JOURNAL OF COMPUTER AND SYSTEM SCIENCES 22, 365–383 (1981)

On Uniform Circuit Complexity*

WALTER L. RUZZO

Department of Computer Science, University of Washington, Seattle, Washington 98195

Received August 10, 1979; revised December 1, 1980

We argue that uniform circuit complexity introduced by Borodin is a reasonable model of parallel complexity. Three main results are presented. First, we show that alternating Turing machines are also a surprisingly good model of parallel complexity, by showing that simultaneous size/depth of uniform circuits is the same as space/time of alternating Turing machines, with depth and time within a constant factor and likewise log(size) and space. Second, we apply this to characterize NC, the class of polynomial size and polynomial-in-log depth circuits, in terms of tree-size bounded alternating TMs and other models. In particular, this enables us to show that context-free language recognition is in NC. Third, we investigate various definitions of uniform circuit complexity, showing that it is fairly insensitive to the choice of definition.

1. INTRODUCTION

This paper is principally motivated by an interest in computational complexity of highly parallel computers. Recently, a variety of abstract models of such machines have been proposed [6, 10, 13, 20, 24, 26]. For several problems, very striking speedups are possible over conventional models. For example, on vector machines we can do $n \times n$ Boolean matrix multiplication in $\log n$ steps, transitive closure or context free language recognition in $\log^2 n$, or satisfiability in linear time [19, 20, 22, 23]. Such exponential speedups serve to emphasize the potential utility of parallelism, but the results are unsatisfying in two respects.

The first problem is that we have ignored an important factor—hardware size. For example, the linear time satisfiability algorithm requires exponentially long vectors. This seems just as impractical as an exponential time algorithm. Clearly, a measure of both hardware size and time is needed.

The second problem is that the multiplicity of models makes meaningful comparisons of results difficult. The commonly used "unit cost" measure is particularly suspect. Goldschlager's Conglomerates [12] are a very useful step toward clarifying this situation. We feel our work extends his by providing a more systematic treatment of parallel time complexity with a simultaneous restriction on hardware size.

* This material is based upon work supported by the National Science Foundation under Grant MCS77-02474.

0022-0000/81/030365-19$02.00/0

The combinational circuit model seems to address both the issues of size and time cost. Circuit depth provides the "time" measure. Size of circuits (number of gates) is a clear reflection of the hardware costs (although it may be somewhat pessimistic when depth is large—see [8]). Other models can be translated into circuits without too much difficulty, so comparisons are possible.

One difficulty with the combinational circuit model is that it is non-uniform: unlike most machine models, where one machine handles problems of all sizes, we must provide a different circuit (perhaps radically different) for each different size input. A *uniform* family of circuits for some problem will be a family of circuits for which there is a computationally simple rule for constructing the various circuits. A practical motivation for restricting attention to uniform families is given in the following example.

Suppose we are given some variety of general purpose parallel machine M on which some problem A of interest is efficiently solvable by some algorithm P. In the foreseeable future, any implementation of M will be made from logic gates of some modest fan-in (say, 2). Consider the computation of algorithm P on M for all problem instances of size n. We can "unroll" these computations into some combinational circuit c_n which represents just the subset of M's hardware which comes into play for inputs of size n. The depth of c_n seems an honest measure of the (worst case) running time of M, and the size of c_n likewise measures the amount of hardware used in obtaining the running time. Clearly, a lower bound on the complexity of circuits for A is a lower bound for any algorithms for A on machine M. We can say even more: c_n must be "easily" computable from n, since in some sense, c_n is "constructed" by M in real time. Further, for most machines M it seems reasonable to suppose that the design of one version easily generalizes to larger versions, e.g., vector machines with million-bit vectors are not much harder to describe than ones with thousand-bit vectors. Thus the construction of c_n seems no harder than the construction of M, which presumably is not too hard if M is a feasible machine architecture. Hence, bounds on the complexity of such "easily" constructible circuits are bounds on A. Perhaps more importantly, making the circuits "uniform in n" means that *upper* bounds on the circuit complexity give upper bounds for other models, too. Theorem 4 is one example of this.

This notion of uniform circuit complexity was suggested by Borodin [2]. A dual notion due to Schnorr [25] makes the machines non-uniform instead, by giving them "oracles" depending on n. Our results apply equally well in this setting, as will be shown in Section 5.

We are particularly interested in the class NC of functions computable by uniform circuits of polynomial size and "poly-log" (i.e., $(\log n)^{O(1)}$) depth. Pippenger first identified NC as a class of interest, and obtained a surprising characterization of it [18] discussed below. NC is Cook's [5] mnemonic for "Nick's Class" in recognition of this contribution.

There are several reasons for interest in NC. First, this class seems to encompass the functions for which dramatic speedups are possible on feasibly constructible parallel machines. Second, poly-log time may not be achievable in some applications, say, due to machine architecture or I/O bottlenecks. However, existence of such rapid parallel algorithms for these problems implies that they are somehow decomposable into a large number of nearly independent subproblems. This degree of decomposability suggests that these problems may admit a wide variety of efficent parallel implementations, for example, covering a variety of parallel computer architectures, or allowing a broad spectrum of compromises between time and number of processors. Third, some problems of great practical importance are known to be in NC. One of the contributions of this paper is to develop some new tools for showing membership in NC, and to apply these tools to other natural problems.

Borodin has shown a large class of natural problems to be in NC, namely the class NSPACE($\log n$) of problems solvable in nondeterministic space $O(\log n)$, plus the class co-NSPACE($\log n$) of problems whose complements are in NSPACE($\log n$) [2]. This class encompasses the bulk of the problems currently known to be in NC. It includes a wide variety of graph-theoretic problems which depend on existence of paths in graphs, such as transitive closure, bi- and k-connectivity, and shortest path. Other important problems such as sorting, matrix multiplication, and pattern matching are also in this class, in fact in DSPACE($\log n$). Only a few problems are known to be in NC but not known to be in (co-)NSPACE($\log n$). Until recently, matrix inversion and related problems [7] were the principal examples.

A natural question about uniform circuit complexity is raised by the following correspondence between Turing machines and circuits. It is known that Turing machine *time* is polynomially related to circuit *size*, and likewise TM *space* to *depth*. Specifically [2, 15, 16],

$$\begin{aligned} \mathrm{DTIME}(T) &\subseteq \text{uniform size } (T \log T) \subseteq \mathrm{DTIME}(T \log^3 T), \\ \mathrm{NSPACE}(S) &\subseteq \text{uniform depth}(S^2) \subseteq \mathrm{DSPACE}(S^2). \end{aligned} \tag{1}$$

Borodin [2] raised the question of whether *simultaneous* time and space on deterministic TMs correspond to simultaneous size and depth of circuits. In particular, Cook [5] defined[1] SC, the analog of NC, namely, the class of languages recognizable by deterministic TMs in polynomial time and poly-log space, and asked whether NC = SC. Transitive closure is in NC [2], and deterministic context-free languages are in SC [5], but Cook suggested that neither problem was in the other class, hence NC $\neq$ SC.

This paper contains three main results. One is that context-free languages *are* in NC. Some other interesting problems are shown to be in NC as well. These include the tree isomorphism problem and various "dynamic programming" problems such as ordering matrix multiplies and finding optimal binary search trees. These are among the few known examples of natural problems in NC which are not known to be in

[1] Cook's term was PLOPS; SC is Pippenger's mnemonic for "Steve's Class," in recognition of the contribution of [5].

(co-)NSPACE($\log n$). We believe these results serve to highlight the importance of the NC class, since its study may lead to practical highly parallel algorithms for compiling and other important problems. This result, based on a simulation of tree-size bounded ATMs [22] and the characterization of NC given in Section 3 below, is presented in Section 4.

(We remark that there is still evidence to support Cook's conjecture that NC $\neq$ SC. Specifically, at the risk of mnemonic confusion, we define the Narrow (Shallow) Circuit Value problem NCV (SCV) as the restriction of the circuit value problem [14] to circuits of polylog width (depth). Then SCV $\in$ NC and NCV $\in$ SC, but Paul and Tarjan [21] give evidence based on pebbling that SCV $\notin$ SC and Pippenger [18] gives similar evidence based on pebbling with auxiliary pushdowns that NCV $\notin$ NC.)

Returning to the question raised by (1) above, Pippenger [17] has previously obtained a very interesting characterization of NC in terms of simultaneously restricted Turing machine resources, but not time and space as expected. Instead he has shown that NC equals polynomial time and poly-log *head reversals*.

Our second main result, given in Section 3, is to give a new characterization of NC. We show that simultaneous size and depth do correspond to simultaneous space and time, but of *alternating* TMs. More precisely uniform circuit size $Z(n)$ and depth $D(n) =$ alternating space $\log Z(n)$ and time $D(n)$. Note that depth becomes ATM time, not space as with DTMs. Thus the previously observed time/size and space/depth relationships are a manifestation of the known DTM-time/ATM-space and DTM(NTM)-space/ATM-time relations. As [12] illustrates, a polynomial connection between time on two different parallel models is common. Equivalence within a constant factor while simultaneously restricting size/space suggests a more fundamental connection. It is only a slight exaggeration to say that an ATM *is* a uniform circuit and vice versa.

In Section 4 we also give characterizations of NC in terms of other alternating TM resources, and in terms of auxiliary pushdown machines [4]. These results, like [17], have the virtue of providing radically different frameworks in with to explore the development of efficient parallel algorithms.

Our third main result is of a more technical nature. Part of the definition of uniform circuits is the uniformity condition, which specifies what we mean by an "easily constructible" circuit. Nearly a dozen different definitions have appeared in the literature, but little attempt has been made to relate them. In Section 2 we consider several uniformity conditions and show that the circuit complexity of a language is relatively insensitive to the choice of uniformity condition. NC is particularly robust, being unchanged through a very wide range of definitions. At the "low-end" of NC, however, i.e., for depths $o(\log^2 n)$, the definition is more critical. We propose definitions which are more suitable for this complexity range.

2. Uniform Circuits: Definitions and Robustness

We assume familiarity with deterministic and non-deterministic Turing machines (DTMs and NTMs, resp.). We will also be using *alternating Turing machines* (ATMs) [6].

ATMs are a generalization of nondeterministic Turing machines described informally as follows. The states are partioned into "existential" and "universal" states. As with an NTM, we can view a computation of an ATM as a tree of configurations. A tree is a *computation tree* of an ATM M on a string w if its nodes are labeled with configurations of M on w, such that the descendants of any non-leaf labeled by a universal (existential) configuration include all (resp., one) of the successors of that configuration. A *full* computation tree includes all successors of each non-leaf node. A computation tree is *accepting* if it is finite and all the leaves are accepting configurations. *M accepts w* if there is an accepting computation tree whose root is labeled with the initial configuration of M on w. Notice that for ATMs with only existential states, acceptance is essentially the same as for NTMs. We assume that ATMs have an end-marked, read-only input tape.

We will use a "random access input" variation of ATMs similar to that defined in [6]. In this model the ATM has no "input head." Instead it has a special "index" tape and a special "read" state. Whenever it enters the read state with "a, i" written on the index tape, it halts, and accepts if and only if the ith input symbol is "a." It is not hard to show that there is only a constant loss in space and time when converting to this normal form, assuming space and time at least $\log n$.

An ATM uses *time* $T(n)$ (*space* $S(n)$) if for all accepted inputs of length n there is an accepting computation tree of height $\leqslant T(n)$ (each of whose nodes is labeled by a configuration using space $\leqslant S(n)$). As usual, we denote the class of languages accepted by ATMs within space $O(S(n))$ by $ASPACE(S(n))$, and similarly for NSPACE, DSPACE, ATIME, NTIME, and DTIME. Likewise, $ASPACE, TIME(S(n), T(n))$ denotes languages accepted by ATMs running in space $O(S(n))$ and time $O(T(n))$ simultaneously, and similarly for other simultaneous pairs of resource bounds. More formal definitions of ATMs, configurations, etc., may be found in [6].

A *combinational circuit* is a directed acyclic graph, where each node (gate) has indegree $d \leqslant 2$, and is labeled by some Boolean function of d variables, or has indegree 0 and is labeled by "x" (an *input*). Nodes with outdegree 0 are *outputs*. We will mainly consider circuits with one output, although the results can be generalized easily.

Throughout this paper, we will be considering a *family* $C = (c_1, c_2, \ldots)$ of circuits, where c_n has n inputs and one output. We will assume the gates of c_n are numbered so that gate 0 is the output and gates $1, \ldots, n$ are the inputs. We further restrict the gate numbering so the largest gate number is $(Z(n))^{O(1)}$, where $Z(n)$ is the size of c_n, so the gate numbers coded in binary have length $O(\log Z(n))$.

The family C *recognizes* $A \subseteq \{0, 1\}^*$ if for each n, c_n recognizes $A^{(n)} = A \cap \{0, 1\}^n$, i.e., the value of c_n on input $x_1, \ldots, x_n \in \{0, 1\}$ is 1 iff $x_1 \cdots x_n \in A$. If c_n has at most

$Z(n)$ gates and depth $T(n)$, then the *size and depth complexity* of C is $Z(n)$ and $T(n)$. A *language* A is of size and depth complexity $Z(n)$ and $T(n)$ if there is a family of circuits of the corresponding complexity which recognizes A.

Circuit complexity as defined above has the obvious problem that there are arbitrarily difficult languages with trivial circuit complexities. For example, take *any* language L, even a nonrecursive one, where for each n either all or no strings of length n are in L. Borodin [2] has suggested avoiding this problem by considering only families of circuits which are uniformly constructible, i.e., c_n is "easily" determined from n.

The definition of "easy" obviously affects the circuit complexity of problems. If our uniformity condition is too strong, i.e., we demand that the circuits be very easy to compute, then the circuits may correspond to unrealistically simple parallel computers. On the other hand, if it is too weak, we may trivialize the theory by putting most of the computational power into the circuit constructor, rather than the circuit. The example in the preceding paragraph illustrates this effect. In this case, the depth of the circuit does not reveal anything about the complexity of the language. Note, however, that this phenomenon sets in far below the level of non-recursive sets:

EXAMPLE 1. For any $T(n) \geqslant n$ and any A in DSPACE($T(n)$) we can construct in space $T(n)$ a circuit recognizing $A^{(n)}$ of depth at most $T(n)$, in fact one of depth n. This can be done since every n-ary Boolean function has a disjunctive normal form expression of depth at most n, which can be computed in space $T(n)$ by testing all strings of length n for membership in $A^{(n)}$.

One of the difficulties with formulating a useful notion of uniformity is that there are a myriad of plausible definitions, between the two extremes mentioned above, but no concrete basis for choosing one over antoher. We feel one of our contributions is to show that uniform circuit complexity is fairly insensitive to changes in the definition of uniformity. For example, NC is identical under a wide range of uniformity conditions. This leaves us free to choose those definitions which are easiest to apply, and/or give the sharpest theorems. This is exactly what we will do in Section 3. In the remainder of this section, we will consider five different definitions of uniformity and illustrate that the class of uniform circuits is relatively insensitive to the choice of definition.

DEFINITION. Let $C = (c_1, c_2, \ldots)$ be a family of circuits. The *standard encoding* $\bar{c}_n$ of c_n is the string of 4-tuples of the form $[g, t, g_L, g_R]$ where gate number g's left(right) input is the output of gate number g_L (g_R) and g is a t-gate, $t \in \{x, \wedge, \vee, \neg, \ldots\}$ (g_R omitted for $\neg$ gates, etc.).

(Gates need not be numbered or listed in topological order, as is sometimes required).

DEFINITION. The family $C = (c_1, c_2, \ldots)$ of size and depth complexity $Z(n)$, $T(n)$ is *U_B-uniform* (*U_{BC}-uniform*) if the mapping $n \rightarrow \bar{c}_n$ is computable by a DTM in space $T(n)$ (resp. space $\log Z(n)$). Also, for $X = U_B$ or U_{BC}, let *X-SIZE, DEPTH*$(Z(n)), T(n))$ denote the class of languages recognizable by X-uniform circuits of size and depth complexity $O(Z(n))$ and $O(T(n))$.

U_B is the definition of Borodin [2]. U_{BC} is the definition of Borodin and Cook [5]. Note that $T(n)$ is always $\geqslant \log Z(n)$, so U_{BC}-uniform implies U_B-uniform.

Note that in this and subsequent definitions of uniformity, the complexities of the "uniformity TMs" are given in terms of n, the number of circuit inputs, rather than the more customary convention using the length of the TM's input. For our applications, this parameterization in terms of "n" seems more natural.

Cook's definition of NC is the following, although we will see that the precise definition usually is not critical (although $\mathrm{NC}^{(k)}$ for $k < 2$ is more fragile).

DEFINITION.

$$\mathrm{NC} = U_{BC}\text{-SIZE, DEPTH}(n^{O(1)}, \log^{O(1)} n),$$
$$\mathrm{NC}^{(k)} = U_{BC}\text{-SIZE, DEPTH}(n^{O(1)}, \log^{k} n)$$

Defining uniformity in terms of the complexity of constructing the standard encoding is very natural. However, it is quite inconvenient for our purposes since our simulations will have neither the time to generate the standard encoding, nor the space to store it. Therefore, we take a different, but we feel equally natural, approach to defining uniformity. Namely, we will define it on terms of the complexity of computing *local* connection properties of the circuit, rather than the global connection pattern given by the standard encoding. This approach is essentially that take by Goldschlager [12], although ours differs in certain technical ways.

Let $C = (c_1, \ldots)$ be a family of circuits. If g is any gate in c_n and $p \in \{L, R\}^*$, let $g(p)$ denote the gate reached by following the path p of inputs to g. For example, $g(\varepsilon)$ is g, $g(L)$ is g's left input, $g(LR)$ is g's left input's right input, etc.

DEFINITION. The *direct connection language* of the family C, L_{DC} is the set of strings of the form $\langle n, g, p, y\rangle$ $n, g \in \{0,1\}^*$, $p \in \{\varepsilon, \mathrm{L}, \mathrm{R}\}$, $y \in \{x, \wedge, \vee, \neg, \ldots\} \cup \{0,1\}^*$ such that in c_n either (*i*) $p = \varepsilon$ and gate g is a y-gate, $y \in \{x, \wedge, \vee, \neg, \ldots\}$, or (ii) $p \neq \varepsilon$ and gate $g(p)$ is numbered y, $y \in \{0,1\}^*$. The *extended connection language* L_{EC} is as above, except $p \in \{\mathrm{L}, \mathrm{R}\}^*$ and $|p| \leqslant \log Z(n)$.

Thus the direct connection language encodes the type of each gate, and the names of its immediate predecessors. The extended connection language encodes this information for all predecessors within distance $\log Z(n)$. We include the direct connection language for its simplicity, but we will mainly use the extended language. The extended connection language is obviously the harder of the two to recognize, but it turns out to be easy enough for our purposes, and the extra flexibility provided by having paths of length > 1 will be useful.

DEFINITION. The family $C = (c_1, c_2, \ldots)$ of size and depth $Z(n)$, $T(n)$ is

U_D-uniform if there is a DTM recognizing L_{DC} which on inputs of the form $\langle n, -, -, -\rangle$ takes time $O(\log Z(n))$. Similarly, C is *U_E-uniform* (*U_{E^*}-uniform*) if there is a DTM (ATM) recognizing L_{EC} in time $\log Z(n)$ (resp. time $O(T(n))$ and space $O(\log Z(n))$).

Again, note that the TM complexity is parameterized by the number of *circuit* inputs, not the length of the TM's inputs.

U_E and U_{E^*} are the two definitions we will use in Section 3. Of our definitions, these give the sharpest forms of our theorems. We also feel that they are no harder to use than the other definitions. We invite the reader to convince himself that, say, the standard size n^3, depth $\log n$ Boolean matrix multiplication circuit is U_E-uniform.

Some discussion of the five definitions is now in order. (See Fig. 1.) Our efficient simulation of circuits in the next section hinges on being able to rapidly follow a path of moderate length ($\log Z(n)$) in the circuit. Uniformity condition U_E is the strongest of the definitions in this respect. Intuitively, it says that a circuit is U_E-uniform only if a DTM can follow such a path using an average of $O(1)$ time for each step in the path. Condition U_D is weaker (i.e., fits more circuits). It fallows the DTM to use time $O(\log Z)$ per step. Condition U_{BC} is weaker still, allowing time polynomial in Z per step but still within space $\log Z$. Condition U_B even allows non-polynomial time (if $T = \omega(\log Z)$). Condition U_{E^*} significantly weakens U_E in a different way. Namely, it allows the path-following to be done by an ATM rather than a DTM, and also allows more time (T versus $\log Z$), but again within $\log Z$ space.

The difficulty of simulating circuits uniform according to the various definitions increases roughly in the way indicated above, but not nearly as drastically as might be expected. The following lemmas show this.

The first lemma shows that the connection languages have exactly the same space complexity as the standard encoding function.

LEMMA 1. *For $S(n) = \Omega(\log Z(n))$ the following statements are equivalent*:

(1) *$n \to \bar{c}_n$ is computable in* DSPACE($S(n)$).

(2) *L_{EC} is recognizable in* DSPACE($S(n)$).

(3) *L_{DC} is recognizable in* DSPACE($S(n)$).

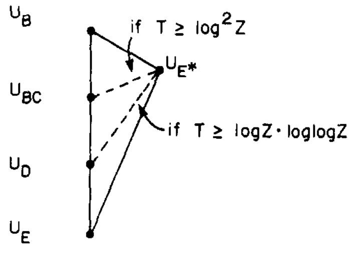

FIG. 1. Relationships among uniformity conditions.

Proof. (1) $\Rightarrow$ (2): to decide whether $\langle n, g, p, y\rangle \in L_{\mathrm{EC}}$ simulate the DTM which computes $n \to \bar{c}_n$ until it outputs $[g, t, g_{\mathrm{L}}, g_{\mathrm{R}}]$ for some $t, g_{\mathrm{L}}, g_{\mathrm{R}}$. Then if $p = \varepsilon$ accept if $t = y$. Otherwise let $p = Hp'$ with $H \in \{\mathrm{L}, \mathrm{R}\}$. If $p' = \varepsilon$ then accept if $g_H = y$, otherwise restart as if the input were $\langle n, g_H, p', y\rangle$.

(2) $\Rightarrow$ (3): immediate.

(3) $\Rightarrow$ (1): we can test whether there is a gate numbered g in c_n by testing whether there is some $y \in \{x, \wedge, \vee, \ldots\}$ for which $\langle n, g, \varepsilon, y\rangle \in L_{\mathrm{DC}}$. Given that g exists, we find, e.g., its left input by testing $\langle n, g, L, g_{\mathrm{L}}\rangle \in L_{\mathrm{DC}}$ for $g_{\mathrm{L}} = 0, 1, 2, \ldots$. The quadruple $[g, y, g_{\mathrm{L}}, g_{\mathrm{R}}]$ can then be added to the output. This process is repeated for $g = 0$ to maxgateno, where maxgateno is initially 0 and is increased whenever a g_{L} or g_{R} is discovered as above which is larger than the current maxgateno. Since gate 0 is the unique "sink" in the graph, this process will eventually find all gates. ∎

We believe use of the connection languages rather than the standard encoding will simplify proofs of uniformity in many cases. Lemma 1 serves to show that the two formalisms are equivalent. For example,

COROLLARY. *C is U_{BC}-uniform iff L_{DC} and L_{EC} have space complexity* $\log Z(n)$.

The next lemma gives a fast alternating time acceptor for L_{EC}, given one for L_{DC}.

LEMMA 2. *If* $L_{\mathrm{DC}} \in$ ASPACE, TIME($\log Z(n)$, $R(n)$) *then* $L_{\mathrm{EC}} \in$ ASPACE, TIME($\log Z(n)$, $R(n) + \log Z(n) \cdot \operatorname{loglog} Z(n)$).

Proof (Sketch). Use a divide and conquer method similar to Savitch's theorem: guess the gate number at the middle of the path, and recursively verify both halves of the path. At the bottom level, verify a one step path using L_{DC}. The path length is $\log Z$, so the number of levels of recursion is $\log \log Z$. Each level takes time $\log Z$ to write the guessed gate name and update the path. ∎

These results and known time/space complexity results give the following relationships between the various uniformity conditions.

THEOREM 1 (see Fig. 1). *For any family $C = (c_1, \ldots)$ of circuits*

(1) *U_{E}-uniform $\Rightarrow$ U_{D}-uniform $\Rightarrow$ U_{BC}-uniform $\Rightarrow$ U_{B}-uniform.*

(2) *U_{E}-uniform $\Rightarrow$ U_{E^*}-uniform $\Rightarrow$ U_{B}-uniform.*

(3) *if $T(n) \geqslant \log^2 Z(n)$ then U_{BC}-uniform $\Rightarrow$ U_{E^*}-uniform.*

(4) *if $T(n) \geqslant \log Z(n) \cdot \log \log Z(n)$ then U_{D}-uniform $\Rightarrow$ U_{E^*}-uniform.*

Note that for any family of circuits $T(n) \geqslant \log Z(n)$ due to the indegree 2 restriction. Thus the conditions on (3) and (4) are restrictive, but not overly restrictive.

This result deals with a particular family of circuits. A more interesting question is the circuit complexity of a given *language* with respect to the various definitions.

Since Theorem 1 shows that U_E is the strongest of the definitions, we will show that there is little or no complexity increase in converting to a U_E-uniform circuit.

THEOREM 2. *Let $Z(n)$ and $T(n)$ be any functions such that $Z(n) = n^{\Omega(1)}$, $T(n) = \Omega(\log n)$ and $\log Z(n)$ and $T(n)$ are computable by a DTM given input n (in binary) in time $O(\log Z(n))$. Then for $X = U_{BC}$, U_D, or U_{E^*}, if language A has X-uniform size and depth complexity $Z(n)$ and $T(n)$, then it has U_E-uniform complexity:*

(1) $Z^{O(1)}(n)$ *and* $O(T(n))$, *if* X *is* U_{E^*},
(2) $Z^{O(1)}(n)$ *and* $O(\max(T(n), \log Z(n) \cdot \log\log Z(n)))$, *if* X *is* U_D,
(3) $Z^{O(1)}(n)$ *and* $O(\max(T(n), \log^2 Z(n)))$, *if* X *is* U_{BC}.

Proof. Part (1) is a simple corollary of the results we will present in Section 3. Parts (2) and (3) follow from (1) and Theorem 1 by (if necessary) padding the circuits' depths sufficiently to make them U_{E^*}-uniform. The constructibility condition on Z and T derives from the one in Theorem 3. Actually the exact computation of $\log Z(n)$ is unnecessary; if suffices to be able to construct any function which is $O(\log Z(n))$.

The theorem shows that circuits uniform by any definition in the broad range between U_E (deterministic space and time $\log Z$) and U_{E^*} (alternating space $\log Z$ and time T) can be translated to U_E-uniform ones with at most a constant loss in depth, and a polynomial loss in size. Note that this includes the Borodin–Cook definition in the important case where $T = \Omega(\log^2 Z)$. Even when $T = o(\log^2 Z)$, the loss in depth is small. Furthermore, we have:

COROLLARY. NC *is the same class under definitions U_E, U_D, U_{BC}, and U_{E^*}. Further, for $k \geqslant 2$* $\mathrm{NC}^{(k)}$ *is identical under these four definitions.*

In fact "ASPACE-TIME($\log n$, $\log^{O(1)} n$)-uniform" still does not change NC. As we will see, NC is characterized by this alternating space-time class, so, loosely speaking, this statement asserts the closure of NC under "NC-uniformity."

Unfortunately U_B-uniformity does not seem to share these properties. All we can say is that U_B-uniform depth T is between alternating time T and deterministic space T [2]. The fact that it equals deterministic space T for $T \geqslant n$ (Example 1) makes us suspect that it is too weak a condition to use when size is of interest as well as depth. (Ref. [2] was only concerned with uniform depth.) Similarly, U_D and U_{BC} may be too weak to usefully define $\mathrm{NC}^{(k)}$ for $k < 2$. For example, as defined, $\mathrm{NC}^{(1)}$ is the class of circuits of log depth constructed by DTMs in log space. It is generally believed that log space is more powerful than uniform log depth, so this definition gives the undesirable situation of a circuit constructor which is (apparently) more powerful than the circuit being constructed. Either U_E or U_{E^*} seems preferable in this range.

3. Circuits and Alternating Turing Machines

Of the parallel machine models mentioned in Section 1, the one which probably has been studied the most is the alternating Turing machine. However, most of its applications to date have been in "traditional," rather than parallel complexity. As a parallel model, it seems to suffer the problems mentioned in Section 1 (and more). It is not obvious how to "implement" an ATM; nor how much hardware it would take; nor whether unit cost is appropriate when arbitrarily long tapes need to be duplicated at $\forall$- or $\exists$-branches; etc. Furthermore, tape memory as an architectural feature of a modern computer is quaint, at best. In spite of these apparent objections, our main theorems show that ATMs and uniform circuits are very closely related. Indeed, we might say that an ATM *is* a uniform circuit.

Theorem 3. *Assume $T(n)$ and $S(n) = \Omega(\log n)$ are computable by a DTM with input n (in binary) in time $O(S(n))$. Then over the input alphabet* $\{0, 1\}$

$$\text{ASPACE, TIME}(S(n), T(n)) \subseteq U_E\text{-SIZE, DEPTH}(2^{O(S(n))}, T(n)).$$

Proof. The circuit will have gates labeled (t, α) for $0 \leqslant t \leqslant T(n)$ and α a configuration of the ATM using space $S(n)$. The output gate is $(0, \alpha_0)$, where α_0 is the initial configuration. Normally, the gate type $(\wedge, \vee)$ is the same as for the configuration α $(\forall, \exists)$ and the inputs to (t, α) are $(t+1, \beta)$ such that $\alpha \vdash \beta$. One exception to this rule occurs when $t + 1 > T(n)$ or when β uses space $> S(n)$, in which case $(t + 1, \beta)$ is replaced by the constant 0. The other exception is when α contains the special "read" state with "a, i" on its index tape. In this case gate (t, α) is "$a \equiv$ input i."

Note that the depth of this circuit is exactly $T(n)$ and its size is $T(n)$ times the number of possible configurations within space $S(n)$ which is $2^{O(S(n))}$. (As with DTMs and NTMs, for ATMs $T(n) \leqslant$ number of possible configurations, hence $\leqslant 2^{O(S(n))}$.)

Consider the full computation tree of the ATM with all nodes at depth $\geqslant T(n) + 1$ and all nodes using space $\geqslant S(n) + 1$ considered to be rejecting. By definition the ATM accepts within time $T(n)$ and space $S(n)$ if and only if there is an accepting subtree of this modified computation tree (rooted at the initial configuration). It is easy to show by induction on t that a node labeled α at depth t in this computation tree is the root of an accepting subtree if and only if gate (t, α) has value 1 in our circuit. Thus the circuit recognizes the same language as the ATM.

With appropriate codings of gate names, it is not hard to see that L_{EC} is recognizable in time $O(S(n))$ on a DTM: given $\langle n, g, p, x \rangle$, decode g as (t, α), then simulate the ATM transitions starting at α, following path p. By assumption, $S(n)$ and $T(n)$ are computable given n in time $O(S(n))$, so the DTM can also recognize those cases where the path p reaches past depth $T(n)$ or space $S(n)$. ∎

We remark that the factor of $T(n)$ in the size of the circuit constructed above may be eliminated in the common case where the ATM explicitly halts within $T(n)$ steps, and is "non-looping"; i.e., $\alpha \vdash^+ \alpha$ is impossible. The resulting circuit has size (and width, [17]) equal to the number of configurations of the ATM in space $S(n)$.

THEOREM 4. *For* $T(n) = \Omega(\log n)$, *and* $Z(n) = n^{\Omega(1)}$

$$U_{E^*}\text{-SIZE, DEPTH}(Z(n), T(n)) \subseteq \text{ASPACE, TIME}(\log Z(z), T(n)).$$

Proof. Let $C = (c_1, c_2, ...)$ be the family of circuits recognizing A. We construct an ATM M which nondeterministically guesses successive gates on paths through a circuit c, uses its $\exists, \forall, \neg$ states to simulate $\vee, \wedge, \neg$ gates, and in parallel simulates a recognizer for L_{EC} to verify that c is c_n. M can "name" gates rapidly since e.g., the left input of $g(p)$ is $g(p \cdot L)$. Thus at most time $T(n)$ is needed to "traverse" a path in the circuit. Similarly, space $\log Z(n)$ is sufficient, since a name $g(p)$, where $|p| = \log Z$, can be shortened by guessing h, verifying $h = g(p)$, then replacing $g(p)$ by $h(\varepsilon)$.

This algorithm is specified more precisely by the following recursive "circuit value" procedure. CV(n, g, p) will accept if and only if gate $g(p)$ in circuit c_n has value 1. Assume that M' is an ATM accepting L_{EC} in time $T(n)$ and space $\log Z(n)$.

Step 1. If the length of $p = \log Z(n)$, guess an integer h, then split at a universal state doing both of the following; otherwise proceed to Step 2.

Step 1a. Check that $h = g(p)$, by running M' on $\langle n, g, p, h\rangle$.

Step 1b. Set g to h, p to ε and proceed with the following.

Step 2. Guess the type $t \in \{x, \wedge, \vee, \neg, ...\}$ of gate $g(p)$, then universally do both of the following.

Step 2a. Check that $g(p)$ is of type t by guessing h, then running M' on both $\langle n, g, p, h\rangle$ and $\langle n, h, \varepsilon, t\rangle$.

Step 2b. Evaluate gate $g(p)$ by:

(i) If $t =$ "x" then guess $1 \leqslant h \leqslant n$, check that $g(p) = h$, read input h and halt, accepting if and only if it was a 1.

(ii) Otherwise, recursively evaluate gate $g(p)$; e.g., if $t =$ "$\wedge$" then do CV$(n, g, p \cdot L)$ *and* CV$(n, g, p \cdot R)$.

It is easy to show by induction on the depth of gate $g(p)$ that CV(n, g, p) accepts if and only if $g(p)$ has value 1. The complete simulation begins by guessing and verifying n, then calling CV$(n, 0, \varepsilon)$, which accepts if and only if the circuit has output 1.

The space used is easily seen to be $O(\log Z(n))$. The timing analysis is a little trickier. First, note that Steps 1a, 2a, and 2b(i) are *not* recursive. Thus they contribute to the total running time of M by at most an *additive* term of $T(n)$, $\log Z(n) + T(n)$ or $T(n) + \log n$, respectively. The maximum depth of recursion is $T(n)$, since Step 2b(ii) reduces the depth of the gate being considered by at least one per recursive level. Step 2b(ii) takes time $O(1)$ per recursive level, hence $O(T(n))$ in total. Step 1 (excluding 1a) takes time $O(\log Z(n))$ on one out of every $\log Z(n)$ consecutive levels and takes time $O(1)$ on all the others (since $|p|$ increases by only 1 per call). Thus Step 1 averages $O(1)$ per level. This gives an overall time of $O(T(n))$. No "constructibility" conditions are needed on Z or T since the ATM nondeterministically selects appropriate values. ∎

These results are sharp enough to suggest another way of defining uniform circuits. We might say C is a uniform circuit if and only if C is the computation graph of an ATM. This has the advantage of eliminating the need to discuss the complexity of the circuit constructor as distinct from the complexity of the circuit. Even if one is unwilling to accept this as a definition of uniform circuits, it is certainly a sufficient condition, which may often simplify proofs of uniformity. In particular, Theorems 3, 4, and 1 give the following useful result.

COROLLARY 1. *For all* $k \geqslant 2$, $\mathrm{NC}^{(k)} = \mathrm{ASPACE, TIME}(\log n, \log^k n)$.

Notice that this equality probably does not hold for $k < 2$ since NC is defined in terms of U_{BC} (i.e., log space) uniformity. By Theorems 3 and 4 the equality would extend to all $k \geqslant 1$ if instead NC were defined via U_{E} or U_{E^*} uniformity. Stated another way, the theorems show that these "circuit constructors" are no more powerful than the circuits they construct, which seems not to be true of the other uniformity conditions considered in Section 2. Thus, the U_{E} or U_{E^*} definitions seem more appropriate for defining and investigating $\mathrm{NC}^{(k)}$, $k < 2$.

4. APPLICATIONS

A few interesting corollaries of the theorems are given below. Known relationships among some of the complexity classes discussed are summarized in Fig. 2. The first corollary gives the space/depth result of [2]. (We could also give a time/size result, but it is not as sharp as [15].)

COROLLARY 2 (cf. [2]).

$$\begin{aligned}
\mathrm{NSPACE}(S(n)) &\subseteq \mathrm{ASPACE, TIME}(S(n), S^2(n)) && ([6]);\\
&\subseteq U_{\mathrm{E}}\text{-SIZE, DEPTH}(2^{O(S(n))}, S^2(n)) && (\textit{Theorem } 3);\\
&\subseteq U_{\mathrm{B}}\text{-SIZE, DEPTH}(2^{O(S(n))}, S^2(n)) && (\textit{Theorem } 1);\\
&\subseteq U_{\mathrm{B}}\text{-DEPTH}(S^2(n)) && (\textit{Immediate});\\
&\subseteq \mathrm{DSPACE}(S^2(n)) && ([2]).
\end{aligned}$$

The containment of CFLs in NC and several interesting characterizations of NC follow from results about the "tree-size" complexity of ATM computations [22],

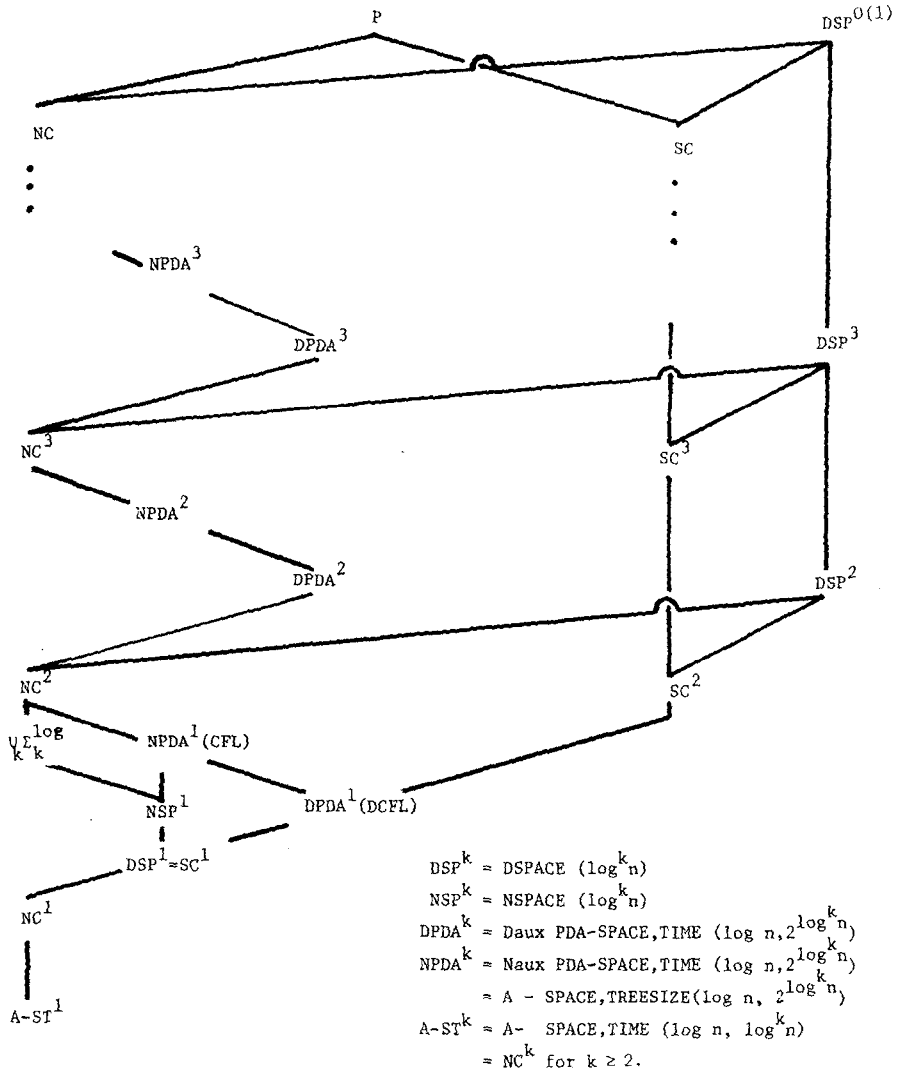
P
$DSP^{O(1)}$
NC
SC
$NPDA^3$
$DPDA^3$
DSP^3
NC^3
SC^3
$NPDA^2$
$DPDA^2$
DSP^2
NC^2
SC^2
$\bigcup_k \Sigma_k^{\log}$
$NPDA^1$(CFL)
NSP^1
$DPDA^1$(DCFL)
$DSP^1 = SC^1$
NC^1
$A\text{-}ST^1$
DSP^k = DSPACE $(\log^k n)$
NSP^k = NSPACE $(\log^k n)$
$DPDA^k$ = Daux PDA-SPACE,TIME $(\log n, 2^{\log^k n})$
$NPDA^k$ = Naux PDA-SPACE,TIME $(\log n, 2^{\log^k n})$
= A - SPACE,TREESIZE$(\log n, 2^{\log^k n})$
$A\text{-}ST^k$ = A- SPACE,TIME $(\log n, \log^k n)$
= NC^k for $k \geq 2$.

FIG. 2. Relations among some complexity classes.

"alternation" complexity of ATMs [6], and about auxiliary pushdown automata [4]. We recall the relevant definitions and results here.

DEFINITION. A language L is accepted by an alternating Turing machine M within simultaneous *tree-size* bound $Z(n)$ and space bound $S(n)$ if for every string w of length n in L there is at least one accepting computation tree for M on w of size (number of nodes) $Z(n)$, each node of which is labeled by a configuration using space $S(n)$. Further, for $w \notin$ L there is no accepting computation tree.

DEFINITION. An ATM is $A(n)$ *alternation bounded* if for each accepted input of length n there is an accepting computation tree in which each root-leaf path has at most $A(n)$ alternations between universal and existential states.

We observe that Theorem 4.5 of [6], attributed therein to Borodin, can be stengthened as follows:

PROPOSITION 1.

$$\begin{aligned}&\text{ASPACE, ALTERNATION}(S(n), A(n))\\ &\quad\subseteq \text{ASPACE, TIME}(S(n), S(n)\cdot(S(n)+A(n))).\end{aligned}$$

DEFINITION. An *auxiliary pushdown automaton* (*AuxPDA*) is a space-bounded Turing machine with an additional worktape which is constrained to operate as a pushdown store. Space on the pushdown is *not* included in the machine's space bound.

THEOREM 5 [22].

(1) $\text{CFL} \subseteq \text{ASPACE, TREESIZE}(\log n, n^{O(1)})$.

(2) $\text{ASPACE, TREESIZE}(S(n), Z^{O(1)}(n))$
$= \text{NauxPDA-SPACE, TIME}(S(n), Z^{O(1)}(n))$.

(3) $\text{ASPACE, TREESIZE}(S(n), Z^{O(1)}(n)) \subseteq \text{ASPACE, TIME}(S(n), S(n)\cdot\log Z(n))$.

These results, coupled with Theorems 3 and 4, give the following interesting characterizations of NC.

COROLLARY 3.

(1) $\text{NC} = \text{ASPACE, TIME}(\log n, \log^{O(1)} n)$.

(2) $\text{NC} = \text{ASPACE, ALTERNATION}(\log n, \log^{O(1)} n)$.

(3) $\text{NC} = \text{ASPACE, TREESIZE}(\log n, 2^{\log^{O(1)} n})$.

(4) $\text{NC} = \text{NauxPDA-SPACE, TIME}(\log n, 2^{\log^{O(1)} n})$.

(5) $\text{NC} = \text{DauxPDA-SPACE, TIME}(\log n, 2^{\log^{O(1)} n})$.

Proof. It suffices to show for all $k \geqslant 1$ that:

(a) $\text{NC}^{(k)} \subseteq \text{DauxPDA-SPACE, TIME}(\log n, 2^{\log^k n})$.

(b) $\subseteq \text{NauxPDA-SPACE, TIME}(\log n, 2^{\log^k n})$.

(c) $= \text{ASPACE, TREESIZE}(\log n, 2^{\log^k n})$.

(d) $\subseteq \text{ASPACE, ALTERNATION}(\log n, \log^k n)$.

(e) $\subseteq \text{ASPACE, TIME}(\log n, \log^{k+1} n)$.

(f) $= \text{NC}^{(k+1)}$.

Containment (a) holds since a circuit's value may be computed by a deterministic auxPDA in space $\log n$ and time exponential in the circuit's depth by doing a depth-first search from the output node. Further, for all $k \geqslant 1$, DauxPDA-SPACE, TIME($\log n$, $2^{\log^k n}$) is sufficient to generate the encoding of an $\mathrm{NC}^{(k)}$ circuit whether uniformity is defined as U_{BC}, U_{D}, U_{E}, or U_{E^*} (but probably not U_{B}). Containment (b) is immediate. Equality (c) is Theorem 5, part 2. Containment (d) is implicit in the direct simulation of tree-size by time in ([22], or Theorem 5, part 3 above). Containment (e) follows from Proposition 1 above. Equality (f) follows from Corollary 1. ∎

These results also show membership of some interesting language families in NC, namely, the log-space hierarchy $\Sigma_k^{\log}$ [6] which is analogous to the polynomial time hierarchy [27], and context-free languages.

COROLLARY 4.

(1) $\bigcup_k \Sigma_k^{\log} \subseteq \mathrm{NC}^{(2)}$.

(2) $\mathrm{CFL} \subseteq \mathrm{NC}^{(2)}$.

(3) $\mathrm{DCFL} \subseteq \mathrm{NC}^{(2)} \cap \mathrm{SC}^{(2)}$.

Proof. Statement (1) follows from Proposition 1 and Corollary 1; (2) follows from Theorem 5 parts 1 and 3, and Corollary 1; (3) follows from (2) and [5] in which the containment of DCFL in $\mathrm{SC}^{(2)}$ is shown. ∎

The size of the CFL circuit given by the above simulations is $>n^{15}$. This can be improved to about n^6 by a more direct simulation. This is still too large to be of much practical utility. Further size reductions, even at the expense of some increase in depth, would be of interest. Our result at least raises the possibility that more practical parallel algorithms may be feasible.

The membership of CFLs in NC implies fast parallel algorithms for some other problems. First, several other families of formal languages are known to be log-space reducible to context-free languages, and hence are in NC also [9, 28, 29]. Second, Goldschlager [11] has shown that a certain class of dynamic programming algorithms may be implemented on auxiliary PDAs in $\log n$ space and polynomial time, hence are also in NC. Problems solvable by such algorithms include constructing optimal binary search trees and finding a minimal cost ordering for a chain of matrix multiplies [11, 1]. Both results are under the assumption that the input numbers are small—$O(\log n)$ bits for inputs of length n. A third problem is the tree isomorphism problem (cf, e.g., [1, 3]) for trees of small degree ($O(\log n)$). A $\log n$ space auxiliary PDA can solve this problem in polynomial time by doing a depth-first search of one tree, while in parallel traversing the other, nondeterministically choosing an ordering of the descendants of each node. Small degree allows the PDA to insure that it has visited all descendants exactly once. Subtree isomorphism and some automorphism questions may be tested similarly.

The characterizations of NC given above should provide a powerful tool for studying parallel complexity. These results involve radically different models, giving us radically different viewpoints for design of efficient parallel algorithms. Each has its own virtues. For instance, the key step in showing that context-free languages were in NC involved simulating a tree-size bounded alternating Turing machine by a small space and small time bounded one. Obviously we could have simulated an auxiliary pushdown automaton, instead. However, we believe the simplicity of the tree structure was a significant conceptual aid in solving this problem. For some other purposes, the auxiliary pushdown automaton may be the preferable model. It is a sequential model which is natural for writing recursive or backtracking programs. Even pushdown automata which use super-polynomial time ($2^{\log^{O(1)} n}$) yield efficient parallel algorithms. Likewise, Pippenger's characterization of NC as D-TIME, REVERSAL ($n^{O(1)}$, $\log^{O(1)} n$) [17] has yet another set of strengths which may make it very useful for certain problems. For instance, it is the only one of the characterizations which does not place severe restrictions on space. We hope that these tools will provide important insights leading to practical new parallel algorithms.

5. Non-uniform Circuit Complexity

Another approach for studying the connection between circuit and Turing machine complexity has been discussed in the literature. This involves relating non-uniform circuit families to non-uniform Turing machines, i.e., TMs with "oracles" [2, 17, 25]. Our results also apply in this formulation. (The terminology below follows [17].)

Definition. A *non-uniform* ATM M is at ATM with a specially designated "query" state and a designated "query" worktape. There are no transitions out of the query state. For any language B, a configuration of M containing the query state with y on the query tape is *accepting modulo B* if and only if $y \in B$. The language *accepted* by M *modulo B* is then defined in the obvious way. M's space bound *in*cludes space on the query tape. We denote by, e.g., *ASPACE*($S(n)$) (*non-uniform*) the set of languages accepted in Space $O(S(n))$ by non-uniform ATMs modulo B for some B.

The non-uniform analog of Theorems 3 and 4 is the following.

Theorem 6.

$$\begin{aligned}&\text{SIZE-DEPTH}(Z^{O(1)}(n), T(n))\\&\quad = \text{A-SPACE, TIME}(\log Z(n), T(n))\ (\textit{non-uniform}).\end{aligned}$$

Proof. Simulation of non-uniform ATMs by circuits is as in the proof of Theorem 3, except that gates representing queries are "wired" to 0 or 1, depending on the contents of the query tape. Simulation of circuits by non-uniform ATMs is as in the proof of Theorem 4 except that questions about L_{EC} are answered directly by the oracle. ∎

6. Conclusion

We have attempted to motivate uniform circuit complexity as a realistic model of parallel computation. Uniform circuit complexity was shown to be fairly insensitive to our definitions. We have shown that simultaneous size and depth of circuits is very closely related to simultaneous space and time on alternating Turing machines. This rather surprisingly makes ATMs a realistic model of parallel computation, too. We think this result gives a useful new perspective on the connection between TM time/space and circuit size/depth. We have given several new characterizations of NC, which we believe corresponds to the class of problems for which substantial speedups are possible on implementible parallel machines. We have also extended the list of natural problems known to be in NC to include context-free language recognition and related problems. We hope our techniques will be useful in extending this list to other problems of practical importance.

Acknowledgments

Thanks to A. Borodin, S. Cook, M. Fischer, and N. Pippenger for stimulating discussions, and especially to M. Tompa for several important and terribly insightful observations.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, Mass., 1974.
2. A. Borodin, On relating time and space to size and depth, *SIAM J. Comput.* **6**, No. 4 (1977), 733–743.
3. C. J. Colbourn and K. S. Booth, "Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs," University of Waterloo, Computer Science Department, Tech. Report CS-79-06, March 1980.
4. S. A. Cook, Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18**, No. 1 (1971), 4–18.
5. S. A. Cook, Deterministic CFL's are accepted simultaneously in polynomial time and log squared space, *in* "Proceedings, 11th ACM Symposium on Theory of Computing, 1979," pp. 338–345.
6. A. K. Chandra, D. C. Kozen, and L. J. Stockemeyer, Alternation, *J. Assoc. Comput. Mach.* **28**, No. 1 (1981), 114–133.
7. L. Csanky, Fast parallel matrix inversion algorithms, *in* "Conference Record, IEEE 16th Annual Symposium on Foundations of Computer Science, 1975," pp. 11–12.
8. P. W. Dymond and S. A. Cook, Hardware Complexity and Parallel Computation, *in* "21st IEEE Symposium on Foundations of Computer Science, 1980," pp. 360–372.
9. W. Erni, "Some Further Languages Log-Tape Reducible to Context-Free Languages," Institüt für Angewandte Informatik ünd Formale, Beschreibüngverfahren, Universität Karlsrühe, Karlsrühe, West Germany.
10. S. Fortune and J. Wyllie, Parallelism in random access machines, *in* "Proceedings, 10th ACM Symposium on Theory of Computation, 1978," 114–118.
11. L. M. Goldschlager, "Synchronous Parallel Computation," Ph. D. dissertation, University of Toronto, Department of Computer Science TR-114, December 1977.

12. L.M. Goldschlager, A unified approach to models of synchronous parallel machines, *in* "Proceedings, 10th ACM Symposium on Theory of Computation, 1978," pp. 89–94.
13. J. Hartmanis and J. Simon, On the power of multiplication in random access machines, *in* "Proceedings, 15th IEEE Symposium on Switching and Automata Theory, 1974," pp. 13–23.
14. R. E. Ladner, The circuit value problem is log space complete for P, *SIGACT News* 7, No. 1 (1975), 18–20.
15. N. Pippenger and M. J. Fischer, Relations among complexity measures, *J. Assoc. Comput. Mach.* **26**, No. 2 (1979), 361–381.
16. N. Pippenger, "Fast Simulation of Combinational Logic Networks by Machines without Random-Access Storage," IBM Research Report RC 6582, 1977.
17. N. Pippenger, On simultaneous resource bounds, *in* "Proceedings, 20th IEEE Symposium on Foundations of Computer Science, 1979."
18. N. Pippenger, "Pebbling with an Auxiliary Pushdown," IBM Research Report RJ3012, 1980.
19. V. R. Pratt, M. O. Rabin, and L. J. Stockmeyer, A characterization of the power of vector machines, *in* "Proceedings, 6th Annual ACM Symposium on Theory of Computing, 1974," pp. 122–134.
20. V. R. Pratt and L. J. Stockmeyer, A characterization of the power of vector machines, *J. Comput. System Sci.* **12** (1976), 198–221.
21. W. J. Paul and R. E. Tarjan, Time-space trade-offs in a pebble game, *Acta Inform.* **10** (1978), 111–115.
22. W. L. Ruzzo, Tree-sized bounded alternation, *J. Comput. System Sci.* **21** (1980), 218–235.
23. W. L. Ruzzo, "An improved characterization of the power of vector machines," University of Washington, Computer Science Department Technical Report 78-10-01, in preparation.
24. W. J. Savitch, Parallel and nondeterministic time complexity classes, *in* "Automata, Languages and Programming, 5th Colloquium, Udine, 1978," pp. 411–424, Lecture Notes in Computer Science No. 62, Springer–Verlag, Heidelberg, 1978.
25. C. P. Schnorr, The network complexity and the Turing machine complexity of finite functions, *Acta Inform.* 7 (1976), 95–107.
26. W. J. Savitch and M. J. Stimson, Time bounded random access machines with parallel processing, *J. Assoc. Comput. Mach.* **26**, No. 1 (1979), 103–118.
27. L. J. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1976), 1–22.
28. I. H. Sudborough, Time and tape bounded auxiliary pushdown automata, *in* "Mathematical Foundations of Computer Science," pp. 493–503, Lecture Notes in Computer Science No. 53, Springer-Verlag, Heidelberg, 1977.
29. I. H. Sudborough, The complexity of the membership problem for some extensions of context-free languages, *Internat. J. Comput. Math. A* **6** (1977), 191–215.