Parallelizing Time With Polynomial Circuits

Ryan Williams *
Institute for Advanced Study, Princeton, NJ 08540
ryanw@ias.edu

Abstract

We study the problem of asymptotically reducing the runtime of serial computations with circuits of polynomial size. We give an algorithmic size-depth tradeoff for parallelizing time t random access Turing machines, a model at least as powerful as logarithmic cost RAMs. Our parallel simulation yields logspace-uniform $t^{O(1)}$ size, $O(t/\log t)$-depth Boolean circuits having semi-unbounded fan-in gates. In fact, for appropriate d, uniform $t^{O(1)}2^{O(t/d)}$ size circuits of depth $O(d)$ can simulate time t. One corollary is that every log-cost time t RAM can be simulated by a log-cost CRCW PRAM using $t^{O(1)}$ processors and $O(t/\log t)$ time. This improves over previous parallel speedups, which only guaranteed an $\Omega(\log t)$-speedup with an exponential number of processors for weaker models of computation. These results are obtained by generalizing the well-known result that $\mathsf{DTIME}[t] \subseteq \mathsf{ASPACE}[\log t]$.

*This work was performed while the author was a student at Carnegie Mellon University, supported by the NSF ALADDIN Center (NSF Grant No. CCR-0122581). A preliminary version of this work appeared in the *17th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'05)* [25].

1 Introduction

A fundamental problem in complexity theory and parallel computing is to determine the extent to which arbitrary time-bounded computations can be parallelized. For example, the $\mathsf{NC} = \mathsf{P}$ question asks if polynomial time computations can be captured by $(\log n)^{O(1)}$ depth, bounded fan-in, $n^{O(1)}$ size circuits on inputs of length n. Over the years, numerous parallel simulations have been given for speeding up general (serial) computational models (*e.g.* [17, 18, 4, 13, 14, 11]). Each such parallel simulation can be interpreted as a family of Boolean circuits, where the number of processors in the parallel algorithm is polynomially related to the circuit size of the family. However, no known parallel simulation achieves an $\omega(1)$ speedup with only a polynomial number of processors, cf. [5, p.63]. Put another way, it is not known how to construct $t^{O(1)}$-size circuits of depth $o(t)$ for a time t serial algorithm. As a partial result in this direction, Meyer auf der Heide [15] has shown that restricted RAMs (lacking a full set of operations and indirect addressing) can indeed be sped up by a $\log t/(\log\log t)^2$ factor using $t^{O(1)}$ processors, in a similarly restricted PRAM model.

In 1985, after presenting various speedups of deterministic Turing machines on parallel machines with an exponential number of processors, Dymond and Tompa [4] concluded that it would be "interesting to know what speedups can be achieved using a number of processors that is only polynomial in the time bound." While this is certainly an interesting theoretical problem, it should also have practical import, perhaps more so than parallel simulations which obtain a faster speedup but require exponentially many processors. The primary intent of this paper is to present positive answers to Dymond and Tompa's problem, demonstrating that we have not yet reached the end of theoretical improvements.

The only result we could find that directly addresses Dymond and Tompa's problem is from Mak [14] in 1997, who shows that a time t RAM can be sped up to $\varepsilon t + n$ by a CREW PRAM of $t^{O(1)}$ processors, for $\varepsilon > 0$.[1] However, work of Chandra, Stockmeyer, and Vishkin [3] implies the existence of circuits of $o(t)$ depth and polysize for time t Turing machines. In particular, the authors prove the following.

Theorem 1.1 (Chandra-Stockmeyer-Vishkin [3], Theorem 4.3) *For any $\varepsilon > 0$, every circuit with fan-in two, $S(n)$ size, and $D(n)$ depth can be simulated by an unbounded fan-in circuit of $O(2^{(\log n)^\varepsilon} \cdot S(n))$ size and $O(D(n)/(\varepsilon \log\log n))$ depth.*

Their construction can be easily generalized to bounded fan-in circuits of $S(n)$ size and $D(n)$ depth, which can be simulated by $O(2^{c(\log n)^\varepsilon} \cdot S(n))$ size and $O(D(n)/(\varepsilon \log\log n))$ depth for some constant $c \geq 1$.

Let us briefly outline their construction, and why it implies good low-depth circuits for Turing machines. The authors observe that a bounded fan-in circuit with depth $\varepsilon \log\log n$ and a single output gate can be seen as a Boolean function f taking $O((\log n)^\varepsilon)$ inputs. This f can be represented by an unbounded fan-in depth-2 circuit (in particular, a DNF) of size $2^{O((\log n)^\varepsilon)}$. Therefore, every $\varepsilon \log\log n$ levels of a given depth $D(n)$ circuit can be replaced with a collection of $2^{O((\log n)^\varepsilon)}$ size, depth 2 circuits. This results in an unbounded fan-in circuit of $S(n) \cdot 2^{O((\log n)^\varepsilon)}$ size and $O(D(n)/\log\log n)$ depth. The construction is uniform in the sense that each depth-2 circuit can be produced in $2^{O((\log n)^\varepsilon)}$ time from a truth table for each function f. Since it follows from

[1] Note that unit cost RAMs do not have a linear speedup theorem like Turing machines.

standard tableau constructions (cf. [21]) that time t Turing machines can be simulated by bounded fan-in circuits of depth $O(t)$ and size $O(t^2)$, the above transformation results in a simulation via $O(t/\log\log n)$ depth, unbounded fan-in polynomial circuits.

Corollary 1.1 *Let L be a language recognized by a Turing machine using time $t(n)$. Then L is recognized by a circuit family having unbounded fan-in, $t^{O(1)}(n)$ size, and $O(t/\log\log n)$ depth.*

1.1 Our Results

We shall present a construction that works for more general models, and produces circuits of smaller depth. The more general computational model is known in the literature as the *random access Turing machine*. Let us informally describe this model. Random access TMs are equipped with the normal tapes of a Turing machine, but instead of the usual two-way access, we have random access to each tape, in the following sense. Each tape T also has an *index tape* associated with it, which holds a small binary integer indicating the location of the tape head on T. The machine can move the tape head of T by editing the bits of the index tape of T, which has the effect of "jumping" the head to the current location indicated by the bits. See the Preliminaries (Section 2) for more details.

Starting in Section 3.1, we give a new proof of the old result that $\mathsf{DTIME}[t] \subseteq \mathsf{ASPACE}[\log t]$ by Chandra, Kozen, and Stockmeyer [2], which works for random access Turing machines. The objective of this new proof is to demonstrate how one can simulate a computation efficiently with fewer alternations than expected. (Very roughly speaking, a lower number of alternations translates to lower depth circuits, so this kind of simulation is naturally of interest to us.) Our simulation of $\mathsf{DTIME}[t]$ in $\mathsf{ASPACE}[\log t]$ implicitly constructs and evaluates a *computation history* of a time t random access Turing machine. While this history is of length $O(t \log t)$, we show how to compute this history implicitly with only $O(t)$ alternations and logarithmic space. In Section 3.2, we strengthen our result in Section 3.1 by working with a specially designed computation history that has length only $O(t)$. More precisely, we prove:

Theorem 1.2 *For $t(n) \geq n \log n$ and $a(n)$ such that $\log n \leq a(n) \leq t(n)/\log t(n)$,*

$$\mathsf{DTIME}[t(n)] \subseteq \Sigma_{3a(n)}\mathsf{SPACE}\left[\frac{t(n)}{a(n)} + \log t(n)\right],$$

where the machine model is a random access Turing machine.

That is, random access TMs that run in time t can be replaced by alternating machines that use only $3a(n)$ alternations and $O(\frac{t(n)}{a(n)} + \log t(n))$ space. Recalling known relationships between alternating machines and uniform circuits (which we cover in the Preliminaries), Theorem 1.2 implies that we can simulate time t random access TMs with uniform $t^{O(1)}2^{O(t/d)}$ size, $O(d)$ depth circuits of semi-unbounded[2] fan-in, where d is a parameter obeying certain constraints. Setting $d = t/\log t$ implies the following (cf. Corollary 3.1).

Corollary 1.2 *Random access Turing machines running in time t can be simulated by uniform $t^{O(1)}$ size circuits of semi-unbounded fan-in and depth $O(t/\log t)$.*

[2]Confer with Section 2.1 for a definition.

In Section 3.3, we show how the circuit construction can be applied to obtain a simulation of logarithmic cost RAMs on logarithmic cost CRCW PRAMs using $t^{O(1)}$ processors and $O(t/\log t)$ time. The paper concludes with a discussion of several related open problems of interest.

2 Preliminaries

We use the notation $[n] := \{1, \ldots, n\}$. We assume familiarity with basic concepts of complexity theory, though we shall briefly review some notions required for this paper. Throughout, we implicitly assume that all functions considered are those efficiently constructible under the appropriate resources.

Circuit Uniformity. In this paper, our circuit constructions shall be space-bounded *uniform*, a notion first introduced by Borodin [1]. Let $s(n) \geq \log n$. We define a circuit family $\{C_n\}$ to be *$s(n)$-space uniform* if there is an $O(s(n))$-space bounded Turing machine that, given 1^n, outputs a description of C_n on its output tape. Stronger notions of uniformity exist, but this notion is all we need since the circuits that arise in our construction have at least linear depth.

Deterministic Machine Model. Our main result simulates *random access Turing machines*. This machine model is fairly powerful in that such machines can simulate logarithmic cost RAMs with only a constant factor overhead in runtime [19]. We shall not give a complete formal definition of the model, as our construction does not depend on a specific definition. Instead we shall highlight how it differs from the Turing machine model.

A random access TM has, along with all the usual multitape Turing machine equipment, $k+1$ binary *index* tapes that are write-only, one for the input and k for the k worktapes. (By write-only, we mean that the specific symbol over the index tape head never affects the behavior of M—only the current state and current symbols on the worktapes affect M.) Each random access TM has a special *access state* which, when entered, move the heads of the non-index tapes to the cells described by the respective index tapes, in one timestep. More precisely, to access the ith cell of the input or a worktape, one writes i in binary on the respective index tape (in $O(\log i)$ steps, if the index is blank), and switches to a special access state which moves the respective head to the ith cell of the respective tape in one step. Therefore, any location on a tape with s cells can be accessed within $\log s$ steps. In this paper, we shall often represent the index tape I for a tape with s cells as a non-negative integer, in the interval $[1, s]$. Thus it makes sense for us to refer to the Ith cell of a tape.

For a transition r of a random access TM having only one tape, we define the *initial state of r* to be the state required in order for r to apply, and the *read-symbol of r* to be the symbol on the (non-index) tape that must be read by r in order for the transition to be taken. We also define the *read-symbol of step i* to be the symbol read (from the non-index tape) in the ith step of the computation. The *write-symbol of r* and *write-symbol of step i* are both defined analogously. Note that a random access Turing machine can write to both the usual tape and the index tape in one step. The write to the index tape only affects the head position of the (non-index) tape. The above notions are easily generalized to multiple tapes.

2.1 Some Background on Alternation and Uniform Circuits

It shall be convenient to describe circuits with alternating machines, a natural model of parallelism [2]. Alternating machines generalize both nondeterministic and co-nondeterministic machines, having "existential" and "universal" states; the reader may refer to Papadimitriou [16] for definitions.

$\mathsf{ATIME}[t]$ and $\mathsf{ASPACE}[s]$ denote the classes of alternating machine time t and space s, respectively; $\mathsf{ATISP}[t, s]$ denotes the sets accepted by alternating machines using both time t and space s simultaneously. $\Sigma_a\mathsf{SPACE}[s]$ denotes the class of sets accepted by alternating machines starting in an existential state and taking at most a alternations and $O(s)$ space in every branch.

Strong relationships between alternating computations and uniform circuits have been established in the past. In the following, we give a short history of the relevant results from this line of work. Ruzzo [20] was the first to establish a close relationship between the size/depth complexity of uniform circuits for languages and the time/space complexity of alternating machines recognizing languages. He defined the notion of U_E-uniformity (a type of uniformity stronger than ours), and proved that a language L is recognized by $O((\log n)^c)$-time log-space alternating machines if and only if L is recognized by a U_E-uniform family of $O((\log n)^c)$-depth, bounded fan-in, and polynomial size. Thus the class NC can be characterized in terms of alternating machines. In general, Ruzzo characterized alternating time t and space s in terms of uniform bounded fan-in circuits of depth t and size $2^{O(s)}$.

Later, similarly strong relations were established between unbounded fan-in circuits and alternating machines with no *a priori* time bound. Stockmeyer and Vishkin [22, Theorem 3], citing Ruzzo and Tompa (unpublished work), showed (in the context of relating alternating machines and CRCW PRAMs) that the class $\Sigma_a\mathsf{SPACE}[s]$ is equivalent to $O(s)$-space uniform unbounded fan-in circuits of $2^{O(s)}$ size and depth $O(a)$. For completeness, we give a proof of the relevant portion of the result proved by Stockmeyer-Vishkin, following a sketch by Immerman [9, p.87].

Theorem 2.1 (Ruzzo-Tompa and Stockmeyer-Vishkin [22]) *For $a(n) \geq s(n) \geq \log n$, any language in $\Sigma_{a(n)}\mathsf{SPACE}[s(n)]$ is accepted by an unbounded fan-in, $O(s(n)+\log a(n))$-space uniform circuit family of $O(a(n))$ depth and $a(n) \cdot 2^{O(s(n))}$ size.*

Proof. Let M be a $s(n)$ space machine making $a(n)$ alternations. Let C_1, C_2 denote configurations of M on an n-symbol input x. Define the predicate

$E(C_1, C_2, x) \iff$ There is a computation in $M(x)$ from C_1 to C_2, through existential states only.

Similarly, define

$U(C_1, C_2, x) \iff$ There is a computation in $M(x)$ from C_1 to C_2, through universal states only.

By definition, the set of triples (C_1, C_2, x) satisfying E (respectively, U) is in $\mathsf{NSPACE}[s(n)]$ (respectively, $\mathsf{coNSPACE}[s(n)]$). Recall $\mathsf{NSPACE}[s(n)] = \mathsf{coNSPACE}[s(n)]$ (Immerman [8] and Szelepcsenyi [23]). Moreover, $\mathsf{NSPACE}[s(n)]$ is recognized by $O(s(n))$ space uniform, unbounded fan-in circuits of $2^{O(s(n))}$ size and $O(s(n))$ depth (cf. Borodin [1], Karp and Ramachandran [10, Sections 3.9 and 4.4]).

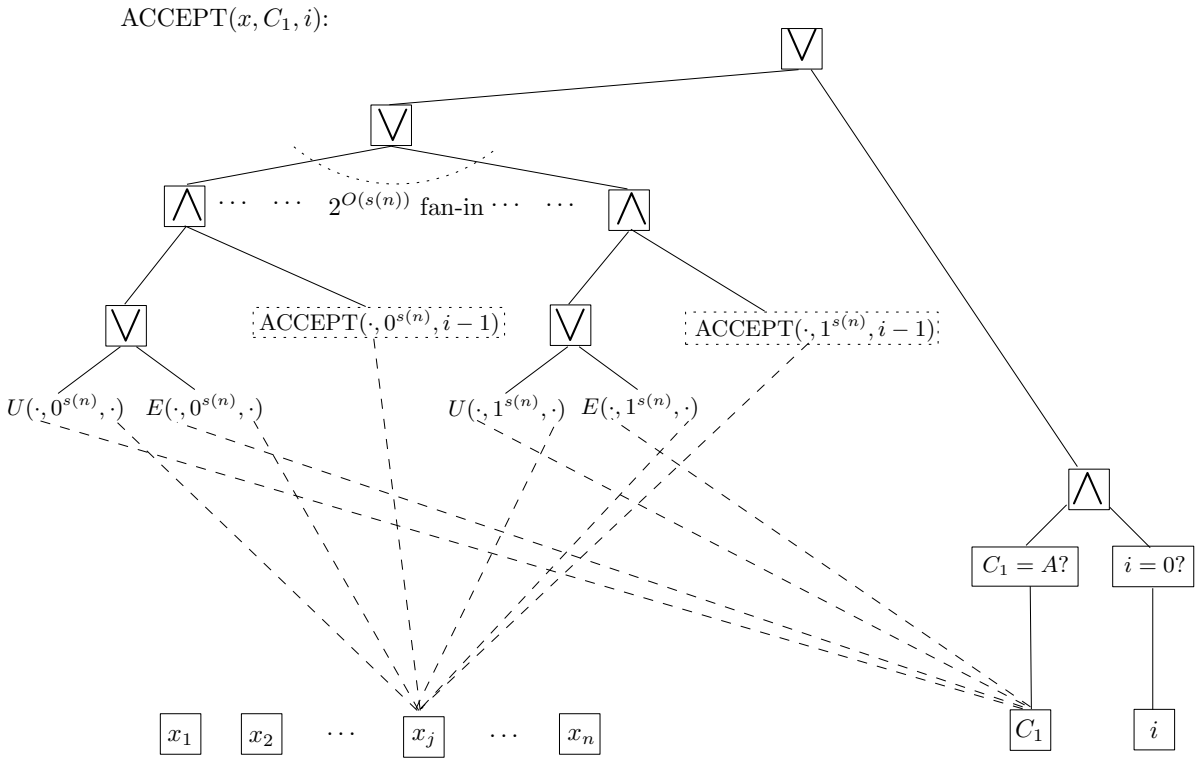

Figure 1: **A schematic view of the ACCEPT circuit.** The "$i = 0$?" and "$C_1 = A$?" boxes represent equality tests. The OR-gate with $2^{O(s(n))}$ fan-in has an input wire for every possible choice of the configuration C_2. For each C_2, a copy of the circuit for ACCEPT$(x, C_2, i-1)$ is constructed, and the input C_1 is plugged into copies of the circuits for $U(C_1, C_2, x)$ and $E(C_1, C_2, x)$. The dashed wires leading out of the input x_j show how an arbitrary bit of x connects to the rest of the circuit. The dashed wires leading out of the input configuration C_1 show how it connects to the rest of the circuit.

Without loss of generality, let I and A be the unique initial and accepting configuration for M on n-symbol inputs. Now define a circuit ACCEPT inductively, where x is an input to M, C_1 is an $O(s(n))$-length string representing a configuration of M on x, and $i \in [a(n)]$:

$$\begin{aligned}\text{ACCEPT}(x, C_1, i) \quad := \quad & ((i = 0) \wedge (C_1 = A)) \vee \\ & (\exists \text{ configuration } C_2)[\text{ACCEPT}(x, C_2, i-1) \wedge (E(C_1, C_2, x) \vee U(C_1, C_2, x))].\end{aligned}$$

See Figure 2.1 for a schematic view of the ACCEPT circuit.

We claim that, for all $x \in \{0,1\}^n$, ACCEPT$(x, I, a(n))$ is true iff $M(x)$ accepts. It suffices for us to show that ACCEPT(x, C, i) is true iff there is a computation path from C to the accepting configuration A that uses at most i different quantifiers ($i - 1$ alternations). We prove this by induction on i.

For $i = 0$, since the computation is alternating, every state is either existential or universal. Thus "zero quantifiers" are used iff no steps are taken at all, *i.e.* if $C = A$. But this is precisely the case in which ACCEPT$(x, C, 0)$ is true.

For $i > 1$, there is a computation path from C to A using at most i quantifiers, iff there is a configuration C_2 such that there is a computation path from C to C_2 such that all configurations in the path have only one type of state (existential or universal), and C_2 can reach A using at most $i-1$ quantifiers. This is true iff

$$(\exists \text{ configuration } C_2)[\text{ACCEPT}(x, C_2, i-1) \wedge (E(C_1, C_2, x) \vee U(C, C_2, x))] \iff \text{ACCEPT}(x, C_1, i),$$

by definition of E, U, and ACCEPT.

The size of a circuit in the family is dominated by the guessing of C_2, which requires $2^{O(s)}$ gates for each $i = 1, \ldots, a(n)$. (Notice there are only $s(n)^2$ copies of the $2^{O(s(n))}$-size circuits E and U in the complete circuit: one copy for each pair of configurations.) All in all, the circuit size is $a(n) \cdot 2^{O(s(n))} = 2^{O(s(n))}$. The depth is $O(s(n) + a(n)) = O(a(n))$, due to the depths of E, U, and the recursive construction for ACCEPT.

We now argue that the circuit family resulting from this construction can be made $O(s(n) + \log a(n))$-space uniform. Proving this in detail would be rather tedious, so we give an outline of the ideas. For every fixed configuration C_1 and $i \in [a(n)]$, let the "ACCEPT$(\cdot, C_1, i)$ gadget" be the portion of the ACCEPT diagram above that only includes the ANDs, ORs, and equality tests; *i.e.* the E, U, and dashed boxes of nested ACCEPTs are omitted. There are at most $(s(n)+\log a(n))^{O(1)}$ gates within every ACCEPT$(\cdot, C, i)$ gadget, and the circuits involved are constant-depth uniform. In particular, testing equality of two strings is easily in uniform AC^0. Combining these observations with the uniformity of E and U, the uniformity of ACCEPT follows by induction on i. □

Semi-unbounded Fan-in Circuits. A *semi-unbounded fan-in* circuit family has circuits where only OR gates are unbounded (AND gates have bounded fan-in). This circuit type was first identified and studied by Venkateswaran [24]. Note that one can always turn an unbounded fan-in circuit of depth d and size s into a semi-unbounded one of depth $O(d \log s)$, by replacing large fan-in ANDs with trees of bounded fan-in ANDs. However, this transformation can increase the number of gates: for example, a one-gate circuit consisting of a single AND of n bits becomes a bounded fan-in circuit on $O(n)$ gates.

In the uniform setting, there appears to be a difference in the complexity of languages recognized by semi-unbounded and unbounded fan-in circuits. For example, Venkateswaran's work showed that uniform semi-unbounded circuits of $O(\log n)$ depth and $n^{O(1)}$ size capture *exactly* LOGCFL, the class of languages which are logspace-reducible to context-free languages. However, it is believed that $\mathsf{LOGCFL} \subsetneq \mathsf{AC}^1$ (recall that AC^1 captures uniform *unbounded* circuits of logdepth and polysize). We note that Theorem 2.1 actually holds for semi-unbounded circuits when $a(n)$ is sufficiently large:

Corollary 2.1 *For $a(n) \geq s^2(n)$ and $s(n) \geq \log n$, any language in $\Sigma_{a(n)}\mathsf{SPACE}[s(n)]$ is accepted by a uniform circuit family of semi-unbounded fan-in, $O(a(n))$ depth, and $2^{O(s(n))}$ size.*

Proof. First, note $\mathsf{NSPACE}[s(n)] = \mathsf{coNSPACE}[s(n)]$ can be recognized by $O(s(n))$-space uniform, bounded fan-in circuits of $2^{O(s(n))}$ size and $O(s^2(n))$ depth (cf. Borodin [1], Karp and Ramachandran [10, Sections 3.9 and 4.4]). Therefore the copies of E and U in the circuit for ACCEPT in Theorem 2.1 can be assumed to have bounded fan-in and $O(a(n))$-depth, by assumption on $a(n)$.

After replacing E and U in ACCEPT with circuits of the above type, observe that the circuit is only unbounded with respect to its *OR* gates: in particular, the choice of the configuration C_2 is the only kind of unbounded choice. □

It immediately follows that, if Theorem 1.2 holds, then time-t random-access TMs can be simulated by uniform circuits of semi-unbounded fan-in, $O(a(n))$-depth, and $t^{O(1)}2^{O(t(n)/a(n))}$ size, for suitable $a(n)$. In particular, $a(n) = t(n)/\log t(n)$ yields polysize circuits of non-trivial depth.

2.2 Review of CRCW PRAMs

We assume a RAM model with the usual set of operations. We shall work with the Concurrent Read, Concurrent Write (CRCW) PRAM. In such a model, we have a collection of RAM processors $P_1, \ldots, P_k, \ldots$ running a common program in parallel; the only difference between the processors is an instruction that allows P_i to load its own number i into a register. Each processor has its own local memory, and they all share a global memory. Global memory locations can be concurrently read and concurrently written, in that when more than one processor wants to write to the same location, we imagine a memory control that simply allows the processor of lowest index to write and ignores the rest (this is sometimes called the PRIORITY CRCW model).

Two measures of time are typically used for RAMs and PRAMs: unit cost and logarithmic cost. In the unit cost measure, every instruction takes one unit of time. Such a model can be "abused" in that it permits, for example, the addition of arbitrarily large integers in constant time, provided they are already present in memory. We use the logarithmic cost measure, which is more realistic but context sensitive: every instruction on integers i and j takes $\lceil \log i \rceil + \lceil \log j \rceil$ units of time.

3 Main Result

In this section, we prove Theorem 1.2 from the Introduction, which claims:

For $t(n) \geq n \log n$ and $a(n)$ such that $\log n \leq a(n) \leq t(n)/\log t(n)$,

$$\mathsf{DTIME}[t(n)] \subseteq \Sigma_{3a(n)}\mathsf{SPACE}\left[\frac{t(n)}{a(n)} + \log t(n)\right].$$

As shown in Section 2.1, this implies that uniform semi-unbounded fan-in circuits of $t^{O(1)}2^{O(t/a)}$ size and $O(a)$ depth can simulate time t, for $a(n)$ satisfying the parameters of Theorem 1.2. The driving idea behind our result is to generalize the well-known fact for Turing machines that $\mathsf{DTIME}[t] \subseteq \mathsf{ASPACE}[\log t]$, *i.e.*, deterministic time is in alternating logspace, which was proven by Chandra, Kozen, and Stockmeyer [2]. This sort of approach is a departure from all other parallel speedups of deterministic time that we are aware of, which instead build upon the proofs of $\mathsf{DTIME}[t] \subseteq \mathsf{ATIME}[o(t)]$ (cf. [18, 4]) or $\mathsf{DTIME}[t] \subseteq \mathsf{SPACE}[t/\log t]$ (cf. [7, 19, 6]), for a variety of computational models. However, these constructions can be modelled using "pebbling games" which were studied quite exhaustively in the 70's and 80's, and have known limitations [5, p.68–69].

3.1 DTIME versus ASPACE

As mentioned above, we start by giving a new proof that $\mathsf{DTIME}[t] \subseteq \mathsf{ASPACE}[\log t]$ for random access Turing machines. Our particular proof technique shall give insight into how a generalization is possible.

We first need some terminology to effectively describe the behavior of our simulation. Let M be random access TM running in time $t(n)$ in the following. For simplicity, we assume that M has only one tape; this restriction shall be lifted in the proof of the main result.

Definition 3.1 *The* **local action** *of M on input x at step $i \in [t]$ is the triple (r, p, I), where r is the transition taken, $p \in [\log t]$ is the index tape head position, and $I \in [t]$ is the index tape of $M(x)$ at step i, represented as a non-negative integer. We use $\ell(M, x, i)$ to denote the ith local action of M on input x at step i.*

Clearly, a local action takes at most $O(\log t)$ symbols to describe. The intuition behind the name "local action" is that it does not give a complete description of $M(x)$ at step i, but merely the *local* information of the current state, read-symbol, write-symbol, index tape head, and index tape.

Definition 3.2 *The* **history of** *$M(x)$, written as $H_{M,x}$, is the sequence of local actions of $M(x)$ in chronological order,* i.e.

$$H_{M,x} \triangleq \ell(M, x, 1)\ \ell(M, x, 2)\ \ell(M, x, 3)\ \cdots\ \ell(M, x, t(|x|)).$$

Our simulation of M in alternating logspace constructs $H_{M,x}$, one local action at a time.

Theorem 3.1 *For random access Turing machines,* $\mathsf{DTIME}[t] \subseteq \mathsf{ASPACE}[\log t]$.

Proof. Let M be a $\mathsf{DTIME}[t]$ machine and x be an input. We describe an alternating simulation of $M(x)$ with two mutually recursive procedures, VERIFY and LAST-WRITE, having the specifications:

- $\mathsf{VERIFY}(x, \ell', i)$ accepts $\iff \ell' = \ell(M, x, i)$, and
- $\mathsf{LAST\text{-}WRITE}(x, I, i, \sigma)$ accepts $\iff \sigma$ is the write-symbol in the chronologically last timestep such that I is the index tape, over timesteps $1, \ldots, i$ of $M(x)$.

Intuitively, $\mathsf{LAST\text{-}WRITE}(x, I, i, \sigma)$ accepts precisely when σ is the last symbol written to the Ith cell, over steps $1, \ldots, i$ of $M(x)$.

Assuming VERIFY can be implemented in alternating logspace, the simulation of M is simple: existentially guess a local action $\ell' = (r', p', I')$ such that r' changes to an accept state, then *accept* iff $\mathsf{VERIFY}(x, \ell', t(|x|))$ accepts. The procedure LAST-WRITE aids in the description of VERIFY. The two are described as follows:

$\mathsf{VERIFY}(x, \ell', i)$:

Let $\ell' = (r', p', I')$. If $i = 1$, then *accept* iff σ is the first symbol of x, $I' = 0$, $p' = 0$, and the initial state of r' is the initial state of M.

Existentially guess $\ell'' = (r'', p'', I'')$.

Universally do 1 and 2:

1. Call $\mathsf{VERIFY}(x, \ell'', i - 1)$, and

2. Let σ' be the read-symbol of r'. We universally do two steps.

   First, we ensure that p', I', and the initial state for r' are correct: let q' and q'' be the initial states of r' and r'', respectively. If p'', I'', and q'' become p', I', and q' after simulating the head movement and state change given by r'', then *accept* else *reject*.

   Second, we ensure that σ' is correct. That is, *accept* iff LAST-WRITE$(x, I', i-1, \sigma')$ accepts.

LAST-WRITE(x, I, i, σ):

If $i = 1$, then *accept* iff the Ith cell of the initial input is σ.

Existentially guess $\ell' = (r', p', I')$.

Universally do 1 and 2:

1. Call VERIFY(x, ℓ', i), and
2. If $I = I'$ then *accept* iff the write-symbol of r' is σ.

   Otherwise, *accept* iff LAST-WRITE$(x, I, i-1, \sigma)$ accepts.

We claim that the two procedures meet their prescribed specifications as given above. The proof is by induction on i.

When $i = 1$, VERIFY clearly accepts iff the head positions are zero and the machine is in its initial state: this corresponds to the first local action. Similar can be said for LAST-WRITE.

Now assume the specifications hold for $i = k - 1$.

When $i = k$, let's consider VERIFY: The first universal branch of VERIFY accepts iff ℓ'' is the $(i-1)^{th}$ local action, by induction hypothesis. Therefore, the check that p', I', and the initial state for r' are correct works by simulating from ℓ'' for one step. The verification that σ' is a correct read-symbol follows by induction hypothesis on LAST-WRITE.

Now for LAST-WRITE: By the above argument for VERIFY (whose correctness has been proven for $i = k$), ℓ' is indeed the k^{th} local action. Thus if $I' = I$ then the kth step is the last time a symbol was written to I, so the procedure must accept iff that symbol is σ'. If $I' \neq I$, then the last point in which a symbol was written to I is prior to step i. By induction, LAST-WRITE$(x, I, i-1, \sigma)$ is correct, therefore LAST-WRITE(x, I, i, σ) is also correct in this case. □

One can discern by induction on i that the number of alternations taken by VERIFY(x, ℓ, i) is at most $c \cdot i$, for some constant $c > 1$. (A formal proof of such a bound can be found in the proof of the main result, in the next section.) Therefore, the above alternating simulation takes $O(t(n))$ alternations, which translates to $O(t(n))$-depth circuits. Our primary issue is how much this alternation bound can be reduced, without increasing the space usage significantly.

Notice that $|H_{M,x}| = O(t(n) \log t(n))$. This is suboptimal, as the computation history of M on x could also be described by giving the list of transitions that M takes at each timestep. This representation takes only $O(t(n))$ symbols; however, it cannot be efficiently accessed, in that it is difficult to quickly determine the local action at a particular timestep from the list of transitions. The next section shows how to represent histories in $O(t(n))$ space, in such a way that local actions can be quickly reconstructed. This succinct representation allows us to efficiently perform computations on $O(\log t(n))$-sized blocks of transitions at a time, which results in fewer recursive calls to VERIFY and LAST-WRITE, consequently lowering the number of required alternations.

3.2 Proof of Main Result

We now establish Theorem 1.2. Our simulation is a generalization of the above VERIFY/LAST-WRITE scheme. To reduce the total number of alternations used by the simulation, we shall guess many transitions at a time, in blocks. In parallel, we can check the correctness of transitions within the current block, as well as chronologically earlier blocks that affect the current block. By properly choosing the block size and increasing the fan-in of gates, we simultaneously reduce the circuit depth and increase the size by at most polynomial.

Without loss of generality, suppose $a(n)$ divides $t(n)$ in the following.

Definition 3.3 *Let x be an input string to M. The* **brief history of** *$M(x)$, denoted as $BH_{M,x}$, is the string of the form*

$$\ell_0 \ \vec{r}_0 \ \ell_1 \ \vec{r}_1 \ \cdots \ \ell_{a(n)-1} \ \vec{r}_{a(n)-1},$$

where ℓ_i is the local action of $M(x)$ at step $(i \cdot t(n)/a(n) + 1)$, and $\vec{r}_i$ is the vector of the next $(t(n)/a(n) - 1)$ transitions taken by $M(x)$, starting from step $(i \cdot t(n)/a(n) + 2)$.

Note that since $a(n) \leq t(n)/\log t(n)$, it follows that $|BH_{M,x}| = O(a(n) \log t(n) + t(n)) = O(t(n))$. One may think of the brief history of $M(x)$ as a succinct description of $H_{M,x}$.

Definition 3.4 *Fix a machine M and input x. A* **block** *is a substring of $BH_{M,x}$ of the form $\ell_i \ \vec{r}_i$.*

That is, a block is a local action followed by $t(n)/a(n) - 1$ transitions, and the number of different blocks is precisely $a(n)$. Notice that the length of a block is $O(t(n)/a(n) + \log t(n))$, which is the same as the space usage of the alternating simulation, as claimed by Theorem 1.2. (This is not a coincidence—the alternating machine shall hold roughly one block in memory at a time, along with a few small auxiliary variables.)

We are now ready to begin the proof.

Proof of Theorem 1.2. For simplicity, just as in Section 3.1, we begin by assuming that the given random access TM M has only one read-write tape which initially contains the input and $t(n)$ blanks. Later, we show how the proof can easily be extended to any (finite) number of random access tapes.

In the following, we define an alternating machine A to simulate M. The goal of A is to essentially reconstruct the brief history of $M(x)$, while only holding one block in memory at any point in time. Just as in the proof of Theorem 3.1, the simulating machine A uses a VERIFY and LAST-WRITE procedure, both of which are assumed to have direct access to the input tape of A and the description of M (*i.e.* the table of its transition function). They have the specifications:

- VERIFY(x, b, i) accepts $\iff$ b is the ith block of $BH_{M,x}$, and
- LAST-WRITE(x, I, i, σ) accepts $\iff$ σ is the write-symbol in the chronologically last timestep such that I is the index tape, over timesteps $1, \ldots, i \cdot t/a$.

Intuitively, these procedures are analogous to those of the same names in Theorem 3.1, except instead of working over the *local actions* of the history, they work over the *blocks* of the brief

history. For example, LAST-WRITE(x, I, i, σ) accepts precisely when σ is the last symbol written to the Ith cell, over the timesteps covered by blocks $b_1, \ldots, b_i$.

The procedures contain a number of deterministic checks, each of which are implementable in $O(t(n)/a(n) + \log t(n))$ space. We presume that if any check fails, then the machine *rejects* the current computation branch. We also presume that a recursive call erases all worktape content, except for $t(n)$ and $a(n)$ (written in binary), and the relevant arguments for the call.

The description of an alternating machine A in terms of the above procedures is very simple.

$A(x)$: Initialize variable $i := a(n)$.
Existentially write a block $b = \ell\vec{r}$ to tape.
If the last transition of $\vec{r}$ is not accepting, then *reject*.
Accept iff VERIFY(x, b, i) accepts.

It is obvious that, if the specification for VERIFY is correct, then A is correct: $A(x)$ accepts $\iff$ the $a(n)$th block of $BH_{M,x}$ ends in an accepting state $\iff$ $M(x)$ accepts. The space usage of A is $O(\log t)$ (the size of i), plus $O(t(n)/a(n) + \log t)$ (the size of a block), plus the space usage of VERIFY. Finally, the number of alternations used by A is at most one plus the number of alternations used by VERIFY. Therefore, in the remainder of our analysis we shall concentrate on proving the correctness of VERIFY, its space usage, and the number of alternations it requires.

The next subsection describes VERIFY in its entirety. We shall sometimes abbreviate $t(n)$ to t, and $a(n)$ to a.

3.2.1 VERIFY(x, b, i)

Let ℓ and $\vec{r}$ be the local action and transition vector asserted by block b, respectively. There is one base case.

- If $i = 1$, then we need to determine that b is the first block of $BH_{M,x}$. First, check that ℓ is in the initial state, the read-symbol is the first symbol of x, the index tape is 0, and the index tape head is at the 0th index tape cell. Then, simulate $M(x)$ for the first $t(n)/a(n)$ steps, recording the transitions taken.
  Accept iff for all $j \in [t(n)/a(n) - 1]$, the transition taken at the $(j+1)$th step equals $\vec{r}[j]$.
  This check runs in $O(t(n)/a(n) + \log t)$ space.

At this point in the procedure, $i > 1$, and the following assumption is made:

($*$): *ℓ is correct, and all read-symbols in transitions of $\vec{r}$ are correct. That is, ℓ is indeed the local action of $M(x)$ at the beginning of block b, and $\vec{r}$ is indeed the sequence of transitions taken by $M(x)$ afterwards in that block.*

From here, the procedure can be mentally divided into two parts.

1. Assuming ($*$), holds, the procedure determines that b is indeed the ith block. This part is fairly simple.

2. Check that assumption $(*)$ is true. This part requires a recursive call.

The procedure accepts iff both parts succeed.

(1) Assume $(*)$ holds, verify b is correct: Observe that if ℓ is correct, and all of the read-symbols in the transitions of $\vec{r}$ are correct, then all that is left to verify is that each transition makes sense.

For all transitions in b, check that the state change made in the current transition equals the state of the next transition. Assuming that all of the read-symbols are correct and the first state (the initial state of r in ℓ) is correct, these transitions are correct (all that affects a transition is the current state and the current symbol read). Therefore, this simple check also verifies the correctness of all write-symbols and index tape modifications. The check takes $O(t(n)/a(n))$ time.

(2) Check that $(*)$ is correct: First, **universally** choose $j \in [t(n)/a(n)]$. If $j = 1$, we shall check that ℓ is correct. If $j > 1$, we shall check that the read-symbol of transition $\vec{r}[j-1]$ is correct. Observe $(*)$ is true iff all these checks accept. More precisely:

If $j = 1$: *(Verify ℓ)*

Let q, σ, p, and I be the initial state, read-symbol, index tape head position, and index tape of ℓ, respectively.

Universally do 1 and 2:

1. *Check that σ is correct, by verifying that σ is the last symbol written to the Ith cell, among blocks $1, \ldots, i-1$.*

Accept iff $\mathsf{LAST\text{-}WRITE}(x, I, i-1, \sigma)$ accepts.

2. *Check that q, p, and I are correct, by guessing the previous block b_{i-1}, verifying that it ends by changing to q, p, and I, and verifying b_{i-1} is correct.*

Existentially guess a block $b_{i-1} = \ell_{i-1}\vec{r}_{i-1}$.

Check that the transition $\vec{r}_{i-1}[(t/a)-1]$ (that is, the last transition in $\vec{r}$) changes the state to q.

Determine the index tape and index tape head at each step in b_{i-1}:

> Start with the index tape I_{i-1} and head position p_{i-1} of ℓ_{i-1}. Simulate the head movements of $M(x)$ for the next $(t/a)-1$ steps, by following the movement directions given by each transition of $\vec{r}$. Call these obtained index tape values and head positions $I_{i-1} = I'_0, I'_1, \ldots, I'_{(t/a)-1}$ and $p_{i-1} = p'_0, p'_1, \ldots, p'_{(t/a)-1}$, given in chronological order.

Check that transition $\vec{r}_{i-1}[(t/a)-1]$ changes $I'_{(t/a)-1}$ to I, and $p'_{(t/a)-1}$ to p.

Accept iff $\mathsf{VERIFY}(x, b_{i-1}, i-1)$ accepts.

End if.

If $j > 1$: *(Verify the read-symbol of $\vec{r}[j-1]$:)*

Let σ be the read-symbol of transition $\vec{r}[j-1]$.

Determine the index tape and index tape head position at each step in b, just as in the above, except in this case let the simulation run for only $j-1$ steps. Call the values obtained $I_0, \ldots, I_{j-1} = I'$, $p_0, \ldots, p_{j-1} = p'$.

Check if I' was the index tape for more than one timestep during this simulation, *i.e.* if there is an $m < j-1$ such that $I_m = I'$. Another way of saying this is that the I'th cell of the tape is accessed more than once during the block.

- If this is true, let m' be the largest such m. The procedure now checks that σ is the write-symbol in the transition taken after $I_{m'}$. There are two cases:
  - If $m' = 0$, then check that σ is the write-symbol in the transition r of ℓ.
  - If $m' > 1$, then check that σ is the write-symbol in transition $\vec{r}[m']$.
- If this is not true, then the last write to the I'th cell occurred in a block prior to b_i. Therefore *accept* iff LAST-WRITE$(x, I', i-1, \sigma)$ accepts.

End if.

That completes the description of VERIFY. Now we describe LAST-WRITE.

3.2.2 LAST-WRITE(x, I, i, σ):

If $i = 1$**,** then simulate $M(x)$ from its initial configuration for $t(n)/a(n)$ steps.

If the index tape is ever I during the simulation, then *accept* iff the chronologically last write to I is σ.

If the index tape is never I, then

- If $I \leq n$, *accept* iff σ is the Ith symbol of input x.
- If $I > n$, *accept* iff σ is blank.

Else $(i > 1)$

Existentially guess block $b_i = \ell_i \ \vec{r}_i$.

Universally do both of the following:

1. *Accept* iff VERIFY(x, b_i, i) accepts.
2. Determine the index tape values at all steps in block b, just as in the above. Call the index tape values $I_0, I_1, \ldots, I_{(t/a)-1}$, in chronological order.
   - If there is a j such that $I_j = I$, then let j' be the largest such j'.
     Claim: The step corresponding to j' *is the most recent timestep where* I *was the index tape, prior to block* i.
     Accept iff σ is the write-symbol in transition $\vec{r}[j']$.
   - If every j is such that $I_j \neq I$, then I is never the index tape in block b_i.
     Accept iff LAST-WRITE$(x, I, i-1, \sigma)$.

End if.

This completes the description of LAST-WRITE.

3.2.3 Extending to Multiple Tapes

We now describe how to modify A to simulate a machine M with multiple tapes. In the general case, where M has k tapes, a local action now contains k index tape values and k index tape head positions, *e.g.* $I_1, \ldots, I_k$ and $p_1, \ldots, p_k$, along with (of course) k read-symbols, k write-symbols, and k head movement directions.

We modify LAST-WRITE to take an additional parameter $e = 1, \ldots, k$, which specifies the particular tape on which we are verifying the write of σ to the Ith cell. Then, the base case and Step 2 in the universal quantifier of LAST-WRITE are only concerned with determining the index tape values for that eth tape. (It is still the case though that the entire block b_i, with information about all tapes, is existentially guessed.)

We now turn to the modifications for VERIFY, which are a little more involved. Clearly, the base case of VERIFY can be carried out analogously, as it is just a t/a-step simulation of $M(x)$ from its initial configuration. Assuming $(*)$ is correct, the relevant state changes/transitions can be checked analogously. To check the correctness of $(*)$, we still universally guess $j \in [t/a]$.

To verify that ℓ is correct, the following modifications are made. In Step (1) of the universal quantifier, now we also use a **universal** quantifier to call LAST-WRITE for k times, one for each tape, in parallel. This is to ensure that each of the k read-symbols in ℓ are correct. In Step (2) of the universal quantifier, after the block b_{i-1} is guessed, we simulate all k index tapes, obtaining index tape values and index head positions for them, checking that they become the index tape and index heads of ℓ.

To verify that the k read-symbols $\sigma_1, \ldots, \sigma_k$ of a transition $\vec{r}[j-1]$ are correct, we simulate all index tapes and index heads for $j-1$ steps. Suppose after j steps, the resulting index tape values are $I'_1, \ldots, I'_k$. Now we **universally** quantify over $e' \in [k]$, and for each $I'_{e'}$, check if the $I'_{e'}$th cell of tape e' has already been accessed more than once in the block.

1. If yes, we need to check that $\sigma_{e'}$ is written at the last access of the $I'_{e'}$th cell, prior to the jth step. Since the cell is accessed more than once in the block, we only need to check the contents of the block to determine if $\sigma_{e'}$ was the last write. This takes only $O(t/a + \log t)$ space.

2. If no, then we need to check blocks to determine if $\sigma_{e'}$ was the last write, so we accept iff LAST-WRITE$(x, i-1, e', \sigma_{e'}, I'_{e'})$ accepts.

Note the above modification only inserts one new alternation per recursive call.

3.2.4 Analysis

The description of A is complete. We now argue that A executes within the desired resources.

It is routine to verify that each deterministic check within a particular call of VERIFY and LASTWRITE can be performed in deterministic $O(t(n)/a(n)+\log t(n))$ space, as each one maintains at most $O(t(n)/a(n)+\log t(n))$ symbols of information in its simulation of a block. Therefore these checks affect neither the number of alternations nor the overall space bound of A.

The number of alternations used by A can be discerned by induction on i, as follows. Define

$Q_{\mathsf{V}}(i)$ to be the maximum number of alternations used in a call to $\mathsf{VERIFY}(\cdot, i)$. Define $Q_{\mathsf{L}}(i)$ similarly, for $\mathsf{LAST\text{-}WRITE}(\cdot, i, \cdot)$. Clearly, $Q_{\mathsf{V}}(1) = Q_{\mathsf{L}}(1) = 0$.

Claim 1 *For all $i > 1$,*

$$\begin{aligned} Q_{\mathsf{V}}(i) &= \max\{1 + Q_{\mathsf{L}}(i-1), 2 + Q_{\mathsf{V}}(i-1)\}, \\ Q_{\mathsf{L}}(i) &= \max\{2 + Q_{\mathsf{V}}(i), 2 + Q_{\mathsf{L}}(i-1)\}. \end{aligned}$$

Proof. By inspection of VERIFY and LAST-WRITE. □

Claim 2 *For all $i \geq 1$, $Q_{\mathsf{V}}(i) \leq 3i$ and $Q_{\mathsf{L}}(i) \leq 3i + 2$.*

Proof. By induction on i. The base case $i = 1$ is obvious. Suppose the claim holds for $i - 1$. By induction, it follows that $Q_{\mathsf{V}}(i) \leq \max\{1 + (3(i-1) + 2), 2 + 3(i-1)\} = 3(i-1) + 3 = 3i$. Plugging this into the expression for $Q_{\mathsf{L}}(i)$, it follows that $Q_{\mathsf{L}}(i) \leq \max\{2 + 3i, 2 + (3(i-1) + 2)\} = \max\{3i + 2, 3i + 1\} = 3i + 2$. □

Therefore, the number of alternations used by A is at most $3 \cdot a(n)$.

The correctness of VERIFY and LAST-WRITE can also be proven by induction on i. These proofs are quite similar in structure to the proof of Theorem 3.1, except that they reason about blocks rather than local actions. As these proofs are quite tedious, we omit them. The key to them is the simple fact that if one can assume that the recursive calls are correct, then the guessed previous block is correct and all writes prior to the current block can be verified. This is enough information to verify the current block.

This completes the proof of Theorem 1.2. □

Corollary 3.1 *For $a(n)$ and $t(n)$ such that*

$$t(n) \geq n \log n \quad \textit{and} \quad t(n)/\log t(n) \geq a(n) \geq (t(n)/a(n) + \log t(n))^2,$$

deterministic time t random access Turing machines can be simulated by uniform $t^{O(1)} \cdot 2^{O(t/a)}$ size circuits of semi-unbounded fan-in and depth $O(a(n))$.

Proof. Follows immediately from the above theorem and Corollary 2.1 in the Preliminaries (Section 2). □

Hence, there exists a spectrum of size-depth tradeoffs for simulating time with semi-unbounded circuits. In particular, semi-unbounded fan-in circuits of polysize and depth $O(t/\log t)$ are possible. Hence, a PRAM model where massive OR-parallelism is cheap but AND-parallelism is expensive would still be capable of performing our simulation.

3.3 Simulating RAMs on CRCW PRAMs

We provide an application of our main result to speeding up RAMs with CRCW PRAMs that do not require too many processors. More precisely, we show:

Theorem 3.2 *The simulation of Theorem 1.2 can be performed by a logarithmic-cost CRCW PRAM in $O(t/\log t)$ time. Therefore every log-cost time t RAM can be simulated by a log-cost CRCW PRAM in $O(t/\log t)$ time with only $t^{O(1)}$ processors.*

To accomplish this, we extend a result of Stockmeyer and Vishkin, who proved that the class $\Sigma_{a(n)}\mathsf{SPACE}[s(n)]$ can be simulated with CRCW PRAMs running in $a(n)$ time and using $2^{O(s(n))}$ processors. Define $(a(n), s(n))$ to be *suitable* if

1. $a(n) \geq s(n) \geq \max\{\log n, \log a(n)\}$,
2. there is a deterministic TM which, on an input of length n, marks precisely $s(n)$ tape cells and computes $a(n)$ in binary, and
3. there is a unit-cost RAM which, on an input of length n, computes a binary representation of $a(n)$ and $s(n)$ in two registers within $a(n)$ steps.

Theorem 3.3 (Stockmeyer and Viskin [22], Theorem 3) *Let $(a(n), s(n))$ be suitable. Then $\Sigma_{a(n)}\mathsf{SPACE}[s(n)]$ for Turing machines is exactly the class of languages recognized by unit-cost CRCW PRAMs running in $a(n)$ time and using $2^{O(s(n))}$ processors.*

There are only two caveats in applying the above to obtain Theorem 3.2. First, the simulation is only claimed to work for the traditional Turing machine model. Luckily for us, Stockmeyer and Viskin's proof technique is extremely general, and extends to random access TMs rather easily. (In fact, there is no mention of specific Turing machine components in their argument. There are only references to the size of machine configurations, which is $O(s(n))$ in both models.)

The second caveat is that Stockmeyer and Vishkin's CRCW PRAMs are unit-cost. We show that, in fact, this condition can be relaxed to logarithmic-cost. With these two caveats removed, Theorem 3.2 immediately follows.

Theorem 3.4 *Let $\{C_n\}$ be a $O(\log Z(n))$-space uniform circuit family of depth $d(n) \geq n$ and size $Z(n) \geq n$. Then $\{C_n\}$ can be simulated by a logarithmic-cost CRCW PRAM running in time $O(d(n))$ and having $O(Z(n))$ processors.*

Proof. Our proof is in the spirit of Stockmeyer and Vishkin [22, Theorem 2]. They show how a unit-cost CRCW PRAM can perform the simulation; we extend their approach to work in the log-cost setting. At a high level, their PRAM introduces a memory location for each gate in C_n, and a processor for each wire of C_n. (Hence each processor deals with only two memory locations.) Each particular processor "waits" for some time, corresponding to the depth of the lowest-depth gate connected to the corresponding wire in C_n. Then the processor looks at the values of its memory locations, and possibly propagates values accordingly.

Our modification to their argument is simply to have the processors count in such a way that the logarithmic cost of decrementing a counter does not asymptotically affect their waiting time. Essentially everything else in our construction remains the same as theirs (except we deal with uniformity issues).

The CRCW PRAM is described as follows. On input x, let $n = |x|$. Gate i in C_n shall correspond to the ith global memory location g_i. All g_is are all initially blank, except for those corresponding to input symbols.

The PRAM first generates the description of the uniform circuit C_n in parallel, in a standard way (cf. [5, p.34–35]). This can be done in $(\log |C_n|)^{O(1)}$, or polylog time. It then assigns a processor $P_{i,j}$ to each wire (i, j) in the circuit C_n, where i is the gate of lesser depth (the longest path from an input symbol to i is less than the longest path from an input to j). At its start, $P_{i,j}$ loads i and j into registers and determines the depth d of i. (To ensure this can be done quickly, one could encode i such that d is a prefix in i's binary encoding.) Next, $P_{i,j}$ computes $\log d$, and with it, $d/\log d$ as well. Observe these can be computed from a binary representation of d in $(\log d)^{O(1)}$ time. Then $P_{i,j}$ counts up to $d/\log d$, by repeatedly decrementing 1 from the register that initially holds $d/\log d$. The point is that this decrementing stage takes $\Theta(d)$ time on a logarithmic cost RAM. In particular, subtracting a register with integer j from a register with integer i costs $\log i + \log j$ time on a log-cost RAM, so the decrementing takes $\sum_{i=0}^{(d/\log d)-1} \log(\frac{d}{\log d} - i)$ time, or $\log(\prod_{i=0}^{(d/\log d)-1}(\frac{d}{\log d} - i)) = \log((\frac{d}{\log d})!) = \Theta(d)$. Since this decrementing takes $\Theta(d)$ time, $P_{i,j}$ waits just long enough (within a constant factor) for an answer to arrive at memory location g_i.

Without loss of generality, assume gate i is an OR and gate j is an AND—the proof is symmetric for the other three cases. After the $\Theta(d)$ time delay, $P_{i,j}$ checks if a 1 was written to g_i. (If gate i was instead an AND gate, $P_{i,j}$ would have checked for a blank.) If a 1 was written, $P_{i,j}$ writes nothing to g_j. (If j was an OR, $P_{i,j}$ would have written a 1.) Arguing inductively, we find that that no write occurs to (the OR gate) g_i iff gate i gets set to 0 in the circuit evaluation, and no write occurs to (the AND gate) g_j iff gate j gets set to 1. NOT gates are handled in the obvious way, by "complementing" the memory location. □

It is instructive to compare the above corollary to Mak [13], who showed that every log-cost time t RAM can be simulated by an alternating log-cost RAM in $O(t \log\log t/\log t)$ time. His proof mimics ideas from the time versus space work of Halpern *et al.* [6] on pointer machines. However, like all other work we have found in this direction, his simulation requires a superpolynomial number of processors, hence the resulting circuit requires at least this many gates.

4 Conclusion

We have of course given only a modest partial answer to the problem of parallelizing deterministic time RAMs with polynomial circuits, as our circuits need semi-unbounded fan-in. One question left open by our work is if our simulation is an artifact of the random access TM model, or if it is a truly model-independent construction. Given the high-level nature of the result, we believe the latter to be the case. It appears that our parallel simulation works as long as the simulating machine has the same register-access powers as the simulated machine.

Another next step would be to find bounded fan-in circuits with similar size-depth parameters, if possible. This would imply, among other things, that the simulation can also be performed with a CREW (Concurrent Read, Exclusive Write) PRAM. Obtaining such a result may be difficult using standard techniques. In particular, there are hard limitations on how well the most natural ways of parallelizing deterministic time can do, cf. Greenlaw, Hoover and Ruzzo [5, p.68–69], and Lengauer and Tarjan [12].

However, we still believe that a bounded fan-in simulation is in the realm of possibility. Since our simulation differs from others, it is possible that a combination of our ideas with a simulation based on $\mathsf{DTIME}[t] \subseteq \mathsf{ATIME}[t/\log t]$ could yield a better size-depth tradeoff in the unbounded fan-in case. More precisely, it is possible that the processing of a b-symbol block could be implemented in $O(b/\log b)$ space (along the lines of [7]), without hurting the parameters of our simulation. For this reason, we conjecture that our polysize circuit simulation of $\mathsf{DTIME}[t]$ can be improved to have depth $\frac{t}{(\log t)\cdot(\log\log t)}$.

Conjecture: $\mathsf{DTIME}[t] \subseteq \Sigma_{t/((\log t)(\log\log t))}\mathsf{SPACE}[\log t]$.

If true, the conjecture would indeed imply a simulation of time t with uniform bounded fan-in polysize circuits of $O(t/\log\log t)$ depth, due to the following proposition of Ruzzo:

Proposition 1 (Ruzzo [20], Proposition 1) $\Sigma_{a(n)}\mathsf{SPACE}[s(n)] \subseteq \mathsf{ATISP}[s(n)\cdot(s(n)+a(n)), s(n)]$.

Hence if the conjecture is true, it would follow that $\mathsf{DTIME}[t] \subseteq \Sigma_{t/((\log t)(\log\log t))}\mathsf{SPACE}[\log t] \subseteq \mathsf{ATISP}[(\log t)^2 + t/\log\log t, \log t]$. Such a result would imply the existence of $o(t)$ bounded fan-in polysize circuits for deterministic time t.

5 Acknowledgements

The author appreciates the very useful commentary from the SPAA'05 and TOCS referees.

References

[1] A. Borodin. On relating time and space to size and depth. *SIAM J. Computing* 6(4):733–744, 1977.

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

[3] A. K. Chandra, L. Stockmeyer, and U. Vishkin. Constant Depth Reducibility. *SIAM J. Comput.* 13(2):423–439, 1984.

[4] P. W. Dymond and M. Tompa. Speedups of Deterministic Machines by Synchronous Parallel Machines. *J. Comput. Syst. Sci.* 30(2):149–161, 1985.

[5] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory.* Oxford University Press, 1995.

[6] J. Y. Halpern, M. C. Loui, A. R. Meyer, and D. Weise. On Time versus Space III. *Mathematical Systems Theory* 19(1): 13–28, 1986.

[7] J. Hopcroft, W. J. Paul, and L. G. Valiant. On Time vs. Space. *J. ACM* 24(2):332–337, 1977.

[8] N. Immerman. Nondeterministic space is closed under complement. *SIAM J. Computing* 17:935–938, 1988.

[9] N. Immerman. *Descriptive Complexity.* Springer-Verlag, 1999.

[10] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science*, J. van Leeuwen (ed.), Elsevier, 1990.

[11] R. J. Lipton and A. Viglas. Non-uniform Depth of Polynomial Time and Space Simulations. In *Fundamentals of Computation Theory*, LNCS 2751:311–320, Springer, 2003.

[12] T. Lengauer and R. E. Tarjan. Asymptotically tight bounds on time-space trade-offs in a pebble game. *J. ACM* 29(4):1087–1130, 1982.

[13] L. Mak. Are Parallel Machines Always Faster than Sequential Machines? (Preliminary Version). In *Proceedings of STACS*, LNCS 775:137–148, Springer, 1994.

[14] L. Mak. Parallelism Always Helps. *SIAM J. Comput.* 26(1):153–172, 1997.

[15] F. Meyer auf der Heide. Speeding up Random Access Machines by Few Processors. In *Proceedings of STACS*, LNCS 210:142-152, Springer, 1986.

[16] C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[17] M. Paterson and L. G. Valiant. Circuit Size is Nonlinear in Depth. *Theor. Comput. Sci.* 2(3):397–400, 1976.

[18] W. J. Paul and R. Reischuk. On Alternation II. A Graph Theoretic Approach to Determinism Versus Nondeterminism. *Acta Inf.* 14:391–403, 1980.

[19] W. J. Paul and R. Reischuk. On Time versus Space II. *J. Comput. Syst. Sci.* 22(3):312–327, 1981.

[20] W. Ruzzo. On uniform circuit complexity. *J. Comput. and Syst. Sci.*, 22:365–383, 1981.

[21] M. Sipser. Introduction to the Theory of Computation (2nd ed). PWS Publishing Company, 2005.

[22] L. Stockmeyer and U. Vishkin. Simulation of Parallel Random Access Machines by Circuits. *SIAM J. Comput.* 13(2):409–422, 1984.

[23] R. Szelepcsenyi. The method of forcing for nondeterministic automata. *Bulletin of the EATCS* 33:96–100, 1987.

[24] H. Venkateswaran. Properties that Characterize LOGCFL. *J. Comput. Syst. Sci.* 43(2):380–404, 1991.

[25] R. Williams. Parallelizing Time With Polynomial Circuits. In *Proc. of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'05)*, 171–175, 2005.