August 1989

UILU-ENG-89-2222
ACT-108

COORDINATED SCIENCE LABORATORY
College of Engineering
Applied Computation Theory

EFFICIENT ON-LINE SIMULATIONS OF TREE MACHINES AND MULTIDIMENSIONAL TURING MACHINES BY RANDOM ACCESS MACHINES

Michael C. Loui
David R. Luginbuhl

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
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-89-2222 (ACT #108)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Air Force Institute of Technology Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Wright-Patterson AFB Arlington, VA 22217 Ohio 45433
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFIT Office of Naval Research 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ONR N00014-85-K-0570
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Wright-Patterson AFB Arlington, VA 22217 Ohio 45433	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Efficient On-Line Simulations of Tree Machines and Multidimensional Turing Machines by Random Access Machines

12. PERSONAL AUTHOR(S)
Loui, Michael C. and Luginbuhl, David R.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM ______ TO ______	August, 1989	31

16. SUPPLEMENTARY NOTATION

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

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

We establish an optimal on-line relationship between tree machines and random access machines (RAMs). We present an on-line simulation of a tree machine of time complexity t by a log-cost RAM of time complexity $O((t \log t)/\log\log t)$. Using information-theoretic techniques, we show that this simulation is optimal.

We adapt the simulation of a tree machine to devise an on-line simulation of a d-dimensional Turing machine of time complexity t by a log-cost RAM running in time $O(t(\log t)^{1-1/d}(\log\log t)^{1/d})$.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION 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

Efficient On-Line Simulations of Tree Machines and Multidimensional Turing Machines by Random Access Machines *

Michael C. Loui†
Department of Electrical and Computer Engineering and
Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, IL 61801

David R. Luginbuhl‡
Department of Computer Science and
Beckman Institute
University of Illinois at Urbana-Champaign
Urbana, IL 61801

August 24, 1989

*The views, opinions, and conclusions in this paper are those of the authors and should not be construed as an official position of the Department of Defense, U.S. Air Force, or other U.S. government agency.

†Supported by the Office of Naval Research under Contract N00014-85-K-0570.

‡Supported by the Air Force Institute of Technology, AFIT/CIRD, Wright-Patterson AFB, OH 45433.

Abstract

We establish an optimal on-line relationship between tree machines and random access machines (RAMs). We present an on-line simulation of a tree machine of time complexity t by a log-cost RAM of time complexity $O((t \log t)/\log\log t)$. Using information-theoretic techniques, we show that this simulation is optimal.

We adapt the simulation of a tree machine to devise an on-line simulation of a d-dimensional Turing machine of time complexity t by a log-cost RAM running in time $O(t(\log t)^{1-1/d}(\log\log t)^{1/d})$.

1 Introduction

The random access machine (RAM) and the Turing machine (TM) are the standard models for sequential computation. Research into the use of time and space by these and other models gives us insight into their computational power. This research includes analyzing how two different models use time and space, and comparing time and space utilization within a single model. Another avenue of investigation is determining how altering the definitions of time and space (for example, log-cost versus unit-cost) for a model affects its computational power. Slot and van Emde Boas (1988), for example, showed how space equivalence of RAMs and Turing machines is affected by varying the definition of space complexity for RAMs.

Paul and Reischuk (1981) used tree machines to investigate the relationships between time and space for random access machines and multidimensional Turing machines. They presented a simulation of a log-cost RAM of time complexity t by a tree machine of time complexity $O(t)$. They also showed that a tree machine of time complexity t can be simulated off-line by a unit-cost RAM of time complexity $O(t/\log\log t)$. Loui (1984b) showed that a multihead tree machine of time complexity t can be simulated by a tree machine with only two worktape heads in time $O((t\log t)/\log\log t)$.

We present an on-line simulation of a tree machine of time complexity t by a log-cost RAM of time complexity $O((t\log t)/\log\log t)$. Using the notion of incompressibility from Kolmogorov complexity (Li and Vitanyi, 1988), we show that this simulation is optimal. This appears to be the first application of Kolmogorov complexity to sequential RAMs. It is significant because few algorithms have been shown to be optimal.

Using similar techniques, we design an efficient on-line simulation of a d-dimensional Turing machine of time complexity t by a log-cost RAM running in time $O(t(\log t)^{1-1/d}(\log\log t)^{1/d})$. For $d = 1$, the running time is $O(t \log\log t)$, which is the same as the result of Katajainen *et al.* (1988).

This work is a complement to Loui's (1983) simulation of tree machines by multidimensional Turing machines and Reischuk's (1982) simulation of multidimensional Turing machines by tree machines.

All logarithms in this paper are taken to base 2.

2 Machine Definitions

All machines that we consider have a two-way read-only input tape and a one-way write-only output tape. The principal differences in the machines are in their storage structures.

A *tree machine*, a generalization of a Turing machine, has a storage structure that consists of a finite collection of complete infinite rooted binary trees, called *tree worktapes*. Each cell of a worktape can store a 0 or 1. Each worktape has one head. A worktape head can shift to a cell's parent or to its left or right child. Initially, every worktape head is on the root of its worktape, and all cells contain 0.

Let W be a tree worktape. We fix a natural bijection between the positive integers and cells of W. We refer to the integer corresponding to a particular cell as that cell's *location*. Write cell(b) for the cell at location b. Define cell(1) as the root of W. Then cell($2b$) is the left child of cell(b) and cell($2b+1$) is the right child of cell(b).

Each step of a tree machine consists of reading the contents of the worktape cells and input cell currently scanned, writing back on the same worktape cells and (possibly) to the currently accessed output cell, and (possibly) shifting each worktape head and the input head. If the tree machine writes on the output tape, it also shifts the output head.

The *time complexity* $t(n)$ of a tree machine is defined in the natural way.

A multihead d-dimensional Turing machine consists of a finite control and a finite number of d-dimensional worktapes, each with one worktape head. A d-dimensional worktape comprises an infinite number of cells, each of which is assigned a d-tuple of integers called the *coordinates* of the cell. The coordinates of adjacent cells differ in just one component of the d-tuple by ± 1. At each step of the computation, the machine reads the symbols in the currently accessed input and worktape cells, (possibly) writes symbols on the currently accessed output and worktape cells, (possibly) shifts the input head, and shifts each worktape head in one of $2d + 1$ directions – either to one of $2d$ adjacent cells or to the same cell.

The random access machine (RAM) (Aho *et al.*, 1974; Cook and Reckhow, 1973; Katajainen *et al.*, 1988) consists of the following: a finite sequence of labeled instructions; a memory consisting of an infinite sequence of registers, indexed by nonnegative integer addresses (register $r(j)$ has address j); and a special register AC, called the accumulator, used for operating on data. Each register, including AC, holds a nonnegative integer; initially all registers contain 0. Each cell on the input tape contains a 0 or 1. The following RAM instructions are allowed ($\langle x \rangle$ denotes the contents of register $r(x)$; $\langle AC \rangle$

denotes the contents of AC):

input. Read the current input symbol into AC and move the input head one cell to the right.

output. Write $\langle AC \rangle$ to the output tape and move output head one cell to the right.

jump θ. Unconditional transfer of control to instruction labeled θ.

jgtz θ. Transfer control to instruction labeled θ if $\langle AC \rangle > 0$.

load $=C$. Load integer C into AC.

load j. Load $\langle j \rangle$ into AC.

load $*j$. (Load indirect) Load $\langle \langle j \rangle \rangle$ into AC.

store j. Store $\langle AC \rangle$ into $r(j)$.

store $*j$. (Store indirect) Store $\langle AC \rangle$ into register $r(\langle j \rangle)$.

add j. Add $\langle j \rangle$ to $\langle AC \rangle$ and place result in AC.

sub j. If $\langle j \rangle > \langle AC \rangle$, then load 0 into AC; otherwise, subtract $\langle j \rangle$ from $\langle AC \rangle$ and place result in AC.

Define the *length* of a nonnegative integer i as the minimum positive integer w such that $i \leq 2^w - 1$ (approximately the logarithm of i). The length of a register is the length of the integer contained in the register (note that the length of a register is a time-dependent quantity).

We consider two time complexity measures for RAMs, based on the cost of each RAM instruction. For the *unit-cost RAM*, we charge each instruction one unit of time. For the *log-cost RAM*, we charge each instruction according to the *logarithmic cost criterion* (Katajainen *et al.*, 1988): the time for each instruction is the sum of the lengths of the integers (addresses and register contents) involved in its execution. The *time complexity* $t(n)$ of a RAM is the maximum total time used in computations on inputs of length n. It is possible, of course, to define time complexity in other ways; e.g., we could charge some other function $f(j)$ for access to register j (Aggarwal *et al.*, 1987).

In our simulations, we group the registers into a finite number of memories, each memory containing an infinite number of registers. This does not increase the cost in time by more than a constant factor, since we could simply interleave these memories into one memory (Katajainen *et al.*, 1988).

We use a technique of Katajainen *et al.* (1988) to pack and unpack registers in order to find the bit representation of a number and vice-versa. This divide-and-conquer strategy involves precomputed shift tables:

Lemma 2.1 (Katajainen *et al.*, 1988) *If the proper tables are available, then it is possible to compute the u-bit representation of an integer* $n < 2^u$, *and the numeric value of a u-bit string, both in* $O(u \log u)$ *time on a log-cost RAM.*

Lemma 2.2 (Katajainen *et al.*, 1988) *The tables necessary for Lemma 2.1 can be built in* $O(u2^u)$ *time on a log-cost RAM.*

A machine M of time complexity t is simulated by a machine M' *on-line* in time $f(t)$ if for every time step s_i where M reads/writes a symbol, there is a corresponding time step s'_i where M' reads/writes the same symbol, and $s'_i \le f(s_i)$.

3 Simulation of a Tree Machine

3.1 Upper Bound

It is straightforward to simulate a tree machine with a log-cost RAM in time $O(t \log t)$. In fact, such a simulation is used in Theorem 3.2 to show that a tree machine can be simulated by a unit-cost RAM in real time. However, we can do better than the straightforward simulation for log-cost RAMs.

For simplicity, we consider tree machines with only one worktape, but our results generalize to multiple worktapes. Let T be a tree machine of time complexity t with one worktape W. We show that there is a RAM R that simulates T on-line in time $O((t \log t)/ \log\log t)$.

We first provide a brief description of the simulation. We choose parameters h and u such that $u = 2^{2h+2} - 1$. We specify the values of h and u later. As noted earlier, R has several memories. R maintains in the *main memory* the entire contents of W. The main memory represents W as overlapping subtrees, called *blocks*. R represents the contents of each block W_x in one register r_x of the main memory. When the worktape head is in a particular block W_x, R represents W_x in the *cache memory*. Step-by-step simulation is carried out in the cache, which represents the block W_x in breadth-first order, one cell of W_x per register of the cache.

Because blocks overlap, when the worktape head exits W_x, it is positioned in the middle of some other block W_y. At this time R packs the contents of the cache back into r_x in the main memory and unpacks the contents of r_y into the cache.

The details of the simulation follow.

Let $W[x,s]$ be the complete subtree of W of height s rooted at cell(x). A *block* is any subtree $W_x = W[x, 2h+1]$ such that the depth of cell(x) is a multiple of $h+1$. Since a block has height $2h+1$, it contains $2^{2h+2}-1=u$ cells. Let the *relative location* of a cell within a block be defined in a manner similar to the location of a cell, where the relative location of the root of the block is 1, the relative locations of its children are 2 and 3, and so on.

Call a block W_p the *parent block* of W_x if cell(p) is the ancestor of cell(x) at distance $h+1$ from cell(x). If W_x is the parent block of W_c, then W_c is a *child block* of W_x. Each block has 2^{h+1} child blocks. The topmost block of W, which contains the root of W, is called the *root block.*

Define the *top half* of a block W_x as $W[x,h]$, and define the *bottom half* of W_x as the remaining cells of the block. Note that the top half of the block W_x is part of the bottom half of W_p, its parent block, so that the blocks overlap. Call the portion of W_x shared by W_p (i.e., the subtree $W[x,h]$) the *common subtree* of W_x and W_p.

R precomputes in separate memories two tables, *half* and *translate.* We explain later how R uses these tables. Here we describe their contents and how they are computed. Let *half*(z) (respectively, *translate*(z)) be the register in *half* (respectively, *translate*) at address z.

Half(z) contains $\lfloor z/2 \rfloor$. For $z = 1, \ldots, u/2$, R stores z in *half*$(2z)$ and *half*$(2z+1)$.

For $z = 2^{2h+1}, \ldots, u$, *translate*(z) contains $(z \bmod 2^{h+1}) + 2^{h+1}$. R never refers to any register in *translate* with address less than 2^{2h+1}. *Translate* is computed as follows:

$i := 2^{h+1}$
for $z = 2^{2h+1}$ **to** u **do**
    translate$(z) := i$
    $i := i + 1$
    if $i = 2^{2h+2}$ **then** $i := 2^{h+1}$

We now show how R simulates the tree machine using the cache. Assume the head of T is currently scanning a cell in block W_x. Let *cache*(z) be the register in the cache with address z and let cell(x, z) be the cell in W_x with relative location z. For each $z = 1, \ldots, u$, register *cache*(z) contains the bit in cell(x, z); for example, *cache*(1) contains the contents of cell$(x, 1) =$ cell(x), the root of W_x. Thus R uses u registers of the cache, each register containing one bit.

While the head of T remains in W_x, R keeps track of the head's location with the *cache address register* in the *working memory*, a memory maintained by R for storing information necessary for miscellaneous tasks. If the cache address register contains z, then cell(x, z) is currently being accessed in T.

To simulate a tree machine operation at cell(x, z), R loads the contents (one bit) of *cache*(z) into AC. Once the contents are in AC, R simulates one step of T by storing either 0 or 1 in *cache*(z).

If the head of T moves to a child of cell(x, z), then the new address for the cache address register, as well as the relative location of the new block cell being read, is either $2z$ or $2z+1$. With one or two additions, R computes this new address and places it in the cache address register. When the head of T moves to the parent of cell(x, z), the address of the corresponding cache register is $\lfloor z/2 \rfloor$. Because R has no division operation, it accesses the proper register of table *half* to retrieve the new address in cache.

To describe what happens when the worktape head moves out of the current block, we first show how the blocks are stored in main memory. Main memory is divided into *pages* consisting of $2^{h+1} + 3$ registers each. A page corresponds to a visited block of W. Let *page*(x) be the page representing W_x. Define the address of a page to be the address of the first register in the page. The first register in *page*(x) is the *contents register*. For the page representing the root block, the contents register contains the entire contents of that block. For every other block W_y, the contents register contains the contents of the bottom half of W_y. The contents of cells in a block are kept in breadth-first order; i.e., reading the binary string in the contents register from left to right is equivalent to reading the bottom half of the block it represents in breadth-first order. Initially, all cells of a block contain 0, so all contents registers initially contain 0.

Following the contents register is the *rank register*, containing a number ℓ between 1 and 2^{h+1} indicating that W_x is the ℓ^{th} child of its parent block. The next register is the *parent register*, containing the address of the page representing the parent block of W_x. The next 2^{h+1} registers are the child registers of W_x. The m^{th} child register of *page*(x) contains the address of the page representing the m^{th} child block of W_x or 0 if that child block has not been visited (see Figure 1).

The first page in main memory corresponds to the root block. Blocks are then stored in the order in which they are visited. The *page address register*, a register in working memory, contains the address of the page in main memory corresponding to the currently accessed block.

Let W_x be the currently accessed block and let W_p be the parent block of W_x. When the tree worktape head moves out of W_x so that it is positioned in the middle of a child block W_c, R makes the proper changes to main memory and load the cache from the contents register of *page*(c).

In main memory, R updates the contents registers of *page*(x) and *page*(p). To update *page*(x), R packs the contents of the registers of the cache which correspond to the bottom half of W_x into a single register in working memory (call it the *transfer register*, denoted by *tr*). Packing information in the cache consists of creating from the registers in the cache one binary string that represents the bottom half of a block (in the same format as a main memory register). The pack operation is that used by Katajainen *et al.* (1988). R then copies *tr* into the contents register of *page*(x) via AC (see Figure 2).

Updating *page*(p) consists of changing the bits of its contents register corresponding to the common subtree of W_x and W_p. R first saves the contents of the cache that encode the common subtree of W_x and W_c in a portion of working memory, since this information is needed in the cache as the top half of W_c. R also saves the contents of the cache that encode the common subtree of W_x and W_p. R then loads the contents register of *page(p)* into *tr* and unpacks the contents into the cache. The bits in working memory

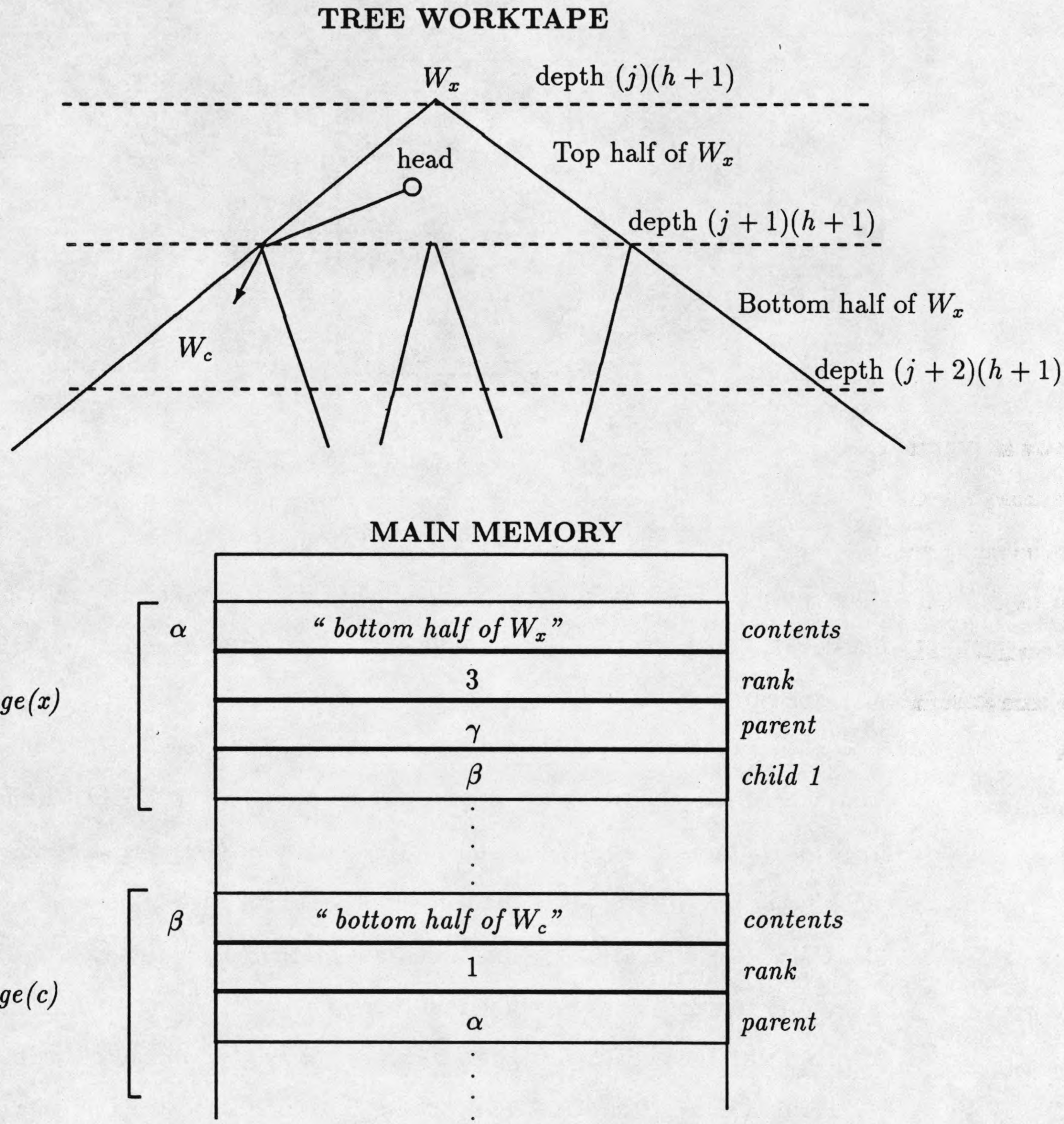

Figure 1: Worktape W (head moves from W_x to W_c)

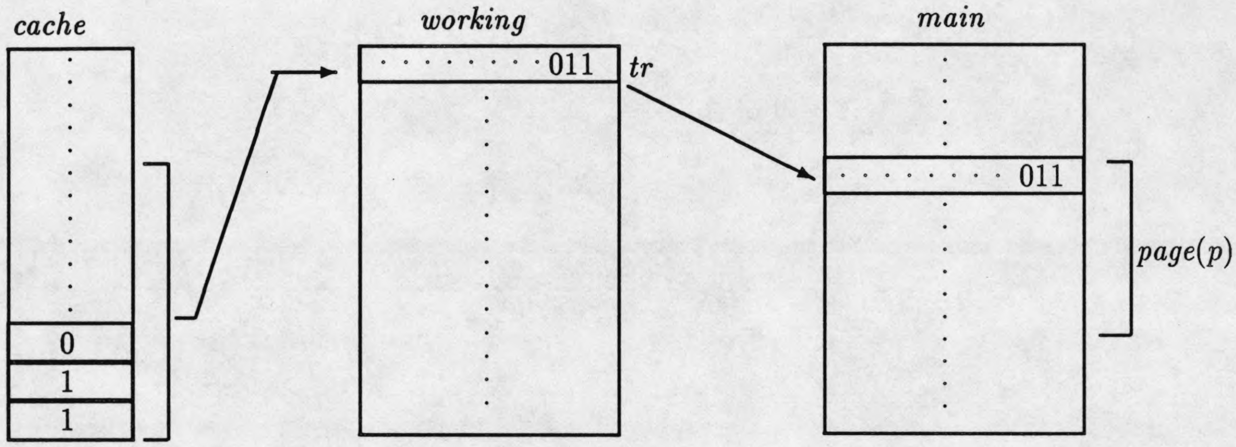

Figure 2: Updating *page*(p) in main memory

corresponding to the common subtree of W_x and W_p are then written into their proper locations in the portion of the cache representing the bottom half of W_p. R then packs the contents of the cache into tr and copies tr into the contents register of *page*(p).

R then determines whether W_c has been visited before by checking the contents of the child register of *page*(x) corresponding to W_c. If the child register contains a valid (i.e., nonzero) address, then R uses that address to access *page*(c). R then loads the contents register of *page*(c) into the cache. This action is similar to the manipulation of *page*(p) discussed above. R loads the contents of the common subtree of W_x and W_c saved in working memory into the registers of the cache representing the top half of the block.

If the child register of *page*(x) contains 0, then R allocates a new page to maintain the information on W_c.

R modifies the page address register to reflect the fact that the worktape head is now scanning block W_c. The address currently in this register is that of $page(x)$. R writes the address of $page(c)$ in main memory to the page address register. R determines from the cache address register the quantity ℓ such that W_c is the ℓ^{th} child of W_x. Then by accessing the ℓ^{th} child register of $page(x)$ in the main memory, R can determine the address of $page(c)$.

To modify the cache address register to reflect the relative location of the head within block W_c, R first translates the relative location of the leaf cell(x, z) in W_x to its relative location in W_c. Since leaf cell(x, z) in W_x is the same as cell$((c, z \bmod 2^{h+1}) + 2^{h+1})$ in W_c, R uses the table *translate* described above. Using one or two additions, R then calculates the relative location in W_c of this cell's left or right child, depending on which branch the worktape head used to exit W_x. R then writes this new relative location into the cache address register.

A similar sequence of operations occurs if the worktape head moves out of a block (and further) into its parent block instead of into a child block. Then R uses the parent register to determine the address of the page representing the parent block, and R uses the rank register to determine the relative location of the worktape head within the parent block.

If R does not know the input size n ahead of time, then we let R adopt an incremental technique of Galil (1976). R begins by assuming that $n = 2$. If the input head reads a third symbol, then R begins again with $n = 4$, but it does not output symbols already printed. In general, R assumes $n = 2^k$ until it reads the $(2^k + 1)$th symbol, at which time R starts over with $n = 2^{k+1}$.

The values of u and h depend on the value of n; therefore u and h are recomputed each time the value of n is doubled.

Let the actual simulation (without the incremental method) run in time $t'(n)$, where $t'(n) \geq n$. It can be shown by induction that the simulation with the incremental method runs in time at most $k't'(n)$, for some constant $k' > 0$.

By evaluating the cost of the simulation on a log-cost RAM, we derive the following result.

Theorem 3.1 *A tree machine running in time $t(n)$ can be simulated on-line by a log-cost RAM running in time $O((t(n)\log t(n))/\log\log t(n))$.*

Proof. Because the blocks have height $2h+1$ and overlap by height $h+1$, each time the worktape head moves out of a block, it is exactly in the middle of another block; i.e., it will take at least $h' = h+1$ steps before it exits this new block. Since the tree machine computation has at most t steps, the work of updating main memory from cache (packing), loading a new block to cache (unpacking), and directly simulating h' steps is performed at most t/h' times.

Updating main memory and loading a new block in cache involve the pack and unpack operations and a constant number of accesses to main memory. Registers in main memory have addresses no larger than $(t/h')(2^{h+1}+3)$. Thus accesses to main memory take time $O(\log t + h)$.

By Lemma 2.1, the time for the pack and unpack operations is $O(u \log u)$. By Lemma 2.2, the time to create the tables necessary for these operations is $O(u2^u)$. The time to compute tables *half* and *translate* is $O(u)$.

Simulating one step of the tree machine consists of a constant number of accesses to cache, taking time $O(\log u)$. Thus simulating h' steps takes time

$O(h' \log u)$.

The total time required for R, then, is

$$(t/h')(O(\log t + h) + O(u \log u) + O(h' \log u)) + O(u2^u).$$

Since $h = O(\log u)$, the total time is

$$O(((t \log t)/\log u) + tu + t \log u + u2^u).$$

Choose h so that $u = (\log t)/\log\log t$. Then the total time for the simulation is $O((t \log t)/\log\log t)$. □

For unit-cost RAMs, we have a much stronger result:

Theorem 3.2 *A tree machine can be simulated by a unit-cost RAM in real-time.*

Proof sketch. We design a unit-cost RAM R simulate tree machine T with worktape W. R has a *contents memory*, a *parent memory*, and several working registers. Let *contents*(x) (respectively, *parent*(x)) be the register with address x in the contents (respectively, parent) memory. *Contents*(x) at address x contains the contents of cell(x) at location x in the worktape of T. If cell(x) is visited by T, then *parent*(x) contains the worktape location of the parent of cell(x). The working registers are used as temporary storage and to keep track of which cell is currently accessed by T.

R simulates one step of T with a constant number of accesses to the two memories and the working registers. For example, if the head moves from cell(x) to a child of cell(x), then R computes location $2x$ of the left child or

$2x+1$ of the right child with one or two additions and stores x in *parent*$(2x)$ or *parent*$(2x+1)$. Thus to simulate t steps of T takes $O(t)$ time on T. □

An immediate consequence of Loui's upper bound on the simulation of a tree machine by a multidimensional TM is the following:

Theorem 3.3 (Loui, 1983) *A log-cost RAM running in time $t(n)$ can be simulated on-line by a multihead d-dimensional Turing machine running in time $O(t(n)^{1+1/d}/\log t(n))$.*

Using our simulation of a tree machine by a log-cost RAM, we obtain a nonlinear lower bound for simulating a RAM by a multidimensional Turing machine:

Corollary 3.4 *There is a log-cost RAM R running in time $t(n)$ such that for any multihead d-dimensional Turing machine S, S simulates R on-line in time $\Omega((t(n)^{1+1/d}(\log\log t(n))^{1+1/d})/(\log t(n))^{2+1/d})$.*

Proof. Let T be the tree machine described in the lower bound proof of Loui(1983). Let R be the RAM that uses the method in the proof of Theorem 3.1 to simulate tree machine T. T runs in real time, so by Theorem 3.1, R runs in time $t(n) = O((n\log n)/\log\log n)$. Now assume there is a d-dimensional Turing machine that simulates R on-line in time $o((t^{1+1/d}(\log\log t)^{1+1/d})/(\log t)^{2+1/d})$. We thus have an on-line simulation of tree machine T running in time n by a d-dimensional Turing machine running in time $o(n^{1+1/d}/\log n)$. But we know from Loui (1983) that the lower bound on such a simulation is $\Omega(n^{1+1/d}/\log n)$; hence we have a contradiction. □

3.2 Lower Bound

We now show that the time bound of Theorem 3.1 is optimal within a constant factor. We begin with an overview of Kolmogorov complexity, which we use to prove the lower bound.

For strings σ, τ in $\{0,1\}^*$, let $K(\sigma)$ be the *Kolmogorov complexity* of σ with respect to a universal Turing machine U. Define $K(\sigma)$ to be the length of β where β is the shortest binary string such that $U(\beta)$ equals σ. Informally, $K(\sigma)$ is the length of the shortest binary description of σ.

We say a string σ is *incompressible* if $K(\sigma) \geq |\sigma|$. Note that for all n there are 2^n binary strings of length n, but there are only $2^n - 1$ strings of length less than n. Thus for all n, there is at least one incompressible string of length n.

A useful concept in Kolmogorov complexity is the *self-delimiting string.* For natural number n, let $bin(n)$ be the binary representation of n without leading 0's. For binary string w, let $\overline{w}$ be the string resulting from placing a 0 between each pair of adjacent bits in w and adding a 1 to the end. Thus $\overline{110} = 101001$. We call the string $\overline{bin(|w|)}w$ the *self-delimiting version* of w. The self-delimiting version of w has length $|w| + 2\lceil \log(|w|+1) \rceil$. When we concatenate several binary string segments of differing lengths, we can use self-delimiting versions of the strings so that we can determine where one string ends and the next string begins with little additional cost in the length of the concatenated string. Note that in such a concatenation it is not necessary to use a self-delimiting version of the last string segment.

Kolmogorov complexity has recently gained popularity as a method for proving lower bounds. Li and Vitanyi (1988) provide a thorough summary of lower bound (and other complexity-related) results obtained using Kolmogorov complexity.

Theorem 3.5 *There is a tree machine T running in time n such that for any log-cost RAM R, R requires time $t(n) = \Omega((n \log n)/\log\log n)$ to simulate T on-line.*

Proof. For simplicity, we omit floors and ceilings in this proof.

Tree machine T has one tree worktape and operates in real time. T's input alphabet is a set of *commands* of the form $\langle e, \psi\rangle$, where $e \in \{0, 1, ?\}$ and ψ indicates whether the worktape head moves to a child or parent of the current cell or remains at the current cell. Suppose T is in a configuration in which the cell x at which the worktape head is located contains e'. On input $\langle e, \psi\rangle$, machine T writes e' on its output tape, and the worktape head writes e on cell x if $e \in \{0, 1\}$, but it writes e' (the current contents of x) on x if $e =?$. At the end of the step the worktape head moves according to ψ. For every n that is a sufficiently large power of 2, we construct a series of n tree commands for which R requires time $\Omega((n \log n)/\log\log n)$. As in (Loui, 1983), the string of tree commands is divided into a *filling part* of length $n/2$ and a *query part* of length $n/2$.

Let W be the worktape of T, and let x_0 be the root of W. Let $d = \log(n/8)$. Denote the complete subtree of W of height d whose root is x_0 by W_d. Let $N = n/8$. We consider the complexity of the simulation in terms of N.

We fill W_d with an incompressible string τ of length $2N-1$ such that τ can be retrieved by a depth-first traversal of W_d. This is the filling part, for which T takes time $4N$ $(= n/2)$).

The query part consists of a series of *questions*. A *question* is a string of $2\log N$ tree commands that causes the worktape head to move from the root x_0 of the tree worktape to a cell at depth d and back to x_0 without changing the contents of the worktape. As the head visits each cell during a question, T outputs the contents of that cell. T processes $N/(4\log N)$ questions $Q_1, Q_2, \ldots$ during the query part. We show that after each question Q_j, there is a question Q_{j+1} such that R takes time $\Omega((\log^2 N)/\log\log N)$ to process Q_{j+1}, and Theorem 3.5 follows.

Assume that R has just processed question Q_j. Let $P(N)$ be the maximum time necessary to process any possible next question. We show that some next question takes time $\Omega((\log^2 N)/\log P)$. Consequently, by definition, $P = \Omega((\log^2 N)/\log P)$; thus $P = \Omega((\log^2 N)/\log\log N)$.

We first determine the total time $\hat{t}$ required for R to process all possible next questions.

Divide worktape W into $S = (\log N)/(2\log P)$ *sections*, each of height $2\log P$. For $s = 0, 1, \ldots, S-1$, there are P^{2s+2} exit points (*bottom cells*) in section s. We refer to any initial segment of a question as a *partial question* and the portion of the question that is processed while the worktape head is in one section as a *subquestion* (see Figure 3). To compute $\hat{t}$, we compute for $s = 0, 1, \ldots, S-1$ the total time $\hat{t}_s$ required for R to process all possible subquestions in section s. Since the depth of W_d is $\log N$, there are N possible next questions. Each of the P^{2s+2} bottom cells of section s is visited during

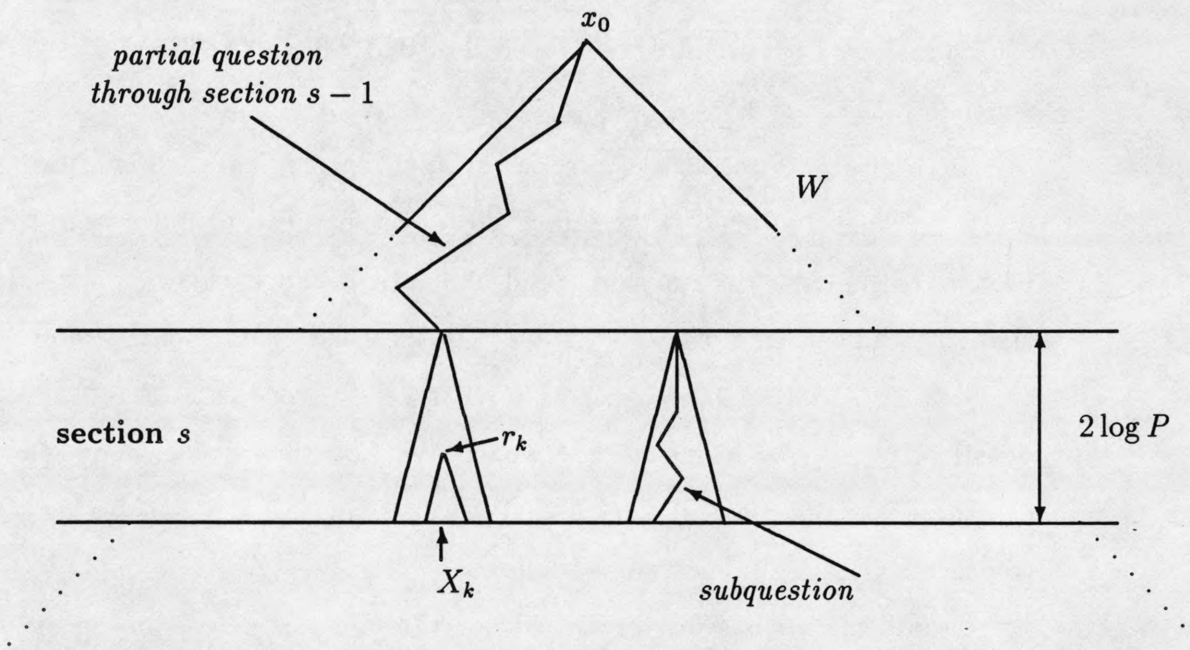

Figure 3: Processing section s of worktape W

N/P^{2s+2} of these questions.

Let σ_s be the string defined by the contents of the bottom cells of section s, from left to right; clearly, $|\sigma_s| = P^{2s+2}$.

Lemma 3.6 *The string σ_s is incompressible up to a term of $O(s \log P)$; i.e., $K(\sigma_s) \geq |\sigma_s| - O(s \log P)$.*

Proof. The incompressible string τ, which gives the contents of W, can be specified by a string composed of the following segments:

1. a self-delimiting string encoding this discussion ($O(1)$ bits)

2. a self-delimiting version of a binary string of length $K(\sigma_s)$ that specifies σ_s ($K(\sigma_s) + O(s \log P)$ bits)

3. self-delimiting versions of the values of s and P ($O(\log s) + O(\log P)$ bits)

4. a string specifying the bits in τ but not in σ_s ($2N - 1 - P^{2s+2}$ bits).

Thus $K(\tau) \leq K(\sigma_s) + (2N - 1 - P^{2s+2}) + O(s \log P)$. But $K(\tau) \geq 2N - 1$; therefore, $K(\sigma_s) \geq P^{2s+2} - O(s \log P)$. □ **Lemma 3.6**

Lemma 3.7 *If* $\ell \geq 1$ *then* $\sum_{i=1}^{\ell} \log i \geq (1/2)\ell \log \ell$.

Proof. For all i such that $1 \leq i \leq \ell$, evidently $(i-1)(\ell - i) \geq 0$; hence $i(\ell - i + 1) \geq \ell$. Consequently

$$\begin{aligned} \sum_{i=1}^{\ell} \log i &= (1/2) \sum_{i=1}^{\ell} (\log i + \log(\ell - i + 1)) \\ &= (1/2) \sum_{i=1}^{\ell} \log(i(\ell - i + 1)) \\ &\geq (1/2) \sum_{i=1}^{\ell} \log \ell \\ &= (1/2)\ell \log \ell. \end{aligned}$$

□ **Lemma 3.7**

Lemma 3.8 *For* $s = 1, 2, \ldots, S-1$, *the maximum number of registers accessed during the processing of all partial questions through section* $s - 1$ *is* $4P^{2s+1}/\log P$.

Proof. Let $C = 4P/\log P$. By Lemma 3.7, for P sufficiently large, $\sum_{i=1}^{C} \log i \geq P$. The processing of each partial question through section $s-1$ could involve no more than C registers; otherwise, because of the total cost of addresses of registers, R would exceed time P for some next question. There are P^{2s} different partial questions possible through section $s-1$, so there are no more than $4P^{2s+1}/\log P$ registers accessed for all possible partial questions. □ **Lemma 3.8**

Let us consider a particular section s. Let $r_1, r_2, \ldots, r_m$ be the registers, in order of increasing address, used to process tree commands in section s. The address of r_i is at least i. For $1 \leq i \leq m$, let X_i be the set of bottom cells x of section s such that r_i is accessed while the worktape head is visiting some cell y in section s, and either y is an ancestor of x or $y = x$ (see Figure 3). We say that r_i *operates* on the bottom cells in X_i.

To compute a lower bound on $\hat{t}_s$, we assess the contribution to $\hat{t}_s$ of accessing register r_i. For $1 \leq i \leq m$, the total access time for register r_i in section s is at least the product of $\log i$ (since the address of r_i is at least i), $|X_i|$ (the number of bottom cells that r_i operates on), and N/P^{2s+2} (the number of questions during which one of these bottom cells is visited). Totalling the time incurred by access to each register yields:

$$\hat{t}_s \geq \sum_{i=1}^{m} (\log i)|X_i|(N/P^{2s+2}). \tag{1}$$

Using Lemma 3.10 below, we can determine a lower bound for $\hat{t}_s$, but we first introduce the following technical lemma.

Lemma 3.9 (Loui, 1984a [Section 4]) *Let J and M be integers such that $M \geq J$. A sorted J-member subset of $\{0, \ldots, M\}$ can be represented with no more than $2J \log(M/J) + 4J + 2$ bits.*

Let $h = (1/7)P^{2s+1}$.

Lemma 3.10 $\sum_{i=h}^{m} |X_i| \geq (1/23)P^{2s+2}$.

Proof. Assume that the conclusion is false. Then $r_1, \ldots, r_{h-1}$ operate on at least $(22/23)P^{2s+2}$ bottom cells in section s. We can specify the string σ_s as follows: we obtain the bits of $X_h, \ldots, X_m$ explicitly. We obtain the other bits of σ_s by simulating R on each partial question to a bottom cell of section s not in $\bigcup_{k=h}^{m} X_k$. On each such partial question, R uses only registers $r_1, \ldots, r_{h-1}$ and registers accessed in sections $1, \ldots, s-1$. Thus σ_s can be specified with a string composed of the following segments:

1. a self-delimiting string encoding the program of R and this discussion ($O(1)$ bits)

2. self-delimiting versions of the addresses and initial contents of registers accessed in sections $1, \ldots, s-1$ (at most $8P^{2s+2}/\log P + O(s \log P)$ bits – by Lemma 3.8, at most $4P^{2s+2}/\log P$ registers are required, and for each register, the contents and the address could each require P bits.)

3. self-delimiting versions of the addresses and initial contents of $r_1, \ldots, r_{h-1}$ ($(2/7)P^{2s+2} + O(s \log P)$ bits)

4. a string specifying positions of cells in X_k for $k \geq h$ (we use Lemma 3.9 with $J = (1/23)P^{2s+2}$ and $M = P^{2s+2}$; this requires at most $(14/23)P^{2s+2}$ bits. The encoding used to achieve Lemma 3.9 is such that the beginning and end of this string can easily be determined.)

5. a string specifying the contents of cells in X_k for $k \geq h$ (at most $(1/23)P^{2s+2}$ bits).

This means that the number of bits needed to specify σ_s is at most $(151/161)P^{2s+2} + O(P^{2s+2}/\log P) < P^{2s+2} - O(s \log P)$ for sufficiently large P. Thus we have a contradiction of Lemma 3.6. □ **Lemma 3.10**

Thus we have:

$$\begin{aligned}
\hat{t}_s &\geq \sum_{i=1}^{m}((\log i)|X_i|(N/P^{2s+2}) && \text{(Inequality 1)}\\
&\geq \sum_{i=h}^{m}((\log i)|X_i|(N/P^{2s+2}))\\
&\geq (N/P^{2s+2})(\log h)\sum_{i=h}^{m}|X_i|\\
&\geq (N/P^{2s+2})(\log h)(1/23)P^{2s+2} && \text{(Lemma 3.10)}\\
&\geq (1/23)N((2s+1)\log P - \log 7) && \text{(definition of } h\text{)}\\
&\geq (1/23)Ns\log P.
\end{aligned}$$

Now sum $\hat{t}_s$ over all s to compute a lower bound for $\hat{t}$, the total time required for R to process all possible next questions:

$$\begin{aligned}
\hat{t} &= \sum_{s=0}^{S-1}\hat{t}_s\\
&\geq \sum_{s=0}^{S-1}((1/23)Ns\log P)
\end{aligned}$$

$$\geq (1/23)N(\log P)((\log^2 N)/(4\log^2 P) - O((\log N)/\log P))$$
$$\geq (1/92)((N\log^2 N)/\log P - O(\log N)).$$

Since there are N questions, we divide $\hat{t}$ by N to derive the average time needed by R to process the next question, $\Omega((\log^2 N)/\log P)$. Some next question must require time greater than or equal to this average time. Since P is the maximum time for some next question, $P \geq \Omega((\log^2 N)/\log P)$; hence, $P = \Omega((\log^2 N)/\log\log N)$.

Thus for each question Q_j, we can choose a next question Q_{j+1} that takes time $\Omega((\log^2 N)/\log\log N)$. Since the query part has $N/(2\log N)$ questions, our choice of questions means that the query part takes time $t = (N/(2\log N))\Omega((\log^2 N)/\log\log N)) = \Omega((N\log N)/\log\log N)$. The entire simulation takes at least time t. Since $N = n/8$, the lower bound holds for n as well. □ **Theorem 3.5**

Because the lower bound proof considers only the time involved in accessing registers, the lower bound holds for RAMs with more powerful instructions, such as boolean operations or multiplication.

4 Simulation of a Multidimensional Turing Machine

By composing our simulation in subsection 3.1 of a tree machine by a log-cost RAM with Reischuk's (1982) simulation of a d-dimensional Turing machine by a tree machine, we obtain an on-line simulation of a d-dimensional

Turing machine of time complexity t by a log-cost RAM running in time $O((5^{d\log^* t} t \log t)/\log\log t)$. But we can improve this upper bound with a direct simulation.

Theorem 4.1 *A d-dimensional Turing machine running in time $t(n)$ can be simulated on-line by a log-cost RAM running in time* $O(t(n)(\log t(n))^{1-1/d}(\log\log t(n))^{1/d})$.

Proof sketch. We design a log-cost RAM R that simulates d-dimensional Turing machine M. For simplicity, assume M has one worktape; our results generalize to d-dimensional Turing machines with more than one worktape. Let $s = ((\log t)/\log\log t)^{1/d}$. Partition the worktape of M into d-dimensional cubes (call them *boxes*) with side length s. Let *corner*(i) be the cell in box i with the coordinates whose components are the smallest.

For box i, if *corner*$(i) = (i_1, i_2, \ldots, i_d)$, let *index*$(i) = i_d t^{d-1} + i_{d-1} t^{d-2} + \ldots + i_1$. R stores the contents of box i in the register in *main memory* with address *index*(i). Step-by-step simulation is carried out in the *cache.* R conducts the simulation in t/s phases, each of s steps of M. For each phase: R unpacks the contents of 3^d boxes that are within distance s of the worktape head (the head remains within these boxes during the phase); R simulates M for s steps; and R packs the contents of the cache back to main memory. Using precomputed values of $t, t^2, \ldots, t^{d-1}$, R quickly computes *index*(i') from *index*(i) when box i' is adjacent to box i. For each phase, R takes time $O(\log t)$ to access main memory, $O(\log t)$ to compute the address of registers in main memory representing the new blocks needed in cache, $O(s\log s)$ to simulate s steps in the cache, and $O(s^d \log s)$ to pack and unpack the appropriate registers (Lemma 2.1). Thus the total time for the simulation is:

$$
\begin{aligned}
(t/s)(O(\log t) + O(s\log s) + O(s^d \log s)) & \\
&= O(((t\log t)/s) + ts^{d-1}\log s) \\
&= O(t(\log t)^{1-1/d}(\log\log t)^{1/d}).\Box
\end{aligned}
$$

Once again, the result for unit-cost RAMs is much stronger:

Theorem 4.2 *A multidimensional Turing machine can be simulated by a unit-cost RAM in real-time.*

Proof. Schönhage (1980) showed that a unit-cost successor RAM can simulate a multidimensional Turing machine in real-time. It follows that a unit-cost RAM with addition and subtraction can simulate a multidimensional Turing machine in real-time as well. □

5 Conclusions

Because the log-cost RAM is considered a "standard" among models of computation, it is important to determine its relationships to other models. Here we have shown an optimal on-line relationship between log-cost RAMs and tree machines. We have constructed an analogous efficient simulation of multidimensional Turing machines by log-cost RAMs. We hope that this work will lead to further study of relationships between other models of computation.

Some further areas of research include:

1. finding an off-line simulation that is faster than our on-line simulation of a tree machine by a log-cost RAM.
2. finding an optimal simulation of a pointer machine (Schönhage, 1980) by a log-cost RAM.
3. finding an optimal simulation of a unit-cost RAM by a log-cost RAM.

References

[Aggarwal *et al.*, 1987] Alok Aggarwal, Bowen Alpern, Ashok K. Chandra, and Marc Snir. A model for hierarchical memory. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 305–314, 1987.

[Aho *et al.*, 1974] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

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

[Galil, 1976] Zvi Galil. Two fast simulations which imply some fast string matching and palindrome-recognition algorithms. *Information Processing Letters*, 4(4):85–87, 1976.

[Katajainen *et al.*, 1988] Jyrki Katajainen, Jan Van Leeuwen, and Martti Penttonen. Fast simulation of Turing machines by random access machines. *SIAM J. Comput.*, 17:77–88, February 1988.

[Li and Vitanyi, 1988] Ming Li and Paul M. B. Vitanyi. Two decades of applied Kolmogorov complexity. 1988. To appear in *Handbook of Theoretical Computer Science* (J. van Leeuwen, Managing Editor), North-Holland. Preliminary version in *Proc. 3rd IEEE Structure in Complexity Theory Conf.*, pages 80–101, 1988.

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

[Loui, 1984a] Michael C. Loui. The complexity of sorting on distributed systems. *Information and Control*, 60:70–85, 1984.

[Loui, 1984b] Michael C. Loui. Minimizing access pointers into trees and arrays. *J. Comput. System Sci.*, 28(3):359–378, 1984.

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

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

[Schönhage, 1980] Arnold Schönhage. Storage modification machines. *SIAM J. Comput.*, 9(3):490–508, August 1980.

[Slot and van Emde Boas, 1988] Cees Slot and Peter van Emde Boas. The problem of space invariance for sequential machines. *Inform. and Comput.*, 77:93–122, 1988.