proof

$$\frac{\log y}{y} + \frac{1-2z/\log y}{1-z/\log y} \leq \frac{\log y}{y} + \frac{1-2z/\log y + (z/\log y)^2}{1-z/\log y}$$

$$
\begin{aligned}
&= \frac{\log y}{y} + 1 - \frac{z}{\log y} \\
&\leq 1 \qquad \begin{array}{l}(\text{since } y \geq 16 \text{ and } z \geq 1 \text{ imply} \\ z/\log y \geq 1/\log y \geq (\log y)/y)\end{array} \quad \square
\end{aligned}
$$

Lemma 6 *If* $e \leq (2D+1)e_0$, *then* $f(\Phi[\Psi_{ad}, c]) \leq Be_0/\log h((2D+1)e_0) + f(\Psi_{ad})$.

Proof By (1) and (6),

$$
\begin{aligned}
f(\Phi[\Psi_{ad}, c]) - f(\Psi_{ad}) &\leq A(\log T + (c-a)) && (\text{since } A \geq A_1) \\
&\leq A(\log T + (2D+1)e_0) && (\text{since } c - a \leq e \leq (2D+1)e_0) \\
&\leq 2A(2D+1)e_0 && (\text{since } (2D+1)e_0 \geq \log T) \\
&= \frac{Be_0}{\log h((2D+1)e_0)} \quad \square
\end{aligned}
$$

Proof of (8) (by induction on e)

Base Case: $e_0 \leq e \leq (2D+1)e_0$

By Lemma 6, $f(\Phi[\Psi_{ad}, c]) - f(\Psi_{ad}) \leq Be_0/\log h((2D+1)e_0) \leq Be/\log h(e)$.

Inductive Step: $e > (2D+1)e_0$

There are two exhaustive cases.

Case I: $e_3 \geq 2e\log(2D+1)/\log h(e)$

Since $e > (2D+1)e_0 = (2D+1)^{4D+2}\log T$, it follows that $h(e) = e/\log T > (2D+1)^{4D+2}$. By definition, $D \geq 1$. Hence,

$$h(e) \geq 16 \tag{9}$$

By Lemma 3, $e > e_1, e_2 \geq e/(2D+1) > e_0$. From (2) and (6),

$$
\begin{aligned}
& f(\Phi[\Psi_{ad}, c]) & \\
\leq\ & B\log T + f(\Phi[\widehat{\Psi}_{bd}, c]) & \text{(since } B \geq A \geq A_2\text{)} \\
\leq\ & B\log T + \frac{Be_2}{\log h(e_2)} + f(\widehat{\Psi}_{bd}) & \text{(by induction hypothesis, since } e > e_2 \geq e_0\text{)} \\
\leq\ & B\log T + \frac{Be_2}{\log h(e_2)} + f(\Phi[\Psi_{ad}, b]) & \text{(since } \widehat{\Psi}_{bd} = \Phi[\Psi_{ad}, b] \text{ by definition)} \\
\leq\ & B\log T + \frac{Be_2}{\log h(e_2)} + \frac{Be_1}{\log h(e_1)} + f(\Psi_{ad}) & \text{(by induction hypothesis, since } e > e_1 \geq e_0\text{)} \\
\leq\ & B\log T + \frac{B(e-e_3)}{\log h(e/(2D+1))} + f(\Psi_{ad}) & \text{(by Lemma 3)} \\
\leq\ & \frac{Be}{\log h(e)}\left(\frac{\log y}{y} + \frac{1-2z/\log y}{1-z/\log y}\right) + f(\Psi_{ad}) & \text{where } y = h(e) \geq 16 \text{ and } z = \log(2D+1) \geq 1 \text{ (by (9) and hypothesis on } e_3\text{)} \\
\leq\ & \frac{Be}{\log h(e)} + f(\Psi_{ad}) & \text{(by Lemma 5)}
\end{aligned}
$$

Case II: $e_3 < 2e\log(2D+1)/\log h(e)$

By (4) and (6),

$$
f(\Phi[\Psi_{ad}, c]) \leq A_3(\log T + |V|) + \max\left\{ \begin{array}{l} \max\limits_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t']) \\ \\ \max\limits_{b \leq t \leq c} f(\Phi[\widetilde{\Psi}_{bt}, t]) \end{array} \right\} \tag{10}
$$

We now bound $\max_{a \leq t' \leq b} f(\Phi[\Psi_{ad}, t'])$ and $\max_{b \leq t \leq c} f(\Phi[\widetilde{\Psi}_{bt}, t])$. For $a \leq t' \leq b$, $e(a,t') \leq e(a,b) < e$. If $e(a,t') \geq e_0$, then

$$
\begin{aligned}
f(\Phi[\Psi_{ad}, t']) - f(\Psi_{ad}) \ &\leq\ \frac{Be(a,t')}{\log h(e(a,t'))} && \text{(by induction hypothesis, since } e > e(a,t') \geq e_0\text{)} \\
&\leq\ \frac{Be_1}{\log h(e_1)} && \text{(by Lemma 4, since } e_1 \geq e(a,t') \geq e_0 \geq 3\log T\text{)}
\end{aligned}
$$

Otherwise, $e(a,t') < e_0$. Then by Lemma 6,

$$\begin{aligned} f(\Phi[\Psi_{ad},t']) - f(\Psi_{ad}) &\leq \frac{Be_0}{\log h((2D+1)e_0)} \\ &\leq \frac{Be_0}{\log h(e_0)} \\ &\leq \frac{Be_1}{\log h(e_1)} \qquad \text{(by Lemma 4, since } e_1 \geq e/(2D+1) > e_0 \geq 3\log T\text{)} \end{aligned}$$

Hence,

$$\max_{a \leq t' \leq b} f(\Phi[\Psi_{ad},t']) \leq \frac{Be_1}{\log h(e_1)} + f(\Psi_{ad}) \tag{11}$$

By similar arguments,

$$\max_{b \leq t \leq c} f(\Phi[\widetilde{\Psi}_{bt},t]) \leq \frac{Be_2}{\log h(e_2)} + f(\widetilde{\Psi}_{bt}) \tag{12}$$

From (10), (11), (12), and (5),

$$f(\Phi[\Psi_{ad},c]) \leq (A_3+A_4)(\log T + |V|) + \max\left\{\frac{Be_1}{\log h(e_1)}, \frac{Be_2}{\log h(e_2)}\right\} + f(\Psi_{ad})$$

Now $|V| = |V_{bc}| = |U_{ab} \cap U_{bc}| = e_3 + 1$. Therefore,

$$\begin{aligned} f(\Phi[\Psi_{ad},c]) - f(\Psi_{ad}) &\leq A(\log T + e_3) + \max\left\{\frac{Be_1}{\log h(e_1)}, \frac{Be_2}{\log h(e_2)}\right\} \quad (\text{since } A \geq 2(A_3+A_4)) \\ &\leq A\log T + Ae_3 + \frac{2BDe/(2D+1)}{\log h(e/(2D+1))} \qquad \text{(by Lemma 3)} \\ &= \frac{Be}{\log h(e)}(X+Y+Z) \end{aligned}$$

where

$$X = \frac{A\log h(e)}{Bh(e)}$$

$$
\begin{aligned}
Y &= \frac{Ae_3}{Be}\log h(e) \\
Z &= \frac{2D/(2D+1)}{1-\log(2D+1)/\log h(e)}
\end{aligned}
$$

It remains to show that $X+Y+Z \le 1$.

$$
\begin{aligned}
X = \frac{A\log h(e)}{Bh(e)} &\le \frac{A(4D+2)\log(2D+1)}{B(2D+1)^{4D+2}} && \text{(by Lemma 4, since } e > (2D+1)e_0 \ge 3\text{)} \\
&= \frac{1}{2(2D+1)(2D+1)^{4D+2}} \\
X &\le \frac{1/2}{4D+2} && \text{(since } D \ge 1\text{)} \qquad (13) \\
Y = \frac{Ae_3}{Be}\log h(e) &\le \frac{A}{B}2\log(2D+1) && \text{(by hypothesis on } e_3\text{)} \\
&= \frac{1}{2(2D+1)^2} \\
Y &\le \frac{1/2}{4D+2} && \text{(since } D \ge 1\text{)} \qquad (14) \\
Z = \frac{2D/(2D+1)}{1-\log(2D+1)/\log h(e)} &\le \frac{4D}{4D+1} && \text{(since } e > (2D+1)e_0\text{)} \\
Z &\le \frac{4D+1}{4D+2} && (15)
\end{aligned}
$$

By (13), (14), and (15), $X+Y+Z \le 1$. This completes the proof of (8). □

2.4 Effects of Storage Structure on Alternating Turing Machine

In Theorem 1, we simulate a deterministic tree Turing machine by an alternating tree Turing machine. The simulator needs tree tapes for the stepwise simulation in Strategy 1 (Section 2.2.1). In fact, we can simulate a deterministic tree Turing machine by an alternating Turing machine with linear tapes without increasing the asymptotic running time of the simulator. This follows from Theorem 1 and our next theorem.

Theorem 7 *Every alternating tree Turing machine running in time T can be simulated by an alternating 1-dimensional Turing machine in time $O(T)$.*

Proof Let M be an alternating tree Turing machine with k tapes and time complexity T. We describe an alternating 1-dimensional Turing machine M' with $k+1$ tapes that simulates M in time $O(T)$. M' keeps track of the positions of the tape heads of M on k tapes and maintains an additional "transcript tape." M' simulates M step by step and records on the transcript tape the steps performed by M. To simulate M for one step, M' guesses the symbols read by the tape heads of M and universally chooses to do (i) carry on with the simulation and (ii) verify the guesses.

M' represents each head position by a string in $\{L, R\}^*$, which encodes the path from the root to the cell under the tape head. Using this representation, M' can maintain every head position in $O(1)$ time per simulated step. To verify a guess of the symbol read by a tape head, M' exploits the information recorded on the transcript tape to "re-run" the simulation, remembering the last symbol written on the cell at the recorded position of the tape head. M' checks that the last symbol written agrees with the guessed symbol. Re-running the simulation takes $O(T)$ time. Hence, M' runs in $O(T)$ time. □

Theorem 7 shows that allowing tree tapes does not increase the computing power of alternating Turing machines. Similarly, allowing multidimensional tapes does not increase the computing power of alternating Turing machines.

Theorem 8 [14] *Every alternating d-dimensional Turing machine running in time T can be simulated by an alternating 1-dimensional Turing machine in time $O(T)$.*

Thus, the class of languages accepted by alternating Turing machines is invariant under different storage structures, namely, linear tapes, tree tapes, and multidimensional tapes. By Theorems 7 and

8, we may, without loss of precision, talk about alternating Turing machines without mentioning the storage structure.

Lemma 9 [19] *Every deterministic d-dimensional Turing machine that runs in time T can be simulated by a deterministic tree Turing machine in time $O(T5^{d\log^* T})$.*

Theorem 10 *Every deterministic d-dimensional Turing machine that runs in time T can be simulated by an alternating Turing machine in time $O(T5^{d\log^* T}/\log T + n)$.*

Proof Immediate from Theorems 1, 7, and Lemma 9. □

As in Theorem 1, we can reduce the running time of the simulator in Theorem 10 to $O(T5^{d\log^* T}/\log T)$ if we allow random access into the input tape. Theorem 10 is an improvement of an earlier result by Mak [14], which shows that every deterministic d-dimensional Turing machine with time complexity T can be simulated by an alternating Turing machine in time $O(T/(\log T)^{1/d})$.

3 The Log-cost RAM

Cook and Reckhow [4] introduced the log-cost RAM. A RAM consists of a read-only input tape, a memory, and a finite program. The memory is an infinite sequence of registers (r_i), $i = 0, 1, \ldots$. The *address* of r_i is the integer i. Each register can hold an integer. Denote by $\langle r_i \rangle$ the content of r_i. Initially, $\langle r_i \rangle = 0$ for all i. The input alphabet is $\{0, 1\}$. The input string is often interpreted as a binary number. The program consists of a finite number of instructions. The allowed instructions are shown in Table 1. The length of an integer is the number of bits in its binary representation. A log-cost RAM executes each instruction in time proportional to the sum of the lengths of the operands and the lengths of the addresses involved in the instruction.

Instruction	Meaning
$r_i \leftarrow C$	r_i gets C, where C is an integer constant.
$r_i \leftarrow r_j$	r_i gets $\langle r_j \rangle$.
$r_i \leftarrow (r_j)$	r_i gets $\langle r_k \rangle$, where $k = \langle r_j \rangle$.
$(r_j) \leftarrow r_i$	r_k gets $\langle r_i \rangle$, where $k = \langle r_j \rangle$.
$r_i \leftarrow r_j + r_k$	r_i gets $\langle r_j \rangle + \langle r_k \rangle$.
$r_i \leftarrow r_j - r_k$	r_i gets $\langle r_j \rangle - \langle r_k \rangle$.
BRANCH q	If $\langle r_0 \rangle \geq 0$, then pass control to the q^{th} instruction; otherwise, pass control to the next instruction.
READ	r_0 gets the input bit under the tape head.
LEFT	Move the tape head left one cell.
RIGHT	Move the tape head right one cell.
ACCEPT	Accept and halt.
REJECT	Reject and halt.

Table 1: Instructions of a RAM

Lemma 11 [18] *Every log-cost RAM that runs in time T can be simulated by a deterministic tree Turing machine in time $O(T)$.*

An alternating RAM is a RAM equipped with existential and universal choices. Like an alternating Turing machine, an alternating RAM can branch out existentially and universally. The definition of acceptance for an alternating RAM is analogous to that of an alternating Turing machine [2].

Katajainen et al. [11] showed that every deterministic Turing machine with time complexity T can be simulated by a log-cost RAM in time $O(T \log\log T)$ by exhibiting a step by step simulation of the former by the latter. This simulation takes $O(\log T \log\log T)$ time per $O(\log T)$ simulated steps. It is straightforward to adapt this stepwise simulation to prove the following lemma.

Lemma 12 *Every alternating Turing machine with time complexity T can be simulated by an alternating log-cost RAM in time $O(T \log\log T)$.*

Proof We augment the simulation described by Katajainen et al. [11] to handle universal and existential choices. In the step by step simulation, we simulate each universal and existential choice of the alternating Turing machine by a corresponding choice of the alternating log-cost RAM. □

Theorem 13 *Every log-cost RAM that runs in time T can be simulated by an alternating log-cost RAM in time $O((T/\log T + n)\log\log T)$.*

Proof By successive application of Lemma 11, Theorem 1, Theorem 7, and Lemma 12. □

Again, we can reduce the simulation time in Theorem 13 to $O(T\log\log T/\log T)$ if we allow the log-cost RAM to have random access into individual cells of the input tape.

4 Discussion

4.1 Parallel time is more powerful than deterministic time

We have shown that for wide range of machine models, parallel machines are indeed faster than their sequential counterparts (Theorems 1, 10, and 13).

Since the alternating time hierarchy is sharp [3], as a corollary of Theorems 1 and 10, we conclude that the class of languages accepted by alternating Turing machines in time $O(T)$ strictly includes the class of languages accepted by deterministic Turing machines in time $O(T)$, regardless of the type of storage structure. By Theorem 13 and the time hierarchy theorem for log-cost RAM's [4], we deduce that the class of languages accepted by alternating log-cost RAM's in time $O(T)$ strictly includes the class of languages accepted by deterministic log-cost RAM's in time $O(T)$. Furthermore, the class of languages accepted by CREW PRAM's in time $O(T)$ strictly includes the class of languages accepted by unit-cost RAM's in time $O(T)$ [15].

We conclude that parallel time is strictly more powerful than deterministic time as a resource of computation.

4.2 A second parallel computation thesis

Let *PSPACE* denote the class of languages accepted by deterministic Turing machines in polynomial space. Fortune and Wyllie [6], Savitch and Stimson [21], and Savitch [20] showed that the classes of languages accepted by various parallel machine models in polynomial time are all equal to *PSPACE*. These results, among others, prompted Goldschlager [7] to promulgate the celebrated "parallel computation thesis," which asserts that the class of languages accepted by any "reasonable" parallel machine model in polynomial time is equivalent to *PSPACE*. The parallel computation thesis provides a criterion to judge whether a parallel machine model is reasonable [16]. This thesis gives an excellent characterization of what parallel machines can do in polynomial time, but it fails to characterize what parallel machines can do in time T for a specific time complexity function T.

Encouraged by our results, we propose the following "second parallel computation thesis."

> The class of languages accepted by any "reasonable" parallel machine model $\mathcal{P}$ in time $O(T)$ strictly includes the class of languages accepted by the "sequential counterpart" of $\mathcal{P}$ in time $O(T)$.

Hence, a "parallel" machine model which cannot speedup its sequential counterpart is too weak to be called a parallel machine model. Contrarily, a "sequential" machine model which cannot be sped up by its parallel counterpart is too strong to be called a sequential model of computation. The second parallel computation thesis provides an additional criterion to gauge the reasonableness of a parallel machine model. Moreover, unlike the original parallel computation thesis, the thesis we proposed is not restricted to polynomial time, but applies to any time complexity function T.

4.3 Effects of storage structure

We demonstrated that a more flexible storage structure does not increase the computing power of an alternating Turing machine (Theorems 7 and 8). Similarly, having multidimensional memories does not increase the computing power of a CREW or CRCW PRAM [15]. Thus, the class of languages accepted by a parallel machine model in time $O(T)$ seems to be invariant under different storage structures. In other words, the power of parallelism is strong enough to overcome the handicap in the storage medium. This observation suggests a further criterion that a reasonable parallel machine model should satisfy: for a reasonable parallel machine model, a modest change in its storage structure should not affect the computing power of the model.

Note that for the deterministic Turing machine, a standard model of sequential computation, a change in the storage mechanism does affect the computing power [8, 9, 13].

Acknowledgements

I would like to thank my advisor, Professor Michael C. Loui, for his motivation and guidance. His extensive suggestions significantly improved the disposition of this paper.

References

[1] S. G. Akl. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[2] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. Assoc. Comput. Mach.*, 28(1):114–133, 1981.

[3] A. K. Chandra and L. J. Stockmeyer. Alternation. In *Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science*, pages 98–108, 1976.

[4] S. A. Cook and R. A. Reckhow. Time bounded random access machines. *J. Comput. System Sci.*, 7:354–375, 1973.

[5] P. W. Dymond and M. Tompa. Speedups of deterministic machines by synchronous parallel machines. *J. Comput. System Sci.*, 30:149–161, 1985.

[6] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc. 10th Ann. ACM Symp. on Theory of Computing*, pages 114–118, 1978.

[7] L. M. Goldschlager. A universal interconnection pattern for parallel computers. *J. Assoc. Comput. Mach.*, 29:1073–1086, 1982.

[8] D. Y. Grigor'ev. Imbedding theorems for Turing machines of different dimensions and Kolmogorov's algorithms. *Soviet Math. Dokl.*, 18:588–592, 1977.

[9] F. C. Hennie. On-line Turing machine computations. *IEEE Trans. Elec. Comput.*, 15:35–44, 1966.

[10] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 17, pages 869–941. MIT Press, Cambridge, Massachusetts, 1990.

[11] J. Katajainen, J. van Leeuwen, and M. Penttonen. Fast simulation of Turing machines by random access machines. *SIAM J. Comput.*, 17:77–88, 1988.

[12] D. Kozen. On parallelism in Turing machines. In *Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science*, pages 89–97, 1976.

[13] M. C. Loui. Optimal dynamic embedding of trees into arrays. *SIAM J. Comput.*, 12:463–472, 1983.

[14] L. Mak. Speedup of determinism by alternation for multidimensional Turing machines. Submitted to *Theoret. Comput. Sci.* for publication, 1992.

[15] L. Mak. Parallelism always helps. In preparation, 1993.

[16] I. Parberry. *Parallel Complexity Theory.* Wiley, New York, 1987.

[17] W. Paul and R. Reischuk. On alternation II. *Acta Inform.*, 14:391–403, 1980.

[18] W. Paul and R. Reischuk. On time versus space II. *J. Comput. System Sci.*, 22:312–327, 1981.

[19] K. R. Reischuk. A fast implementation of a multidimensional storage into a tree storage. *Theoret. Comput. Sci.*, 19:253–266, 1982.

[20] W. J. Savitch. Parallel random access machines with powerful instruction sets. *Math. Systems Theory*, 15:191–210, 1982.

[21] W. J. Savitch and M. J. Stimson. Time bounded random access machines with parallel processing. *J. Assoc. Comput. Mach.*, 26:103–118, 1979.