TOWARDS A COMPLEXITY THEORY OF SYNCHRONOUS PARALLEL COMPUTATION

Autor(en): **Cook, Stephen A.**

Objekttyp: **Article**

Zeitschrift: **L'Enseignement Mathématique**

Band (Jahr): **27 (1981)**

Heft 1-2: **L'ENSEIGNEMENT MATHÉMATIQUE**

PDF erstellt am: **09.08.2024**

Persistenter Link: https://doi.org/10.5169/seals-51742

TOWARDS A COMPLEXITY THEORY OF SYNCHRONOUS PARALLEL COMPUTATION

by Stephen A. Cook [1])

Abstract. This is largely an expository paper on the general theory of synchronous parallel computation. The models of parallel computers discussed include uniform circuit families, alternating Turing machines, conglomerates, vector machines, and parallel random access machines. A classification of these models indicates the need for still more; so "aggregates" and "hardware modification machines" are introduced. The resources sequential time, space, parallel time, circuit size and depth, hardware size etc., are discussed and interrelated. Work in progress at Toronto is mentioned and basic open questions are listed.

1. Introduction

There is now a well developed computational complexity theory of sequential computation. The precisely "right" computer model is not completely clear, but the main contenders for this model do not differ markedly from each other in their computing efficiency. These contenders are multitape Turing machines, possibly with storage structures more general than linear tapes, and various versions of random access machines. Of these models, the storage modification machine (SMM) made popular by Schönhage [S2] carries the most conviction as a stable and general model of a sequential computer; where we take sequential to mean the number of active elements is bounded in time.

To be sure, there is a feeling that one step of an SMM may be a little too powerful. It is hard to imagine a mechanism for reconnecting a given edge out of a node v_1 in the storage structure to a node v_2 in one step, when the candidates for v_2 from the perspective of the whole computation are unlimited. But the fact remains that if we restrict ourselves to fixed storage

[1]) Presented at the *Symposium über Logik und Algorithmik* in honour of Ernst Specker, Zürich, February 1980.

structures, there is no single structure or class of structures which seems to be just right. (Certainly multitape Turing machines are too restrictive.) On the other hand, for random access machine models one is never quite sure which set of operations should be primitive, and whether to charge more than one time unit for an operation capable of manipulating arbitrarily large integers.

Whatever the sequential model, it is clear that the main resources of interest are time and space. Let me repeat that the differences among the leading models in the time and space needed to execute algorithms are minor. And the theory of sequential time and space complexity is a rich and interesting one.

In the past few years it has become increasingly clear that the most powerful computers of the future will not be sequential but parallel. An entire processor can now be placed on a VLSI chip that is so small and cheap that it is not hard to imagine a machine of the future consisting of millions of such processors connected together and operating synchronously. The questions then become: How should the machine be organized and what can be done with the result? Hence the need for a theory of parallel computation. (A second motivation, of course, is that the human brain appears to be a parallel computer.)

I should point out here that the theory I have in mind deals only with synchronous computers. There is indeed a great and interesting literature on asynchronous processes, and the theory has applications when the processes in question cannot easily be synchronized (such as distributed computer systems or operating systems). The theory discussed here assumes one parallel computer whose elements have been designed from scratch to operate synchronously.

The first problem in this theory is to find the right mathematical model of a parallel computer. The parallel models in the literature fall roughly into two classes: those with fixed structure and those with modifiable structure. The fixed structure parallel models correspond to sequential machines with fixed storage structure, namely Turing machines with "tapes" which may be more general than linear arrays, but cannot be modified. The parallel analogs of these include Borodin's uniform circuit families [B1], Goldschlager's conglomerates [G1], [G2], and Hoover's uniform infinite circuits [H1].

The modifiable sequential machines include SMM's and random access machines (RAM's). (Indirect addressing in a RAM gives the effect of a modifiable storage structure, and in fact RAM's which can only add and

subtract one are equivalent to SMM's [S2].) The modifiable parallel machines include various parallel RAM's, such as SIMDAG's [G1] and P-RAM's [SS] and [FW], as well as vector machines as defined in [PS]. As yet no parallel analog of SMM's has appeared, but a tentative candidate is introduced in section 5.

Fortunately, all these models are roughly equivalent from the point of view of computation time, in the sense that each can simulate another while at most cubing the computation time. In fact, the sets or functions computed by each in time $S^{O(1)}$ (i.e. time polynomial in S: this notation appears in [P1]) are the same as those computed by a Turing machine in space $S^{O(1)}$ for any well behaved time bound S. (This phenomenon was observed, for example, in [CS], and called the "parallel computation thesis" in [G1].)

This thesis can be made more specific as follows: For the fixed structure parallel machines; namely, uniform circuit families, conglomerates, "aggregates" (see section 4) and uniform infinite circuits (see [H1]),

(1.1) parallel time (S) $\subseteq$ DSPACE (S) $\subseteq$ NSPACE (S) $\subseteq$ parallel time (S^2)

(See [HU] for the meaning of DSPACE and NSPACE.)
On the other hand, the modifiable parallel machines tend to be more powerful, and the inclusions become (at least for SIMDAG's and the P-RAM's of [FW]):

(1.2) parallel time (S) $\subseteq$ DSPACE (S^2) $\subseteq$ parallel time (S^2).

(For SIMDAG's, the stronger statement NSPACE (S) $\subseteq$ parallel time (S) also holds [G1]).

The modifiable parallel models that have been proposed so far all share the same problem as the sequential RAM models: The choice of primitive operations seems arbitrary, and most of these operations (such as shifts in vector machines and random access to global storage in P-RAM's) seem too powerful to be primitive. Hence I propose a new modifiable parallel model: "Hardware Modification Machines" (HMM's), to be the parallel analog of SMM's. These are discussed in section 5.

Time and space are the fundamental resources in sequential complexity theory. What are their analogs in the parallel theory? Obviously, parallel time plays a fundamental role. The second important parallel resource, I think, should be hardware size; that is, the number of elements of a machine which are active during a computation. For conglomerates, hard-

ware size is the number of active finite state machines, and for vector machines it is the sum of the lengths of the vectors. For SIMDAG's and P-RAM's it corresponds roughly to the number of processors, although it should take into account the total memory used. For circuits, the circuit size is an upper bound on hardware size, but the traditional restriction that circuits are acyclic disallows elements to be reused during a computation and hence may give an unrealistically large value for size. Hence "aggregates" are introduced in section 4. These can be thought of either as circuits with cycles, or as finite conglomerates.

Section 2 discusses two fundamental fixed structure parallel models; namely, uniform circuit families and alternating Turing machines. These turn out to be nearly equivalent. Section 3 gives examples which are log depth complete for deterministic log space, and hence may distinguish between two similar classes: deterministic log space and uniform log circuit depth. Section 4 discusses two fixed structure models useful for considering hardware size as well as parallel time; namely, conglomerates and aggregates. Section 5 introduces hardware modification machines, and section 6 surveys other modifiable parallel models, such as vector machines and parallel RAM's. Section 7 discusses characterizations and interrelationships between two complexity classes defined by simultaneous resource bounds; namely, NC and SC Finally, section 8 lists some open problems.

2. Circuits and Alternating Turing Machines

Perhaps the simplest model for measuring the parallel time to compute a function is the combinational circuit (or simply a circuit). (See [S3] and [P2] for general discussions of circuits.)

Notation. $B_n = \{f \mid \{0, 1\}^n \to \{0, 1\}\}$ = the set of all Boolean functions of rank n.

Definition. A *circuit* α *with* n *inputs* is a finite directed acyclic graph such that each node has a label from $\{x_1, ..., x_n\} \cup B_0 \cup B_1 \cup B_2$. A node labelled x_i must have indegree zero, and is called an *input* node. A node v with label $g \in B_i$ must have indegree i, and one edge into v is associated with each argument of g. Certain nodes are designated *output* nodes. When the variables x_i are assigned values from $\{0, 1\}$ every node v assumes a unique value in $\{0, 1\}$, so that v computes some function f_v of $x_1, ..., x_n$. We say the circuit α *computes* f if $f = f_v$ for some output node v.

We shall assume that every node v has a path from v to some output. That is, we assume there are no syntactically superfluous nodes.

Let $c(\alpha)$ (the complexity of α) be the number of *gates* (i.e. nodes other than inputs) in α, and let $d(\alpha)$ (depth of α) be the length of the longest path in α. If $f \in B_n$, then $c(f) = \min\{c(\alpha) \mid \alpha \text{ computes } f\}$ and $d(f) = \min\{d(\alpha) \mid \alpha \text{ computes } f\}$.

If $A \subseteq \{0, 1\}^*$, then $A^n = A \cap \{0, 1\}^n$. We can regard A^n as a member of B_n by the convention $A^n(x_1, ..., x_n) = 1$ iff $(x_1 \ldots x_n) \in A^n$. A family $\{\alpha_n\}$ of circuits *computes* A iff α_n computes A^n for all n, and each α_n has a unique output node.

Notation. Let $S, T : N^+ \to R$. Then

$$\text{SIZE}(T) = \{A \mid \exists \{\alpha_n\} : \{\alpha_n\} \text{ computes A and } c(\alpha_n) = O(T(n))\}$$
$$\text{DEPTH}(S) = \{A \mid \exists \{\alpha_n\} : \{\alpha_n\} \text{ computes A and } d(\alpha_n) = O(S(n))\}$$

We shall always assume $T(n) \geqslant n$ and $S(n) \geqslant \log n$.

These complexity classes are strange in that they include nonrecursive sets A. In fact, by Lupanov's result (see [S3]) $\text{SIZE}(2^n/n) = 2^{\{0,1\}^*}$, and by disjunctive normal form $\text{DEPTH}(n) = 2^{\{0,1\}^*}$. Nevertheless they are mathematically interesting, and have intuitive significance especially for lower bound results. In particular, a proof that $A \notin \text{DEPTH}(S)$ means that no parallel computer with fixed circuitry could compute A in time O(S). This is because the parallel computation could be unwound to form a circuit with constant delay at each gate. Our assumption that circuits have bounded fan-in (in fact fan-in two) is justified by engineering experience that any general design for a gate with n inputs has a delay at least proportional to $\log n$. On the other hand, Hoover [H1] gives results that show that an assumption of fan-*out* two would not materially alter either the depth or the size complexity of a set A.

Although the circuit depth to compute A is a reasonable lower bound on the parallel time required, it is not a reasonable upper bound in general (unless we want parallel machines to compute nonrecursive sets). Borodin [B1] proposed making it reasonable by requiring that the family $\{\alpha_n\}$ computing A be uniform in some sense. The trouble is there is no clearly correct choice for the definition of *uniform*. (See Ruzzo [R1] for a discussion of various possibilities.) Here we shall adopt the following definition, which has gained some acceptance (see [C1], [R1], and [P1]):

Definition. A family $\{\alpha_n\}$ of circuits is *uniform* provided some deterministic Turing machine can compute the transformation $1^n \to \bar{\alpha}_n$ in space

$O(\log c(\alpha_n))$. (Here $\bar{\alpha}_n$ is a binary string coding the circuit α_n in some reasonable fashion.)

We can now define the uniform complexity classes

$$\begin{aligned}\text{USIZE (T)} &= \{A \mid \exists \text{ uniform } \{\alpha_n\} : \{\alpha_n\} \text{ computes A and } c(\alpha_n) \\ &= O(T(n))\} \\ \text{UDEPTH (S)} &= \{A \mid \exists \text{ uniform } \{\alpha_n\} : \{\alpha_n\} \text{ computes A and } d(\alpha_n) \\ &= O(S(n))\}\end{aligned}$$

Notice that the size $c(\alpha_n)$ is not mentioned in the definition of UDEPTH so it can be taken to be as large as possible consistent with $d(\alpha_n)$. In fact, every circuit of depth d with a unique output can be expanded into an equivalent tree circuit of size $2^d - 1$. Also, our assumption of no superfluous nodes implies that no unique-output circuit of depth d can have more than $2^d - 1$ nodes. This leads to the following

PROPOSITION 2.1. *The class* UDEPTH (S) *remains unchanged if the definition of uniform family* $\{\alpha_n\}$ *is changed to require that the transformation* $1^n \to \bar{\alpha}_n$ *be computable in deterministic space* $O(d(\alpha_n))$ *instead of* $O(\log(c(\alpha_n)))$.

The alternative definition of uniform is in fact the one given by Borodin [B1].

Borodin expresses the general thesis in [B1] that circuit size corresponds to Turing machine time and circuit depth corresponds to Turing machine space. (If we identify uniform circuit depth with parallel time, then the second assertion is an instance of the parallel computation thesis stated in section 1.) One precise statement of Borodin's thesis is the following:

THEOREM 2.1. *If* [log T] *is fully space constructable, then*

$$\text{USIZE}(T^{O(1)}) = \text{DTIME}(T^{O(1)}).$$

If S *is fully space constructable, then*

$$\text{UDEPTH}(S^{O(1)}) = \text{DSPACE}(S^{O(1)}).$$

(See [HU] for the definitions of *constructable*, *DTIME* and *DSPACE*. We have altered the definitions of the latter so they contain only subsets of $\{0, 1\}^*$.)

The first equation is easy in this crude form, and in fact can be made considerably more precise (see [P1]).

The second equation is a consequence of the following result of Borodin [B1], which states that the inclusions (1.1) hold for uniform circuit depth.

THEOREM 2.2. *If* S *is fully space constructable, then*

$$\text{UDEPTH (S)} \subseteq \text{DSPACE (S)}, \quad \textit{and}$$
$$\text{NSPACE} \quad \subseteq \text{UDEPTH } (\text{S}^2).$$

COROLLARY (Savitch's Theorem). *If* S *is fully space constructable, then* NSPACE (S) $\subseteq$ DSPACE (S^2).

Let us sketch the proof of theorem 2.2. The *circuit value problem* (see [HU]) is the set of all binary strings encoding systems $\langle x_1, \ldots, x_n; \alpha \rangle$ where each $x_i \in \{0, 1\}$ and α is a circuit whose unique output is 1 when its n inputs take on the values $x_1, \ldots, x_n$.

LEMMA 2.1. *There is a deterministic Turing machine* M *which recognizes the circuit value problem, and on an input encoding* $\langle x_1, \ldots, x_n; \alpha \rangle$, M *uses space* $\text{O}(d(\alpha))$.

The idea is to perform a depth first search of α from the output node taking left descendants first. M stores the number of the node v currently examined, together with one symbol for each node on the path followed from the root to v. This symbol is either a marker L, if the search is proceeding on through the left input of the node; or the value of the left input if this value has been determined and the search is proceeding on to the right.

The first inclusion of Theorem 2.2 follows from Lemma 2.1 and Proposition 2.1.

To prove the second inclusion, recall that the graph reachability problem (GRP) (see [HU]) is the set of all binary strings encoding the adjacency matrix of a digraph G on nodes $\{1, 2, \ldots, \text{N}\}$ such that G has a path from node 1 to node N.

LEMMA 2.2. GRP $\in$ UDEPTH ($\log^2 n$).

The proof involves constructing a circuit which computes the transitive closure of a Boolean matrix by repeated squaring. The circuit has O ($\log n$) stages, and each stage has depth O ($\log n$) and computes the Boolean square of the matrix resulting from the previous stage. The circuit can be constructed by a deterministic Turing machine in space O ($\log n$).

Given a nondeterministic S space bounded Turing machine M and the input length n, a circuit α_n is constructed which does the following on an

input string w of length n. α_n first computes the adjacency matrix A of the graph whose nodes are the possible configurations of M with an input of length n, and whose edges represent possible steps in a computation with input w. We can assume M has an initial configuration labelled 1 and a unique accepting configuration labelled N. α_n now solves the graph reachability problem for A according to Lemma 2.2. The solution to the problem is positive iff M accepts w. Using Lemma 2.2 it is not hard to see that α_n has depth $O(S^2)$ and can be constructed in deterministic space $O(S^2)$ (in fact, space $O(S)$).

Theorem 2.1 represents one way to make precise Borodin's thesis that size corresponds to time and depth to space. Alternatively, instead of making the circuit family $\{\alpha_n\}$ uniform one can make Turing machines nonuniform (see [S1]). We borrow from Pippenger's terminology [P1]. Suppose $g : \{0, 1\}^* \to \{0, 1\}^*$. We say that a (deterministic or nondeterministic) multitape Turing machine *accepts* A *modulo* g provided that M accepts A under the condition that in addition to the normal input $x \in \{0, 1\}^*$ on a read only input tape M is also provided with $g(x)$ on a separate read only tape called the *reference* tape. The space used by M is the work tape space plus $\lceil \log |g(x)| \rceil$, where $|w|$ is the length of w. (The term $\lceil \log |g(x)| \rceil$ was not counted in [P1], but it should be, since it represents the amount of information stored by the position of the head on the reference tape.) The function g is *length determined* if $g(x)$ depends only on $|x|$, and not otherwise on x. A *nonuniform* machine is a machine M together with a length determined function g. It accepts A provided it accepts A modulo g. We add (NONUNIFORM) after a complexity class to indicate the machines are allowed to be nonuniform.

There is an alternative and more elegant definition of nonuniform space. We say that A is in DSPACE (S) (NONUNIFORM) provided there is a family $\{F_n\}$ of finite automata, each with a two-way read only input tape, such that F_n recognizes A^n, and $\log |F_n| = O(S(n))$, where $|F_n|$ is the number of states of F_n. It is not hard to verify that this definition is equivalent to the one in the previous paragraph (recall our convention that $S(n) \geqslant \log n$).

The above definition does not work for time. However, Les Valiant pointed out that we could change the definition of nonuniform Turing machine to be a family $\{M_n\}$ of Turing machines instead of a single Turing machine with a reference tape. The time complexity $T(n)$ of such a family would be the maximum of $|M_n|$ and the worst case running time of M_n on inputs of length n. The space complexity $S(n)$ would be $\log |M_n|$ plus

the worst case space used by M_n on inputs of length n. This gives the same definition of nonuniform space as before, but the nonuniform time is only the same up to application of a polynomial.

In any case, theorems 2.1 and 2.2 have the following analogs for nonuniform machines:

THEOREM 2.3.

$$\text{SIZE}(T^{O(1)}) = \text{DTIME}(T^{O(1)}) \text{ (NONUNIFORM), } \textit{and}$$
$$\text{DEPTH}(S^{O(1)}) = \text{DSPACE}(S^{O(1)}) \text{ (NONUNIFORM).}$$

THEOREM 2.4.

$$\text{DEPTH}(S) \subseteq \text{DSPACE}(S) \text{ (NONUNIFORM), } \textit{and}$$
$$\text{NSPACE}(S) \text{ (NONUNIFORM)} \subseteq \text{DEPTH}(S^2).$$

To prove these results, the nonuniform machines simulate the circuits by letting $g(x)$ provide a description of the circuit for inputs of length $|x|$. Conversely, a circuit family $\{\alpha_n\}$ can simulate a nonuniform machine by building into α_n the value of $g(x)$ for $|x| = n$.

Note that the following nonuniform version of Savitch's theorem is a consequence of Theorem 2.4:

COROLLARY.

$$\text{NSPACE}(S) \text{ (NONUNIFORM)} \subseteq \text{DSPACE}(S^2) \text{ (NONUNIFORM).}$$

In other words, a 2^s-state 2NFA can be simulated by a $2^{O(s^2)}$-state 2DFA for inputs of length $n \leqslant 2^s$.

A second interesting model of parallel computation, which falls in the fixed structure category, is the *alternating Turing machine* (ATM) ([CS], [K1], [CKS]). An ATM is a generalization of a nondeterministic multitape Turing machine. A nondeterministic machine has *existential* states, for which there are several possible next states, and at least one of the alternatives must lead eventually to an accepting state. In addition to existential states, an ATM also has *universal* states, for which *all* of the possible next states must lead to an accepting state. We define the accepting state to be a universal state with no successors. Every state is either universal or existential. Thus an *accepting computation* of an ATM M *with input* w is a finite tree whose nodes are labelled with configurations of M, such that i) every universal node (i.e. node whose configuration has a universal state) must have all possible next configurations as children, ii) every existential node

must have at least one possible next configuration as a child, and iii) the root is the initial configuration. In order for M to operate in sublinear time we assume it has "random access" to the bits of w instead of a read only input tape. That is, M has a special index tape, and when M writes an index i on the index tape and assumes one of a distinguished set of index states, the i-th symbol of the input w is placed on the index tape. We say M *accepts* w in time s and space l if there is an accepting computation of M with input w whose longest path from root to leaf is s or less, and such that no configuration in the computation has tapes of length exceeding l. The complexity classes for time and space for ATM's are designated ATIME (S) and ASPACE (L), respectively, and we always assume $S(n), L(n) \geqslant \log n$.

As different as ATM's may seem from uniform circuit families, there is a remarkably close correspondence between alternating time and circuit depth, and between alternating space and circuit size. Unfortunately, our definition of *uniform* for circuits is too weak to express the correspondence precisely. Ruzzo gives a number of alternative definitions, of which the strongest is the following: $\{\alpha_n\}$ is U_E *uniform* iff the connection language L_{EC} can be recognized by a deterministic Turing machine in time $O(\log c(\alpha_n))$. Here L_{EC} consists of those quadruples (n, g, p, x) such that if g' is the gate reached by following the path $p \in \{L, R\}^*$ (where $|p| \leqslant \log c(\alpha_n)$) in circuit α_n back from gate g (L, R refer to left and right input, respectively) then g' has label x if $x \in B_2$, and $g' = x$ otherwise. (Assume n, g and x are expressed in binary notation.)

If we use the notation, for example, U_EDEPTH (S) to indicate this notion of uniformity, then we have

$$\text{ATIME (S)} = U_E\text{DEPTH (S)}, \quad \text{and}$$
$$\text{ASPACE (L)} = U_E\text{SIZE } (2^{O(L)}),$$

assuming S (n) can be computed in deterministic time S (n), and L (n) can be computed in deterministic time L (n) given n in binary notation. In fact, Ruzzo [R1] proves the stronger result that the equivalences hold simultaneously. Let us use the notation ATIME-SPACE (S, L) for the class of sets accepted simultaneously in time S and space L on an ATM, (note that this may be a proper subset of the intersection of ATIME (S) and ASPACE (L)), and analogous notation for other simultaneous classes. Then

THEOREM 2.5. ATIME-SPACE (S, L) = U_EDEPTH-SIZE (S, $2^{O(L)}$), *provided* S *and* L *are computable in deterministic time* O (S).

Ruzzo shows the above result still holds when U_E is replaced by U, provided $S \geqslant L^2$.

From their definition, ATM's appear to model a restricted form of parallel computation, because the "processors" in the model are restricted to be Turing machines, and they must be organized in the form of an and-or-tree. This makes Theorem 2.5 all the more interesting. On the other hand, ATM's are more pleasing in one way than circuit families, because there is no question of how to define *uniform*. Each ATM is automatically uniform. In fact, ATM's may be the best candidate proposed so far for defining parallel time, at least in the fixed structure category. But this remains to be seen. The one clear drawback of ATM's is that they do not seem to have any resource that corresponds to hardware size (see section 4).

3. Log Depth vs Log Space

As far as we know, the second inclusions in Theorems 2.2 and 2.4 cannot be improved, even when NSPACE is replaced by DSPACE. (Of course an improvement for NSPACE would improve Savitch's theorem.) Taking $S(n) = \log n$ as the most basic case, it is interesting to look for examples of sets in DSPACE $(\log n)$ which do not appear to be in DEPTH $(\log n)$. Addition of n n-digit binary numbers, and multiplication of two n-digit binary numbers both can be done in $O(\log n)$ circuit depth (see [S3]), as can sorting n n-digit binary numbers (see [MP]). On the other hand, the "cycle free problem" is in DSPACE $(\log n)$ but does not appear to be in DEPTH $(\log n)$.

Definition. The cycle free problem (CFP) is the set of all binary codes for symmetric Boolean $N \times N$ adjacency matrices A of undirected cycle-free graphs.

One can define functions $f: \{0,1\}^* \to \{0,1\}^*$ computable in depth S (or uniform depth S) using circuits with several outputs. We say A_1 is *log depth reducible* to A_2 (respectively *uniformly log depth reducible*) iff there is some function f computable in depth $O(\log n)$ (respectively uniform depth $O(\log n)$) such that $w \in A_1$ iff $f(w) \in A_2$, for all w. We say A is *log depth complete* for the class $\mathcal{S}$ iff $A \in \mathcal{S}$, and every $A' \in \mathcal{S}$ is log depth reducible to A. The uniform case is defined similarly. The main ideas in the proof of the following result appear in Hong [H2].

THEOREM 3.1. (a) CFP *is uniformly log depth complete for* DSPACE ($\log n$).

(b) CFP *is log depth complete for* DSPACE ($\log n$) (NONUNIFORM).

COROLLARY (a) DSPACE ($\log n$) = UDEPTH ($\log n$) iff CFP $\in$ UDEPTH ($\log n$).

(b) DSPACE ($\log n$) (NONUNIFORM) = DEPTH ($\log n$) *iff* CFP $\in$ DEPTH ($\log n$).

We note that because UDEPTH ($\log n$) $\subseteq$ DEPTH ($\log n$), the first equation in the Corollary implies the second. This fact does not seem to be obvious without using the CFP.

To prove CFP $\in$ DSPACE ($\log n$), Hong devised an algorithm for moving several pebbles around the input graph in an attempt to do a depth first search of each of its components. To prove that every

$$A \in \text{DSPACE } (\log n)$$

is uniformly log depth reducible to CFP, one can, given an input w, define a graph whose nodes are $\mathscr{C} \times \{0, 1, ..., T\}$, where $\mathscr{C}$ is the set of possible configurations of the Turing machine M with input w, where M accepts the complement of A in space O ($\log n$), and T is an upper bound on the computation time. Two nodes (c, t) and $c', t')$ are adjacent iff either $c \to c'$ in one step and $t' = t + 1$, or $c' \to c$ and $t = t' + 1$. If we let c_0 be the initial configuration and c_f be the unique accepting configuration, then we also add an edge between $(c_0, 0)$ and (c_f, T). Using the fact that M is deterministic, it is not hard to see that M accepts w iff the graph has a cycle.

A second example for which theorem 3.1 applies is GAP1: the graph reachability problem for directed graphs of outdegree one. The completeness of GAP1 for DSPACE ($\log n$) is proved for reducibilities other than log depth in [J] and in [HIM]. The proof of theorem 3.1 for GAP1 is easier than for CFP.

The following example is interesting, because it is complete for non-uniform $\log n$ space, but no one knows how to solve it in uniform $\log n$ space.

Definition. The undirected graph reachability problem (URP) is the set of codes of symmetric adjacency matrices of graphs with nodes $\{1, 2, ..., N\}$ with a path from node 1 to node N.

THEOREM 3.2. URP *is log depth complete for* DSPACE ($\log n$) (NONUNIFORM).

That URP $\in$ DSPACE ($\log n$) (NONUNIFORM) follows from the existence of a short universal covering string for all n-node undirected connected oriented graphs of fixed degree (see [AKLLR]). The reducibility proof is similar to the above argument.

Many interesting problems have O ($\log^2 n$) as the best known upper bound for both deterministic space and uniform depth. It is interesting to try to reduce these to each other via log depth or uniform log depth reducibility, so as to cut down the number of equivalence classes of problems classified by their depth complexity. For example, the directed graph reachability problem (GRP) is well known to be log space complete for NSPACE ($\log n$) (see [HU]). In fact, it is also *uniform log depth* complete for NSPACE ($\log n$). Two other examples are finding the integer part of the quotient of two n-digit binary numbers, and raising an n-digit number to the power n. The best known upper bound for both problems for both space and depth is O ($\log^2 n$). Hoover [H1] shows that each is log depth reducible to the other, although one of the reductions is not uniform. As a matter of interest, Hoover also points out that the base conversion problem (say converting binary notation to ternary) is in nonuniform depth O ($\log n$) (because the powers of two in ternary can be built in), but the best space upper bound and uniform depth upper bound is O ($\log^2 n$).

4. CONGLOMERATES AND AGGREGATES

Uniform circuits and ATM's are good models for measuring parallel time, but neither is right for measuring the second important resource mentioned in the introduction, namely hardware size. What is needed is to allow circuits to have cycles. Goldschlager's conglomerates [G1] satisfy this requirement. Briefly, a *conglomerate* is an infinite collection $\{M_0, M_1, ...,\}$ of identical deterministic finite state machines connected together in some manner. Each machine has $r \geqslant 1$ inputs and one output, and the connection function f specifies for some inputs of some machines the output of which machine it is connected to. (Inputs left unconnected receive some fixed symbol b.) Cycles are allowed in the connection graph. Initially at time O, the first n machines $M_1, ..., M_n$ store the symbols of the input string $w_1 w_2 ... w_n$, and all other machines start in the initial state q_0. At sub-

sequent times 1, 2, ... each machine assumes a new state and transmits output symbols in a manner determined by its input symbols and state at the previous step. The conglomerate accepts its input if at any time machine M_0 enters the special state q'.

The uniformity condition for conglomerates specifies that the connection function f can be computed within some space bound P on a Turing machine, where $f(i_1 i_2 ... i_k) = s$, if machine M_s is reached by starting with M_0 and tracing back via input i_1, then input i_2 of that machine, and so on. The linear space bound $P(n) = n$ suffices in order for the inclusions (1.1) to hold when parallel time is taken to mean conglomerate time. We have not considered the question of which uniformity condition makes conglomerate time equivalent to uniform circuit depth.

Goldschlager did not define or study the "hardware size" of a conglomerate computation. Rather than do that now, we present a new model (developed in Dymond [D1]) to study, called an *aggregate* which can be viewed either as like a finite conglomerate, or like a circuit with feedback. Similar objects have been called "sequential circuits" or "logical nets" in the switching theory literature. An aggregate has different input/output conventions than these, and we assume every gate has unit delay to avoid any possibility of ambiguous computations. We interpret the result as more a "parallel circuit" than a "sequential circuit".

More formally, an *aggregate* β_n on inputs $x_1, ..., x_n$ is a directed graph (not necessarily acyclic) whose nodes have labels from $B_0 \cup B_1 \cup B_2 \cup \{x\}$. A node v with label $g \in B_i$ must have indegree i, and one edge into v is associated with each argument of g. If a node v has label x, then v is an *input* node and must have indegree zero. Associated with each input node v is a *register* R_v consisting of $[\log n]$ nodes, which specifies which input x_i is presented to x. There is a distinguished pair of nodes designated v_0 and v_1, called *output* nodes. A *configuration* of β_n is an assignment of 0 or 1 to each node v of β_n called the *output* of v. A *computation* of β_n is a sequence $C_0, C_1, ...$ of configurations as follows.

(a) All nodes in C_0 have output 0 except any node labelled with the constant function $1 \in B_0$.

(b) If v has label $g \in B_i$, then in C_{t+1} v has output equal to g applied to the input (s) of v in C_t.

(c) If v is an input node, then v has output 0 in C_t for $t < \lceil \log n \rceil$, and in general in $C_{t+\lceil \log n \rceil}$ v has output x_{i+1}, where i is the value in binary notation of the register R_v in C_t.

The output of β_n is defined to be the output of the node v_0 in the first configuration C_t in which v_1 has output 1. The running time $t(\beta_n)$ of β_n is the maximum over all inputs $x_1, \ldots, x_n$ of this index t. The *hardware size* $h(\beta_n)$ is the number of nodes in β_n.

The peculiar input conventions for aggregates need justification. The reason that inputs x_i are not fed directly into aggregates as they are for circuits is that this would entail $h(\beta_n) \geqslant n$, whereas we are interested in sublinear hardware bounds (see theorem 4.1 below). In fact, the value of an input node v could be computed from the index stored in R_v using a decoding circuit of size $O(n)$ and depth $O(\log n)$ (this is the reason we assume a delay of $\lceil \log n \rceil$ for R_v to affect v). Our convention of not counting the size of the decoding circuit is similar to the convention of not counting the input tape in measuring the space used by an off line Turing machine. (One might imagine, for example, a large number of small aggregates sharing the same large decoding circuits.)

Our input and output conventions could be modified slightly to allow aggregates to compute functions instead of to recognize sets. The particular bit computed of the function would be specified by a part of the input called the output specifier. Then aggregates could be cascaded to compute the composition of two functions in hardware size equal to the sum of the hardware sizes for each of the functions. The output v_0 of the first aggregate β would be connected to the input v' of β', and the register $R_{v'}$ of β' would be connected to the output specifier of β. The timing conventions for the input v' of β' would be changed to allow for the uncertain delay between an input request and its answer (signalled by v_1).

Definition. A family $\{\beta_n\}$ of aggregates is *uniform* provided the transformation $1^n \to \beta_n$ can be computed in deterministic space

$$O(\log h(\beta_n) + \log n).$$

The complexity classes defined by uniform aggregate families of bounded hardware and bounded time and of nonuniform bounded time families can be characterized as follows:

THEOREM 4.1. *Let* H, S *be fully space constructible functions with* $H(n), S(n) \geqslant \log n$. *Then*

(a) UHARDWARE (H) = DSPACE (H),

(b) HARDWARE (H) $\subseteq$ DSPACE (H) (NONUNIFORM),

(c) UAGTIME (S) = UDEPTH (S),

(d) AGTIME (S) = DEPTH (S).

This theorem shows that neither of the resources uniform hardware and uniform aggregate time define new complexity classes in themselves. However, taken together they define apparently new and natural simultaneous complexity classes. Simultaneous resource bounds are discussed in section 7.

Proof sketch (a) and (b): A deterministic Turing machine can simulate an aggregate by updating a bit vector which has one bit for the output of each gate. A queue is kept of the next $\lceil \log n \rceil$ input values for each input node v to facilitate the update of these nodes. Note that there can be at most $O(H(n)/\log n)$ input nodes v, since each has an associated register R_v with $\lceil \log n \rceil$ gates.

An aggregate can simulate a (uniform) deterministic Turing machine for inputs of length n by having a "box" of gates devoted to each work tape square. The box records the current contents of that tape square, and if scanned, it records the state of the Turing machine. The contents of the input tape is obtained by an input node v, whose register R_v is attached to a counter which records the input head position. This simulation does not work for nonuniform Turing machines, since the reference tape could have length exponential in the size of the aggregate.

Proof sketch (c) and (d): An aggregate can be converted to a circuit by implementing each input node by the decoding circuit mentioned earlier. Then each gate v is replaced by a set $\{\langle v, t\rangle \mid 0 \leqslant t \leqslant S(n)\}$ of gates. The gate $\langle v, t+1\rangle$ has inputs from $\langle w_1, t\rangle$ and $\langle w_2, t\rangle$, where w_1 and w_2 are the inputs to v in the aggregate. The circuit output is $\langle v_0, S(n)\rangle$ (we can assume v_0 retains its value in the aggregate once $v_1 = 1$).

To convert a circuit to an aggregate, construct an input node v_i for each circuit input x_i. The register R_{v_i} has constant value i. Let v_0 be the output node for the circuit, and let v_1 be the end of a length $S(n)$ chain of identity gates having the constant function 1 at the beginning.

More details can be found in [D1].

The above theorem sheds some light on the old problem of to what extent feedback in circuits helps reduce the number of required gates. The best result in that direction seems to be due to Rivest [R2] who gives examples showing a linear reduction in size, but only for a multiple output circuit. On the other hand, theorem 4 suggests that disallowing feedback might cause an exponential size blow up in some cases. For example, let A be a set which is log space linear time complete for DSPACE (n) (see Hong [H2] for a natural example). Then by equation (a), A can be recog-

nized by an aggregate family of linear hardware size. On the other hand, as far as we know A requires exponential time on a Turing machine, so by theorem 2.1 it would follow that any *uniform* circuit family recognizing A has size at least 2^{n^ε} for some $\varepsilon > 0$ (indeed $2^{\Omega(n/\log^2 n)}$ by [P3]). In fact, we know of no way to reduce this bound even if we allow nonuniform circuit families.

Of course in other cases a proof that disallowing feedback causes exponential size blow up would imply P $\neq$ NP. For example, SATISFIABILITY can be recognized in linear space and hence is recognized by an aggregate family with linear hardware. If P = NP, then SATISFIABILITY would be recognizable by polynomial size circuits.

We close this section with two little results about aggregates in the style "if horses can whistle then pigs can fly". This style (but not these results) comes from the paper of Karp and Lipton [KL]. The results are intriguing because the hypotheses consist of assumptions concerning the nonuniform complexity of classes and the conclusions assert uniform complexity bounds.

THEOREM 4.2. *If* P $\subseteq$ HARDWARE (log n) *then*
P $\subseteq$ DSPACE (log n . loglog n)

Proof sketch: Since the circuit value problem (CVP) is log space complete for P (see [HU]), it suffices to prove CVP is in the second class given it is in the first class. Thus for each n we assume the existence of an aggregate β_n which correctly solves the CVP on inputs of length n, with $h(\beta_n) = O(\log n)$. A deterministic Turing machine M can represent and simulate a candidate β_n' for β_n in space O (log n . loglog n), and in fact M can cycle through all such candidates β_n'. There is no apparent way to determine in small space whether β_n' gives the correct answer for all inputs c of length n, but given a particular input c (i.e. circuit with inputs specified) M can check that β_n' gives consistent answers for each gate g of c by simulating β_n' three times, with c as input modified so that its output is each of the two inputs to g and g itself. If β_n' gives consistent answers for each gate of c, then β_n' correctly gives the output of c (i.e. tells whether $c \in$ CVP).

THEOREM 4.3. *If* NP $\subseteq$ HARDWARE (log n) *then*
NP $\subseteq$ DSPACE (log n . loglog n).

Proof sketch: It suffices to show that SATISFIABILITY is in the second class given it is in the first class. Reasoning as above, the Turing machine M can check whether a candidate aggregate β'_n correctly tells whether a propositional formula F is satisfiable by making β'_n produce a satisfying assignment bit by bit, by plugging in partial truth assignments to F and asking β'_n about the result. The trouble is M cannot remember the partial assignments in small space. However, the problem of whether "the i-th bit is 1 in the lexocographically first assignment which β'_n says satisfies F" is in P. Thus by theorem 4.2 this bit can be determined in space O ($\log n$. $\log\log n$), and M can determine whether this assignment satisfies F in small space.

5. Hardware Modification Machines

As mentioned in the introduction, there is a need to define a parallel model which is more powerful than an aggregate, in that it can modify its circuits, but less powerful than existing parallel RAM models, in that each unit of hardware can only perform a bounded amount of work in one step. We shall call the new machine a *hardware modification machine* (HMM), since it is intended to be the parallel analog of the storage modification machine. An HMM consists of a finite collection of finite state machines connected together as in a conglomerate. At each step, each machine may, in addition to assuming a new state and transmitting output signals, modify its input connections. Specifically, it may detach any of its inputs and re-attach it to a new machine which it brings into the HMM, or it may re-attach it to an output of any machine which can be reached by a path of length at most two traced backwards from the input.

One advantage of an HMM over circuits, aggregates, and conglomerates is that there is no question of uniformity. The machine is uniform because it constructs itself.

An HMM can execute an algorithm like the one described in [FW] to simulate a deterministic S space bounded machine in time O (S), and HMM time S can be simulated in deterministic space O (S^2). Thus the inclusions (1.2) apply.

The theory of HMM's is developed in [D1].

6. Other Modifiable Models

The first published parallel model introduced and compared in power to space bounded machines was the vector machine of Pratt and Stockmeyer [PS]. A vector machine is like a random access machine, except there are two distinct kinds of registers: index registers and vector registers. Addition, subtraction, and comparison operations can be applied to both kinds of registers, and both can be accessed via index registers. In addition, bitwise Boolean operations can be applied to vector registers, and vector registers can be shifted by an amount specified by an index register. These shift operations allow the vectors to grow in length exponentially in the computation time, and hence the bitwise vector operations represent a high degree of parallelism.

Pratt and Stockmeyer prove that vector machine time $(S) \subseteq \text{DSPACE}(S^2)$ and $\text{NSPACE}(S) \subseteq$ vector machine time (S^2), for suitable $S(n) \geqslant \log n$. These inclusions are weaker than either 1.1 or 1.2. It seems that vector machines have some very powerful operations, such as the shift, which preclude linear space simulation of time. On the other hand, they are apparently not powerful enough to allow a linear time simulation of space.

This aesthetic defect is balanced by other considerations. The model is a pleasant one, and is an extension of actual computer designs. Enough examples of vector machine algorithms are given in [PS] to indicate the machine's suitability for the programming of parallel algorithms. Simon [S4] proved the surprising result that the power of vector machines is only increased by application of a polynomial when no distinction is made between index registers and vector registers, so that a vector register can be shifted by an amount specified by another vector register.

The MRAM's and CRAM's of Hartmanis and Simon [HS] are similar to vector machines, except they have only one type of register, and perform multiplication (or concatenation) instead of shifting. The time space simulation results are similar to those for vector machines.

A number of other variations of parallel random access machines have been introduced. One example is Goldschlager's SIMDAG [G1], which stands for single instruction stream, multiple data stream, global memory. This consists of a control processor (CPU) and an infinite sequence PPU_0, $\text{PPU}_1, \ldots$ of parallel processors, each connected to an infinite random access global memory. In addition, each parallel processor has a local

infinite random access memory. The program is executed by the CPU, which can broadcast instructions to the active PPU's. Each instruction broadcast is executed by the first k PPU_i's, where k is stored in some location of global memory. Each PPU_i executes the same instruction, but the memory locations accessed can be indexed by the subscript i, and so can be different for different PPU_i's. The simulations proved for SIMDAG's are a little stronger than 1.2; namely, SIMDAGTIME (S) $\subseteq$ DSPACE (S^2), and NSPACE (S) $\subseteq$ SIMDAGTIME (S). The reason that nondeterministic space S instead of just deterministic space S can be simulated in time O (S) is apparently because of a powerful SIMDAG instruction which allows any number of PPU's to store into memory at once. If two or more try to store into the same location, the lowest numbered processor succeeds. This gives the effect of a huge fan-in being executed in one step.

The P-RAM of Fortune and Wyllie [FW] is similar to the SIMDAG, except different parallel processors can be executing different parts of their program at once, so it is "multiple instruction stream". Also, there is no instruction comparable to the SIMDAG's instruction which allows a potentially unbounded number of processors to try to store in a given location at once. Wyllie shows in [W1] that the multiple instruction stream gives only a constant factor time advantage over SIMDAG's. On the other hand, the unbounded fan-in for SIMDAG's seems to be a real advantage, since the time space simulation results for P-RAM's are those of 1.2; weaker than for SIMDAG's.

The PRAM's of [SS] have no global memory, but a given processor can initiate offspring processors. The time space simulation results in [SS] are weaker than either 1.1 or 1.2.

In conclusion, all the parallel models in this section have powerful instructions which cannot be considered primitive.

7. Simultaneous Resource Bounds

In section 2 we indicated that sequential time is roughly equivalent to uniform circuit size, and sequential space is roughly equivalent to uniform circuit depth. A natural question to ask is whether simultaneous time and space bounds are roughly equivalent to simultaneous uniform size and depth bounds. To be more specific, the well known class P can be characterized as either DTIME ($n^{O(1)}$) or as USIZE ($n^{O(1)}$), and "polylog space"

is both DSPACE $((\log n)^{O(1)})$ and UDEPTH $((\log n)^{O(1)})$. If we use the notation DTIME-SPACE (T, S) to refer to the class of sets accepted by some deterministic Turing machine which runs both in time T and space S, then the class referred to as PLOPS (polynomial time and polylog space) in [C1] (and now called SC by agreement among several authors), can be written DTIME-SPACE $(n^{O(1)}, (\log n)^{O(1)})$. Note that SC is presumably a proper subset of P $\cap$ DSPACE $((\log n)^{O(1)})$. For example, the graph reachability problem (GRP) (see section 2) is in the intersection class but not known to be in SC.

The corresponding circuit-defined class is USIZE-DEPTH $(n^{O(1)}, (\log n)^{O(1)})$, which is called NC in [C1] and [R1] after Pippenger, who first characterized it (see theorem 7.1). Again NC is presumably a proper subset of the intersection class, although it is remarkably difficult to think of a natural example of something in the intersection class but not in NC. (The universal set UPL defined below may be an artificial example.)

Getting back to the original question, we now ask whether SC = NC? The answer is apparently no, because there are natural problems in NC which do not appear to be in SC. One example is GRP (or any other complete problem for NSPACE $(\log n)$). Other examples are integer division and integer powering (see section 3). (Technically these should be made into recognition problems by specifying an index i as part of the input and asking whether the i-th output digit is 1.) And another class of examples are those context free languages which we don't know how to put into SC (see theorem 7.5 below).

Conversely, it is not so easy to find natural candidates for the difference set SC-NC. In fact, it is difficult to find sets in SC which are not clearly in DSPACE $(\log n)$ (and therefore in NC). Any universal deterministic context free language (DCFL) provides an example because of the result in [C1], but again Ruzzo proved that all CFL's are in NC.

One can still concoct artificial candidates for SC-NC. For example, a universal set UPL for SC^2 (SC^2 = DTIME-SPACE $(n^{O(1)}, \log^2 n)$) can be constructed as follows: Design a machine M which shuts itself off if it attempts to use more than $\log^2 n$ space or n^2 time. Let M on an input coding a pair (x, y) simulate machine number x on input y; and accept iff neither its pace nor time bound is exceeded and machine x accepts y. Then UPL is log space complete for SC^2 and does not appear to be in NC.

We conclude that time and space together do not seem to be even roughly equivalent to uniform size and depth together. However, Pippenger [P1] proves that time and reversal together do correspond to size and depth

together. Here the *reversal* of a computation of a multitape Turing machine is the number of times any of its heads changes direction (a hesitation is not a reversal). Pippenger proves

THEOREM 7.1. NC = DTIME-REVERSAL $(n^{O(1)}, (\log n)^{O(1)})$.

This is one characterization of NC, and Ruzzo [R1] points out several others. In fact, NC appears to be a very stable and interesting class. Intuitively, it is comprised of all problems which can be solved very rapidly on a parallel computer of feasible size. To make this statement more evident, we point out NC can also be characterized in terms of aggregates (see section 4).

THEOREM 7.2. NC = UHARDWARE-AGTIME $(n^{O(1)}, (\log n)^{O(1)})$.

This follows immediately from the definition of NC and the discussion in section 4 about converting circuits to aggregates and vice versa.

I would like to mention three of Ruzzo's [R1] characterizations of NC. First, Ruzzo gives several alternative definitions of *uniform circuit family*, including our definition in section 2, and proves that NC remains the same for all of them. In particular, NC remains unchanged when the strong definition of U_E uniform is chosen. From this and theorem 2.5 Ruzzo concludes the second characterization:

THEOREM 7.3. NC = ATIME-SPACE $((\log n)^{O(1)}, \log n)$.

The third characterization is

THEOREM 7.4. NC = AuxPDA TIME-SPACE $(2^{\log^{O(1)} n}, \log n)$.

Here AuxPDA stands for auxiliary pushdown automaton. The theorem holds whether it is deterministic or nondeterministic.

We now sketch the proof of another interesting Ruzzo result:

THEOREM 7.5. *Every context free language is in* NC.

Part of the interest of the proof is that it was apparently discovered using ATM's (via theorem 7.3), which is an indication that ATM's are a useful tool for discovering and expressing parallel algorithms. The proof of 7.5 follows the classical [LSH] proof that every CFL is in DSPACE $(\log^2 n)$, but needs a new idea. As in [LSH], we assume the grammar is in Chomsky form, and

try to verify the existence of a parse tree for the input string whose nodes have the form (σ, i, j) (which is *valid* if symbol σ generates the segment of the input between the i-th and j-th symbols inclusive). The ATM algorithm proceeds by guessing (via an existential state) a node (σ, i, j) which generates between one-third and two-thirds of the input string and then verifies (via a universal state) that both (1) (σ, i, j) is a valid node, and (2) the original root is valid given (σ, i, j) is valid. These two subproblems are solved by executing the algorithm recursively. Since the depth of the recursion is $O(\log n)$, the alternating time is $O(\log^2 n)$, but unfortunately a general recursive call to the algorithm must remember up to $\log n$ hypothesis nodes $(\sigma_1, i_1, j_1), \ldots, (\sigma_k, i_k, j_k)$, which require a total of $\Omega(\log^2 n)$ space to express, so theorem 7.3 does not apply. The new idea is to keep the number of hypothesis nodes down to two, by guessing at a common ancestor to two of them whenever three hypotheses would otherwise be formed. This keeps the space down to $O(\log n)$, so 7.3 applies.

In addition to comparing time and reversal to size and depth, Pippenger also shows time and space together are roughly equivalent to size and width. To define the last resource, let us say a circuit is *synchronous* if its gates can be divided into levels such that all inputs to the gates at level l are either input nodes x_i or are from gates at level $l - 1$. Then the *width* of a synchronous circuit is the maximum of the number of gates at any level. Pippenger also gives a suitable definition of width for nonsynchronous circuits and proves several relations among width, size, space and time, of which the following is a corollary:

THEOREM 7.6. $SC = USIZE\text{-}WIDTH(n^{O(1)}, (\log n)^{O(1)})$.

Dymond [D1] extends Pippenger's results to relate space and reversals to uniform width and depth. Two easy observations along these lines are that theorem 7.6 still holds if USIZE is replaced by UDEPTH, and SC remains unchanged if time is replaced by reversal in its definition. In addition, it is not hard to see that SC can be characterized in terms of aggregates as follows:

THEOREM 7.7. $SC = UHARDWARE\text{-}AGTIME((\log n)^{O(1)}, n^{O(1)})$.

This result shows an interesting duality with theorem 7.2. The question of whether NC = SC becomes the question of whether hardware size can be traded for computation time in uniform aggregates, without exponential blow up in the other resource.

8. Open Questions

Among the basic open questions in computational complexity are the problems of finding lower bounds for various resources for any simple interesting problem. In particular, for sequential complexity, we don't have any nonlinear time lower bounds nor any nonlogarithmic (i.e. $\omega(\log n)$) space lower bounds on any natural problem in the class NP. For parallel complexity, the same state of ignorance applies to nonlinear circuit size, and nonlogarithmic depth and hardware. Theorems 2.2 and 2.4 indicate that a nonlogarithmic lower bound on circuit depth may be weaker (and therefore easier to obtain) than such a bound on space, so the depth question deserves more attention. (There are already results which show the depth complexity of some simple problems cannot be $\log_2 n + O(1)$: see Neciporuk [N1] and Hodes and Specker [HS2].)

For simultaneous resource bounds, the situation is almost as wide open, although Borodin and Cook [BC] have recently shown that sorting cannot be done *simultaneously* in linear time and logarithmic space. It would be interesting to get similar tradeoff results for other resource pairs, such as size versus depth and aggregate time versus hardware. Another problem is whether there exists any set whose minimum time complexity is at least, say, the square of its minimum space complexity (assuming the latter is at least $\Omega(n)$). Similarly for uniform size versus uniform depth and aggregate time versus hardware size. (We do know by Lupanov's result [S3] that most sets have (nonuniform) size exponential in depth.

Finally, the questions concerning SC and NC mentioned in section 7 are worth emphasizing. In particular, it would be nice to know whether one class is included in the other, and whether they are proper subsets of their (common) intersection class.

Acknowledgement. While forming my ideas on parallel computation, and in particular while writing this manuscript I had frequent conversations with my colleagues Allan Borodin and Patrick Dymond, and their ideas as much as mine have contributed to whatever insights appear here. My thanks also to Nicholas Pippenger for some very helpful conversations.

REFERENCES

[AKLLR] ALELIUNAS, R., R. M. KARP, R. J. LIPTON, L. LOVÁSZ and C. RACKOFF. Random walks, universal traversal sequences, and complexity of maze problems. *Proceedings 20th Annual Symposium of Foundations of Computer Science*, Oct. 1979, pp. 218-223.

[B1] BORODIN, A. On relating time and space to size and depth. *SIAM J. Comp. 6* (1977), pp. 733-744.

[BC] BORODIN, A. and S. COOK. A time-space tradeoff for sorting on a general sequential model of computation. *Proc. 12th Annual ACM Symposium on Theory of Computing*, April 1980, pp. 294-301.

[BW] BURKS, A. W. and J. B. WRIGHT. Theory of logical nets. In: *Sequential Machines: Selected Papers*, E. F. Moore, ed., Addison-Wesley, 1964, pp. 193-212.

[C1] COOK, S. A. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. *Proc. 11th Annual ACM Symposium on Theory of Computing*, May 1979, pp. 338-345.

[CS] CHANDRA, A. K. and L. J. STOCKMEYER. Alternation. *Conference Record IEEE 17th Annual Symposium on Foundations of Computer Science*, 1976, pp. 98-108.

[CKS] CHANDRA, A. K., D. C. KOZEN and L. J. STOCKMEYER. Alternation. IBM Research Report RC 7489, 1978.

[D1] DYMOND, P. *Simultaneous Resource Bounds and Parallel Computation.* Ph.D. thesis, University of Toronto, Dept. of Computer Science, 1980.

[FW] FORTUNE, S. and J. WYLLIE. Parallelism in random access machines. *Proc. 10th ACM Symposium on Theory of Computing*, May 1978, pp. 114-118.

[G1] GOLDSCHLAGER, L. A unified approach to models of synchronous parallel machines. *Proc. 11th Annual ACM Symposium on Theory of Computing*, May 1978, pp. 89-94.

[G2] —— Synchronous parallel computation, Ph.D. thesis and TR-114, Dept. of Computer Science, Univ. of Toronto, December 1977.

[H1] HOOVER, J. Some Topics in Circuit Complexity. M.Sc. thesis and TR-139/80, Univ. of Toronto, Dept. of Computer Science, Dec. 1979.

[H2] HONG, J. W. On some space complexity problems about the set of assignments satisfying a boolean formula. *Proc. 12th Annual ACM Symposium on Theory of Computing*, April 1980, pp. 310-317.

[HIM] HARTMANIS, J., N. IMMERMAN and S. MAHANEY. One-way log tape reductions. *19th Annual Symp. on Foundations of Computer Science*, Oct. 1978, pp. 65-71.

[HS] HARTMANIS, J. and J. SIMON. On the power of multiplication in random access machines. *Proc. of the 15th Annual IEEE Symposium on Switching and Automata Theory*, New Orleans, October 1974, pp. 13-23.

[HS2] HODES, L. and E. SPECKER. Lengths of formulas and elimination of quantifiers I. In: *Contributions to Mathematical Logic*, K. Schutte, ed., North Holland Publ. Co. (1968), pp. 175-188.

[HU] HOPCROFT, J. E. and J. D. ULLMAN. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[J] JONES, N. D. Space-bounded reducibility among combinatorial problems. *JCSS 11* (1975), pp. 68-85.

[K1] KOZEN, D. On parallelism in Turing machines. *Proc. of the 17th Annual Symposium on Foundations of Computer Science*, Houston, Texas, Oct. 1976, pp. 89-97.

[KL] KARP, R. M. and R. J. LIPTON. Some connections between nonuniform and uniform complexity classes. *Proc. 12th Annual ACM Symposium on Theory of Computing*, April 1980, pp. 302-309. (Also presented at the Specker Symposium on Algorithms and Complexity, Zurich, Feb. 1980).

[LSH] LEWIS, P. M., R. E. STEARNS and J. HARTMANIS. Memory bounds for recognition of context-free and context-sensitive languages. *IEEE Conference Record on Switching Circuit Theory and Logical Design*, 1965, pp. 191-202.

[MP] MULLER, D. E. and F. P. PREPERATA. Bounds to Complexities of Networks for Sorting and for Switching. *JACM, vol. 22, No. 2* (April 1975), pp. 195-201.

[N1] NECIPORUK, E. I. A Boolean Function. *Soviet Math. Dokl.* 7, 4 (1966), pp. 999-1000. Originally *Dokl. Akad. Nauk. SSSR 169, 4* (1966), pp. 765-766.

[P1] PIPPENGER, N. On simultaneous resource bounds (preliminary version). *Proc. 20th Annual Symposium on Foundations of Computer Science*, October 1979, pp. 307-311.

[P2] PATERSON, M. S. An introduction to boolean function complexity. *Astérisque, Société Mathématique de France 38-39* (1976), pp. 183-201.

[P3] PIPPENGER, N. Fast simulation of combinational logic networks by machines without random-access storage. *Allerton Conf. on Comm. Contr. and Comp. 15* (1977), pp. 25-33.

[PS] PRATT, V. and L. STOCKMEYER. A characterization of the power of vector machines. *JCSS 12* (1978), pp. 198-221.

[R1] RUZZO, W. L. On uniform circuit complexity (extended abstract). *Proc. 20th Annual Symposium on Foundations of Computer Science*, Oct. 1979, pp. 312-318.

[R2] RIVEST, R. L. The necessity of feedback in minimal monotone combinatorial circuits. *IEEE Trans. on Computers*, June 1977, pp. 606-607.

[S1] SCHNORR, C. P. The network complexity and the Turing machine complexity of finite functions. *Acta Inf. 7* (1976), pp. 95-107.

[S2] SCHÖNHAGE, A. Storage modification machines. Technical Report, Mathematisches Institut, Universität Tübingen, Germany, 1979.

[S3] SAVAGE, J. E. *The Complexity of Computing*. Wiley, 1976.

[S4] SIMON, J. On feasible numbers. *Proc. 9th Annual ACM Symposium on Theory of Computing*, May 1977, pp. 195-207.

[SS] SAVITCH, W. and M. STIMSON. Time bounded random access machines with parallel processing. *JACM 26*, January 1979, pp. 103-118.

[W1] WYLLIE, J. C. The complexity of parallel computations. Ph.D. thesis and TR-79-387, Dept. of Computer Science, Cornell University, 1979.

(Reçu le 5 juin 1980)

Stephen A. Cook

University of Toronto
Department of Computer Science
Toronto 181
Canada