On the Difference Between Turing Machine Time and Random-Access Machine Time

(extended abstract)

Kenneth W. Regan*
State University of New York at Buffalo

Abstract

We introduce a model of computation called the *Block Move* (BM) model. The BM extends the *Block Transfer* (BT) model of Aggarwal, Chandra, and Snir [1], who studied time complexity under various *memory access cost functions* ranging from $\mu_1(a) := a$ to $\mu_{\log}(a) := \lceil \log_2 a \rceil$. We show that up to factors of $\log t$ in the total running time t, BMs under μ_1 are equivalent to multitape Turing machines, and BMs under $\mu_{\log}$ are equivalent to log-cost RAMs. We also prove that for any well-behaved μ the BM classes DμTIME$[t(n)]$ form a tight deterministic time hierarchy. Whether there is any hierarchy at all when μ rather than t varies is tied to long-standing open problems of determinism vs. nondeterminism.

Keywords Computational complexity, theory of computation, Turing machines, random-access machines, models, simulation, finite automata.

1. Introduction

It is widely believed that random-access machines (RAMs) are more efficient and powerful than multitape Turing machines (TMs). However, no conclusive separation has been proven. The standard RAM model ("ordinary RAM" in [21]) has addition and subtraction as basic operations. Inputs $x \in \{0,1\}^n$ are initially placed in the first n registers, one bit per register. The best known result is that for time bounds $t(n)$ which are $\Omega[n \log n]$ and time-constructible by RAMs, a TM running in time $t(n)$ can be simulated by a RAM in time $t(n)/\log t(n)$ [7]. It follows that the TM time class DTIME$[t(n)]$ is properly contained in the RAM time class RAM-TIME$[t(n)]$. However, the gap of a factor of only $\log t(n)$ is not felt to represent the true power of a RAM over a TM, and even here no separation is known in the case $t(n) = O(n)$.

Moreover, the result of [7] is for the unit-cost RAM, and is not known to hold under the more reasonable *log-cost criterion* of Cook and Reckhow [3], which charges e.g. $\lceil \log i \rceil + \lceil \log a \rceil$ for a RAM step which loads a stored integer i from register a. The *RAM-TM* model of [5, 2], which uses fixed-size registers and provides a separate *address tape* for accessing them, and the *tree computer*, namely a TM with multiple binary tree-structured tapes (see [15, 11]), are each equivalent to the log-cost RAM up to a factor of $O(\log t)$ in the running time t. Possibly more constrained is the "RAM with polynomially compact memory" of Grandjean and Robson [4]. Others have considered TMs with planar or d-dimensional tapes (see [17, 10]) and TMs whose heads can "jump" to marked cells [22]. The theorem of [14] that DLIN is properly contained in nondeterministic TM linear time (NLIN) is not known to carry over to any of these models. Abbreviating $t(n)$ to t, we write RAM-TIME$^{\log}[t]$ for time under the log-cost RAM, TC$[t]$ for tree computers, and DTIME$^d[t]$ for TMs with d-dimensional tapes. The best known simulations are:

Proposition 1.1. *For any time function $t(n) \geq n$,*

(a) RAM-TIME$^{\log}[t] \subseteq$ TC$[O(t)] \subseteq$ RAM-TM-TIME$[O(t)]$ [15].

(b) TC$[t] \subseteq$ RAM-TIME$^{\log}[O(t \log t / \log\log t]$ [13].

(c) DTIME$[t] \subseteq$ RAM-TM-TIME$[t]$.[1]

(d) RAM-TIME$^{\log}[t] \subseteq$ DTIME$[t^2]$ [3].[2]

(e) *For* $d \geq 2$, DTIME$^d[t] \subseteq$ RAM-TIME$^{\log}$ $[t(\log t)^{1-(1/d)}/(\log\log t)^{1/d}]$ [12].[3]

(f) *For* $d \geq 2$, RAM-TIME$^{\log}[t] \subseteq$ DTIME^d $[t^{1+(1/d)}/\log t]$ [11].

*The author was supported in part by NSF Research Initiation Award CCR-9011248. Address: Computer Science Department, 226 Bell Hall, UB North Campus, Buffalo, NY 14260-2000. Email: *regan@cs.buffalo.edu*, tel.: (716) 645–3189, fax: (716) 645–3464.

[1] See also [9] for simulations of TMs by log-cost RAMs.

[2] Improved to RAM-TIME$^{\log} \subseteq$ DTIME$[t^2/\log t]$ by combining [17] and [15]. For the ordinary RAM the analogous result is RAM-TIME$[t] \subseteq$ DTIME$[t^3]$ [3].

[3] Reischuk [19] gives a bound closer to, but not quite, linear for simulation by tree machines. We do not have a better bound than Reischuk's for RAM-TMs.

The last two bounds are optimal for on-line simulations [12, 13], and the others are thought to be close. However, no separations have been proved for the language classes, not even DTIME$[O(n)]$ vs. RAM-TM-TIME$[O(n)]$. If we ignore the factors involving $\log t$ and $\log\log t$, we can draw this general picture: The gap between the RAM and the TM appears to be a matter of t vs. t^2. Giving the TMs d-dimensional tapes narrows this gap to t vs. $t^{1+(1/d)}$.

2. The Block Move Model

This paper develops a new way of gauging the difference between a RAM and a TM, one which directly addresses the tradeoff between "serial" and "random" access to memory. The BT model of Aggarwal, Chandra, and Snir [1] added to the standard RAM special *block transfer* instructions of the form

$$copy\ [a_1 \ldots b_1]\ into\ [a_2 \ldots b_2].$$

This is *valid* so long as the two intervals of consecutive registers are disjoint and have the same size m. The cost for the move is $m + \mu(c)$ time units, where $c = \max\{\, b_1, b_2 \,\}$, and $\mu : \mathbf{N} \rightarrow \mathbf{N}$ is a parameter which grades the delay in access time as one moves out in the *memory hierarchy*. The memory cost functions μ studied in [1] included $\mu_{\log}(a) := \lceil \log_2 a \rceil$ and for $d \geq 1$, $\mu_d(a) := \lceil a^{1/d} \rceil$. The authors of [1] prove that every function which depends on all of the input requires more than linear time on the BT, viz. $\Omega[n \log n]$ for $\mu = \mu_1$, $\Omega[n \log\log n]$ for $\mu = \mu_d$ with $d > 1$, and $\Omega[n \log^* n]$ for $\mu = \mu_{\log}$. They prove slightly higher tight bounds for FFT, matrix transpose, and sorting.

The *Block Move* (BM) model introduced here is based on a RAM-TM, and extends the BT by allowing any finite-state transducer S, not just *copy*, to be applied to the data in a *block move* of the form

$$S\ [a_1 \ldots b_1]\ into\ [a_2 \ldots b_2].$$

If x is the string formed by the symbols in cells a_1 through b_1, this means that $S(x)$ is written to the tape beginning at cell a_2 in the direction of b_2, with the proviso that a blank B appearing in the output $S(x)$ leaves the previous content of the target cell unchanged. The BM also allows $b_1 < a_1$ and/or $b_2 < a_2$. The block move is *valid* if the two intervals are disjoint, and *strictly valid* if also $S(x)$ exactly fills $[a_2 \ldots b_2]$. For S we refer to the *generalized sequential machine* (GSM) described in [8], which generalizes the Mealy machine by allowing 0,1, or 2 or more characters to be output in a single transition. The blank B is both a readable and a writable character.

Syntactically, the BM model in its *full form* has one *main tape*, four *address tapes* labeled a_1, b_1, a_2, b_2, and a finite control comprised of *RAM-TM states* and *GSM states*. In a RAM-TM state, the current main-tape address a is given by the content of tape a_1. A RAM-TM state performs *work* 1 and incurs a *memory-access charge* of $\max\{\, \mu(a), \mu(b) \,\}$, where b is the rightmost extent of an address tape head. In a GSM state S, the address tapes give the block boundaries for the block move $S\ [a_1 \ldots b_1]\ into\ [a_2 \ldots b_2]$ as described above, and control passes to some RAM-TM state. The *work* is $|x| + |S(x)|$, and the *memory-access charge* is $\mu(c)$, where $c = \max\{\, a_1, b_1, a_2, b_2 \,\}$. The μ-*time* of a computation is the sum of all the work plus memory-access charges. The BT lower bounds cited above do not affect the BM because other finite transductions can glean information about the input in a way that *copy* cannot, and even under the highest cost function μ_1 that we consider, many interesting nonregular languages and functions are computable in linear time.

Even if the strict validity condition is violated, the above description of a block move is still cogent if all input cells are read before any outputs overwrite them, and the output string is allowed to underflow or overflow the designated block boundary b_2 by any amount. Hence we can define variants in which this condition is not in force. The full form can be further *augmented* by allowing any number $k \geq 1$ of main tapes, each with its own four address tapes, and by having block moves which involve all k tapes at once, using *k-input, k-output* GSMs (for more detail, see [18]). We also define a notationally compact form which does not specify the read-block boundary b_1 in advance. For this we modify GSMs so that they can stop upon reading certain symbols (including B), and call the result a *generalized sequential transducer* (GST).

Definition 2.1. The *reduced form* of a BM M consists of a single GST S whose terminal states q have labels $l_1(q) \in \{\, a, \lfloor a/2 \rfloor, 2a, 2a+1, \text{HALT} \,\}$ and $l_2(q) \in \{\, Ra, La, 0R, 0L \,\}$. M has a single tape which initially contains the input x in cells $0, \ldots, n-1$ ($n := |x|$); the *initial address* is $a = 0$ and the *initial mode* is Ra. The computation of $M(x)$ is a sequence of *passes*. In a pass with *current address* a and *mode* Ra, S begins operation from cell a moving rightward, and the output y is written rightward from cell 0, according to the convention about B. If and when S exits while reading some cell b and in some terminal state q, the labels $l_1(q)$ and $l_2(q)$ determine the address and mode for the next pass. If the mode is La, S reads from cell a moving leftward, with the output written rightward from cell 0 as before. If the mode is $0R$ (resp.

$0L$), S reads from cell 0 moving rightward, and the output is written beginning at cell a and proceeding rightward (resp. leftward). In each case the *validity condition* is that $[0 \dots c]$ and $[a \dots b]$ be disjoint, the *work* is $1 + |b - a| + c$, and the *memory-access charge* is $\max\{\,\mu(a), \mu(b), \mu(c)\,\}$.

Robustness Theorem. [18]. For any memory cost function μ_d, with $d \geq 1$ and rational, the reduced, full, and augmented forms of the BM, with or without the (strict) validity condition, all simulate each other up to constant factors in μ_d-time.

This theorem gives the BM time classes under μ_d, which we denote by $\mathrm{D}\mu_d\mathrm{TIME}[t(n)]$, a good degree of machine independence. In particular, we say a language $L \in \mathrm{D}\mu_d\mathrm{TIME}[O(n)]$ has *dimension d*. Under $\mu_{\log}$, the forms of the BM are equivalent up to a factor of $\log t(n)$, and we define $\mathrm{D}\mu_{\log}\mathrm{TIME}[t(n)]$ with reference to the reduced form. The robustness theorem also lets us describe BM algorithms in the same informal manner that is familiar for RAM algorithms.

3. Results

Our first result shows that the *Hennie-Stearns simulation* [6] can be carried out efficiently by a BM.

Lemma 3.1. *For any time function t,* $\mathrm{DTIME}[t(n)] \subseteq \mathrm{D}\mu_1\mathrm{TIME}[t(n)\log t(n)]$.

Proof Sketch. Let T be a multitape TM with alphabet Γ which runs in time $t(n)$, and consider the two-tape TM T' in the Hennie-Stearns theorem (see [6, 8, 21]). The k-many tapes of T are simulated on $2k$-many tracks of the first tape so that all tape heads of T are maintained on cell 0 of each track. T' uses its second tape only to transport blocks of the form $[2^{j-1} \dots 2^j - 1]$ from one part of the first tape to another. The functions used in these moves are homomorphisms between the alphabets Γ^{2k} and Γ^k which pack and unpack characters in blocks, and so a BM M simulating T' can compute each move in a single GST pass. By the structure of the blocks, any pass which incurs a memory-access charge of $\mu_1(2^j) = 2^j$ simulates at least 2^{j-1} moves of T'. Hence the work and the μ_1 charges to M are both $O(t(n)\log t(n))$. □

Lemma 3.2. *Let M be a BM which runs in μ-time $t(n)$ and space $s(n)$. Then we can find a DTM T which simulates M in time $O[t(n)s(n)/\mu(s(n))]$.*

Proof. We may assume that M is in the reduced form with sole GST S. One tape of T mimics the main tape of M, while the other is used as temporary storage for the output in passes. Let s stand for $s(n)$. Consider a move by M which changes the current address a to $\lfloor a/2 \rfloor$. T can find this cell in at most $3a/2$ steps by keeping count with its second tape. Since $s/a \geq 1$, the property $\mu(Na) \leq N\mu(a)$ with $N := s/a$ gives $a/\mu(a) \leq s/\mu(s)$. Hence the ratio of the time used by T to the μ-time charged to M stays $O[s/\mu(s)]$. The same holds for the moves $a := 2a$ and $a := 2a + 1$. T simulates a move of type Ra or La by writing $S[a \dots b]$ to its second tape, moving to cell 0, copying $S[a \dots b]$ over the first tape, and moving back to cell a. Both this and the analogous simulation of an $0R$ or $0L$ move take time $O(a + b)$, and the ratio of this to the memory access charges $\mu(a) + \mu(b)$, not even counting the number of bits processed by M, keeps the running total of the time logged by T below $t(n)s/\mu(s)$. □

Theorem 3.3. *For any time function $t(n) \geq n$,*

(a) $\mathrm{D}\mu_1\mathrm{TIME}[t] \subseteq \mathrm{DTIME}[t] \subseteq \mathrm{D}\mu_1\mathrm{TIME}[t\log t]$.

(b) $\mathrm{D}\mu_{\log}\mathrm{TIME}[t] \subseteq \text{RAM-TM-TIME}[t] \subseteq \mathrm{D}\mu_{\log}\mathrm{TIME}[t\log t]$.

(c) $\mathrm{D}\mu_{\log}\mathrm{TIME}[t] \subseteq \mathrm{DTIME}[t^2/\log t]$.

(d) For any $d > 1$, $\mathrm{D}\mu_d\mathrm{TIME}[t] \subseteq \mathrm{DTIME}^d[t]$.

Proof. Parts (a)-(c) follow from Lemma 3.2 and previously cited simulations. For (d), let T have one d-dimensional tape on which it winds the main tape of M in a spiral about the origin, and one linear tape on which it buffers outputs by the GST S of M. In any pass which incurs a μ_d charge of $a^{1/d}$, T can walk between cell a and the origin within $a^{1/d}$ steps and complete the move. □

Thus we say that the range of different memory-access cost functions μ spans the difference between a TM and a RAM. We have not been able to obtain a good converse to (d); the combinatorial problem is that a TM with d-dimensional tapes may change its head direction often, and the methods of [16] have more than the $\log t(n)$ overhead of the one-dimensional case [6].

For *quasilinear time*, i.e. time $qlin = n(\log n)^{O(1)}$, the extra $\log n$ factors in the above simulations do not matter. Following Schnorr [20], we write DQL and NQL for the TM time classes $\mathrm{DTIME}[qlin]$ and $\mathrm{NTIME}[qlin]$. The robustness theorem of Gurevich and Shelah [5] implies that $\text{RAM-TIME}^{\log}[qlin] = \mathrm{TC}[qlin] = \text{RAM-TM-TIME}[qlin]$ (we call this DNLT). Perhaps more surprising, [5] proved that the nondeterministic counterparts of these classes are equal not only to each other, but also to NQL. We define *NBMs* by allowing finite-state transducers in block moves to be nondeterministic, (cf. *GSM mappings* in [8]), and let NμTIME$[t(n)]$ stands for the languages accepted by NBMs in μ-time $t(n)$. The above immediately yield:

Corollary 3.4.

(a) Dμ_1TIME$[qlin]$ = DQL.

(b) D$\mu_{\log}$TIME$[qlin]$ = DNLT.

(c) For all $\mu \leq \mu_1$, NμTIME$[qlin]$ = NQL. □

Thus obtaining any separation by more than factors of $\log n$ of the classes DμTIME$[O(n)]$ as μ varies from μ_1 through μ_d to $\mu_{\log}$ runs into the problem of whether DQL $\neq$ NQL, which seems as hard as P vs. NP. Even separating them by a $\log n$ factor is by no means clear.

The key to the following efficient universal simulation theorem is a uniform way to simulate any GSM or GST S in terms of its *monoid of transformations* $\mathcal{M}_S$. For brevity in this abstract we suppose that all GSTs given to our simulator use a fixed alphabet Σ; see [18] for the more-general result with variable Σ. Our simulator uses a larger alphabet Γ. Each character $c \in \Sigma$ induces a mapping $g_c : Q \to Q$ on the state set Q of S. Adding the identity map and closing out under composition yields $\mathcal{M}_S$. Let $Q =: \{ q_0, \ldots q_{N-1} \}$ and $k := \lceil \log_2 N \rceil$; then we represent members g of $\mathcal{M}_S$ by mappings $g' : \{0,1\}^k \to \{0,1\}^k$. For elements u which are not among the first r-many strings in $\{0,1\}^k$, we set $g'(u) := u$. In turn, we represent g' by the *list* $0^k\#g'(0^k)\#0^{k-1}1\#g'(0^{k-1}1)\#\ldots\#1^k\#g'(1^k)\#$. We actually regard each $\#$ separator as combined with the preceding 0 or 1 into one character of Γ, so that the list has length exactly $2k \cdot 2^k$. The list representation of g is also *normal*, meaning that all of its member 0-1 strings have the same length, namely k. We encode $\mathcal{M}_S$ by concatenating cg'_c for all $c \in \Sigma$. Below we also *mark* certain characters c by combining them with the symbol @.

Lemma 3.5. *The following two functions are computable by BMs in linear μ_1-time:*

(a) $mark(\vec{u}, v)$, which marks all occurrences of the string v in the normal list $\vec{u}$.

(b) $shuffle(\vec{u}, \vec{v})$, which is defined for normal lists $\vec{u} := (u_0, \ldots, u_{m-1})$ and $\vec{v} := (v_0, \ldots, v_{m-1})$ of element size k to be $(u_1, v_1, u_2, v_2, \ldots, u_m, v_m)$. Here k as well as m may vary. □

Lemma 3.6. *There is a BM M_0 which, given any $x \in \Sigma^*$ and the list representation of $\mathcal{M}_S$ for any GST S, computes $S(x)$ in μ_1-time $K|x|$, where K depends only on S.*

Proof Sketch. Let K_0 be the length of the encoding $\langle \mathcal{M}_S \rangle$ of $\mathcal{M}_S$, and let $n := |x|$. M_0 first computes $\langle \mathcal{M}_S \rangle^n$ by "recursive doubling," and then shuffles the bits of this with $x' := x_0^{K_0} x_1^{K_0} \cdots x_{n-1}^{K_0}$. A single pass then marks and pulls off the word $\vec{g} := g_0 g_1 \cdots g_{n-1}$, where g_i stands for g_{x_i}. So far the μ_1-time is $O(K_0|x|)$. The idea now is to compose the pairs g_i and g_{i+1} for all even i, producing a word $\vec{g}\,'$ of half the length. This can be done in a number of sweeps which depends on K_0 but *not* on n; hence the total μ_1-time for these compositions is $O(n)$. Since the word length halves at each step of the downward recursion, the total μ_1-time remains $O(n)$. An upward recursion then produces the sequence $\vec{h}$ given by $h_i := g_0 \circ g_1 \circ \ldots \circ g_i$. Indeed, this is just the sequence of "prefix sums" of $\vec{g}$ under composition, and the recursion is essentially the standard log-depth method for parallel prefix sums. By using *mark* to locate the elements $h_i(s)$, where s is the start state of S, followed by a pass which erases unmarked portions of $\vec{h}$ and one which shuffles the remainder with x, the trajectory of S on input x is recovered. One more dilation and shuffle with the tabulated output function ρ of S then yields $S(x)$. □

Theorem 3.7. *For $\mu = \mu_d$ or $\mu = \mu_{\log}$, there is a BM M_U and a computable function code such that for any BM M, there is a constant K such that for all inputs x to M, M_U on input $(code(M), x)$ simulates $M(x)$ within μ-time $K \cdot \mu$-$time(M, x)$.*

Proof Sketch. One track of the main tape of M_U mimics the main tape of the given BM M. Whenever M executes a pass by a GST S which reads a block $[a \ldots b]$, M_U copies the content x of this block and the code of S to separate tracks where it applies M_0, and copies the output y when M_0 is finished to the target block. Since M_0 runs in linear μ_1 time, the μ-time of the whole sequence by M_U is within a constant factor of the μ-time charged to M for the pass. □

The consequence is the following tight deterministic hierarchy theorem for these cost functions. A function t is *μ-time constructible* if $t(n)$ is computable from n in binary notation in μ-time $O(t(n))$.

Theorem 3.8. *Let μ be as in Theorem 3.7, and let t_1 and t_2 be functions such that $t_1(n) \geq n$, $t_1 = o(t_2)$, and t_2 is μ-time constructible. Then* DμTIME$[t_1]$ *is properly contained in* DμTIME$[t_2]$.

Proof Sketch. A BM can keep count of its own elapsed μ-time without slowing down its computation by more than a constant factor, basically because two numbers can be added in one GST pass after shuffling their bits. The rest follows by standard diagonalization methods (see [8]). $\square$

4. Conclusion

We have developed a natural model of computation which provides a parameter μ, whose settings between $\mu(a) = \log a$ and $\mu(a) = a$ span the difference between a log-cost RAM and a multitape Turing machine. This also leads to a concept of "dimensionality" for problems which differs from earlier approaches based on d-dimensional Turing machines or arrays, and which ties into important issues of organizing memory. The concept studied here places a premium on arranging data so that much work can be done in a single sweep through a linear block of memory by a finite-state machine, whereas d-dimensional TMs are allowed to change the direction of their tape heads at each step. The main technical problem left open in the present paper is how efficiently a BM with memory cost function μ_d can simulate a d-dimensional Turing machine. Although we have not separated RAM and TM complexity classes, we expect that this work will foster greater knowledge about the structure of their computations.

Acknowledgments. Thanks are due to Professors Michael C. Loui, Pierre McKenzie, Gilles Brassard, and Russell Miller for opportunities to discuss preliminary forms of the ideas presented here.

References

[1] A. Aggarwal, A. Chandra, and M. Snir. Hierarchical memory with block transfer. In *Proc. 28th IEEE FOCS*, pages 204–216, 1987.

[2] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. In *Proc. 3rd IEEE Structures*, pages 47–59, 1988.

[3] S. Cook and R. Reckhow. Time bounded random access machines. *J. Comp. Sys. Sci.*, 7:354–375, 1973.

[4] E. Grandjean and J. Robson. RAM with compact memory: a robust and realistic model of computation. In *Proc. CSL '90*, LNCS. Springer, 1991.

[5] Y. Gurevich and S. Shelah. Nearly-linear time. In *Proc. Logic at Botik '89*, volume 363 of *LNCS*, pages 108–118. Springer, 1989.

[6] F. Hennie and R. Stearns. Two–way simulation of multitape Turing machines. *J. ACM*, 13:533–546, 1966.

[7] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *J. ACM*, 24:332–337, 1977.

[8] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison–Wesley, Reading, MA, 1979.

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

[10] M. Loui. Simulations among multidimensional Turing machines. *Theor. Comp. Sci.*, 21:145–161, 1981.

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

[12] M. Loui and D. Luginbuhl. The complexity of on-line simulation between multidimensional Turing machines and random-access machines. *Math. Sys. Thy.*, 25:293–308, 1992.

[13] M. Loui and D. Luginbuhl. Optimal on–line simulations of tree machines by random access machines. *SIAM J. Comput.*, 21:959–971, 1992.

[14] W. Paul, N. Pippenger, E. Szemeredi, and W. Trotter. On determinism versus nondeterminism and related problems. In *Proc. 24th IEEE FOCS*, pages 429–438, 1983.

[15] W. Paul and R. Reischuk. On time versus space II. *J. Comp. Sys. Sci.*, 22:312–327, 1981.

[16] W. Paul, J. Seiferas, and J. Simon. An information-theoretic approach to time bounds for on-line computation. *J. Comp. Sys. Sci.*, 23:108–126, 1981.

[17] N. Pippenger and M. Fischer. Relations among complexity measures. *J. ACM*, 26:361–381, 1979.

[18] K. Regan. Linear time and memory efficient computation. Technical report, Computer Science Department, State University of NY at Buffalo, TR 92-28, 1992.

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

[20] C. Schnorr. Satisfiability is quasilinear complete in NQL. *J. ACM*, 25:136–145, 1978.

[21] K. Wagner and G. Wechsung. *Computational Complexity.* D. Reidel, 1986.

[22] R. Weicker. Turing machines with associative memory access. In *Proc. ICALP '74*, volume 14 of *LNCS*, pages 458–472. Springer, 1974.