INFORMATION AND COMPUTATION 80, 205–226 (1989)

Complexity Theory of Parallel Time and Hardware*

PATRICK W. DYMOND† AND STEPHEN A. COOK

Department of Computer Science,
University of Toronto, Toronto, Canada

The parallel resources time and hardware and the complexity classes defined by them are studied using the *aggregate* model. The equivalence of complexity classes defined by sequential space and uniform aggregate hardware is established. Aggregate time is related to (bounded fanin) circuit depth and, similarly, aggregate hardware is related to circuit width. Interelationships between aggregate time and hardware follow as corollaries. Aggregate time is related to the sequential resource reversal. Simultaneous relationships from aggregate hardware and time to sequential space and reversal are shown (and conversely), and these are used as evidence for an "extended parallel computation thesis." These simultaneous relationships provide new characterizations for the simultaneous parallel complexity class *NC* and for the complementary class *SC*. The evaluation of monotone planar circuits is shown to be in *NC*, in fact in *LOGCFL*.

1. INTRODUCTION

The effective use of parallel computers to accelerate computation is an important challenge throughout computer science. In complexity theory parallel time is clearly a fundamental resource for synchronous parallel computation. However, study of parallel time complexity classes alone does not give complete insight into the cost of a parallel computation; the number of processors and amount of circuitry involved are clearly also important. In this paper we examine the parallel resource *hardware*, and relate hardware complexity classes to previously studied resource classes. We also define and study complexity classes obtained by simultaneously bounding both parallel time and hardware. A general method for accelerating certain sequential computations at reasonable hardware cost is described.

* This paper is based on the first author's Ph.D. thesis. A preliminary version of this paper appeared as (Dymond and Cook, 1980). This research was supported in part by the Natural Sciences and Engineering Research Council of Canada, and by the National Science Foundation under Contract DCR-8604031.

† Current address: Department of Computer Science and Engineering, Mail Code C-014, University of California, San Diego, La Jolla, CA 92093.

0890-5401/89 $3.00

As motivation for studying hardware complexity, observe that while there has developed a substantial body of evidence showing that sequential Turing machine space can be simulated efficiently by parallel time (see Chandra and Stockmeyer, 1976; Goldschlager, 1977; where this relationship and its converse are called the "parallel computation thesis"), these simulations do not constrain the hardware used by the parallel machine, and in fact use an exponential (in the Tm space) amount of hardware. For example, the set of satisfiable formulas of propositional calculus can be accepted in linear space, thus in linear time on a parallel RAM (Goldschlager, 1977) but if $P \neq NP$, more than a polynomial amount of hardware will be required to do so.

To lay a foundation for study of the parallel resource hardware we found it appropriate to introduce a new theoretical model of parallelism. The trouble with existing models such as parallel RAMs (Fortune and Wyllie, 1978; Savitch and Stimson, 1979), is that the natural unit of hardware (the processor) is already unbounded in size, and so a hardware analysis would have to consider not only the number of processors but also the number of bits per word of memory and the number of words of global and local memory used. Our proposed model is the *aggregate* which is like a boolean combinational circuit with the addition of feedback. For this model a very close relationship between hardware and sequential space will be demonstrated, and in fact this relationship does not depend on using an exponential amount of parallel time. We show that the complexity classes defined by parallel time on the model are closely related to those defined using previously studied parallel models (such as circuits, alternating Turing machines, and varieties of parallel random access machines), as well as to those defined using the sequential (Turing machine) resource *reversal*. Reversal-bounded complexity classes have been studied before (Hartmanis, 1968); and Pippenger (1979) has used the concept in his study of simultaneous relationships between Turing machine and circuit complexities in which time and reversal are related to circuit size and depth. Although Pippenger did not explicitly discuss parallel hardware, he introduced the idea of *width* of a boolean circuit, a closely related concept. In this paper we discuss simultaneous relationships between sequential (space, reversal), parallel (hardware, time), and circuit (width, depth) resources. These relationships provide evidence for an extended version of the parallel computation thesis mentioned above.

Our basic model of sequential computation is the multi-tape (offline) Turing machine (Tm) with a separate read-only input tape, as discussed, for example, in (Hopcroft and Ullman, 1979). The *reversal* used in a Turing machine computation is 1 plus the number of time steps at which any of the heads changes the direction of its motion. (A pause is not a reversal.) A set A is accepted in reversal $R(n)$ if there exists a Tm which accepts A using

at most $R(n)$ reversals on inputs of length n. Because we will be comparing Tm complexity classes with those of circuits, which recognize only sets over the alphabet $\{0, 1\}$, we will restrict our definitions of Tm time, space, and reversal complexity classes to sets over $\{0, 1\}^*$; this involves no important loss of generality in our results. Thus $TIME(T) = \{A \subseteq \{0, 1\}^* \mid A$ is accepted in time $O(T(n))$ by a deterministic multitape Tm$\}$ and the classes $SPACE(T)$ and $REVERSAL(T)$ are defined analogously.

We shall assume that our resource bounding functions map natural numbers to natural numbers, and that they always grow at least as fast as $\log n$. We will also require them to be *strongly constructible*, defined as follows:

A function $T: N \to N$ is strongly constructible provided there is a deterministic Tm which, on input 1^n, computes $T(n)$ in binary using space $\max\{\log T(n), \log n\}$.

These "strongly constructible" functions correspond to the "linear space honest" functions of Seiferas (1976); they comprise a rich class of functions including most bounding functions of practical interest (e.g., polynomials, $[\log n]^k$).

A basic model of parallel computation which we use is the (boolean combinational) circuit family, which is described in (Borodin, 1977; Cook, 1980; Hoover, 1979; Pippenger, 1979; Ruzzo, 1981.) We review the relevant definitions here. A circuit α_n of n inputs is a directed acyclic graph with nodes labelled from $\{x_1, x_2, ..., x_n\} \cup B_1 \cup B_2$, where $B_i = \{f \mid f: \{0, 1\}^i \to \{0, 1\}\}$ = the set of boolean functions of rank i. Nodes of α_n labelled with x_i are called *inputs* and have indegree 0; a node v with label g in B^i (called a *gate*), has indegree i and one edge into v is associated with each argument of g; there is a sequence of nodes O designated as *outputs*, and every node of α_n lies on a path from an input to an output. An assignment x of n values from $\{0, 1\}$ to the n inputs of α_n induces values on every node of the circuit in the obvious way, and we say that the sequence of values induced on the nodes of O is the *output* of the circuit. α_n computes a function f^n on the strings over $\{0, 1\}^n$ if, for every input assignment $x \in \{0, 1\}^n$, α_n outputs $f^n(x)$.

Let $size(\alpha_n)$ be the number of gates in α_n. The depth $d(v)$ of a node v is 0 if v is an input node; otherwise $d(v)$ = the length of the shortest path from an input to v. $Depth(\alpha_n)$ is $\max_{v \in \alpha_n} d(v)$, and $width(\alpha_n) = \max_{1 \leq i \leq depth(\alpha_n)} \#\{v \mid 0 < d(v) \leq i$ and there is an arc from v to a node w and $d(w) > i\}$. This notion of circuit width, introduced in Pippenger (1979), intuitively corresponds to the optimal number of gate values one would need to remember so as to avoid any recalculation while determining the output of the circuit by first evaluating all gates of depth 1, then using these values to evaluate all gates at level 2, etc., ignoring costs associated with re-examining inputs. For $A \subseteq \{0, 1\}^*$, let $A^n = A \cap \{0, 1\}^n$. A circuit α_n

recognizes A_n if it has a single output node and computes the characteristic function of A^n. A family $\{\alpha_n\}$ of circuits recognizes A if for every n, α_n recognizes A^n. Finally, define $SIZE(L)$ $(DEPTH(L), WIDTH(L)) = \{A \mid \text{some}\{\alpha_n\}$ recognizes A and $size(\alpha_n)$ (resp. $depth(\alpha_n)$, $width(\alpha_n)) = O(L(n))\}$.

The following are basic properties which our Tm's and circuits are taken to have.

PROPERTY T. If M is a Tm which works in time $T(n)$, space $S(n)$, and reversal $R(n)$, then

$$T(n) = \Omega(n),$$

$$S(n) = O(T(n)) \text{ and } R(n) = O(T(n)),$$

$$T(n) = O(R(n) \cdot (n + S(n))),$$

$$S(n) = \Omega(\log T(n)) \text{ and } R(n) = \Omega(\log T(n)).$$

PROPERTY C. If $\{\alpha_n\}$ is a family with $size(\alpha_n) = L(n)$, $width(\alpha_n) = W(n)$, and $depth(\alpha_n) = D(n)$, then

$$L(n) = \Omega(n),$$

$$D(n) = O(L(n)) \text{ and } W(n) = O(L(n)),$$

$$L(n) = O(W(n) \cdot D(n)),$$

$$D(n) = \Omega(\log L(n)) \text{ and } W(n) = \Omega(\log L(n)).$$

All the assertions of Property T can be proved for any halting Turing machine which examines all bits of its input using straightforward techniques, given the convention that all complexity bounds are $\Omega(\log n)$. For example, one shows that $T(n) = 2^{O(R(n))}$ by observing that the number of moves between a reversal depends on the amount of nonblank work-space available, which can be increased by at most a constant factor plus n after each reversal.

The assertions of Property C (except the last) can similarly be established by using the assumption that there is a path from every input node and every gate to the output node. We consider here cases where circuit width is at least as large as the log of the circuit size. Pippenger has observed that by writing the function computed by a circuit as a formula in disjunctive normal form one can construct an equivalent circuit of constant width. Consequently, all sets (even nonrecursive ones) are in $WIDTH(O(1))$. However, there is a sense in which width does correspond closely to Tm resources if we restrict attention to cases where the restriction $W(n) = \Omega(\log (L(n)))$ is satisfied.

2. AGGREGATES

Circuit depth provides a useful characterization of parallel time but the size of a circuit does not necessarily properly estimate parallel hardware requirements. The problem is due to the fact that a circuit has no way to re-use its gates as its computation proceeds, while a reasonable parallel machine would presumably be able to re-use its hardware. The aggregate model defined here combines features from the sequential circuits or logical nets of switching theory (e.g., Burks and Wright, 1964; Ofman, 1965) and from Goldschlager's conglomerate model while differing a bit from both. Like circuits, each aggregate works for exactly one input length and each gate computes only boolean functions of degree $\leqslant 2$. Like the finite state controls in conglomerates, the gates of an aggregate work synchronoulsy so that there is no possibility of ambiguity in a computation. And unlike either sequential circuits or conglomerates, the input convention for aggregates is such that either the hardware used or the time taken in a computation can be sublinear in the input size.

Formally, an aggregate β_n on n inputs $x_0, x_1, ..., x_{n-1}$ is a directed graph (not necessarily acyclic) whose nodes have labels from $B_2 \cup B_1 \cup \{x\}$ (recall $B_i = \{f \mid f:\{0, 1,\}^i \to \{0, 1\}\}$.) A node v with label $f \in B_i$ must have indegree i, and one in-edge of v is associated with each argument of f. A node with label x is an *input* node and must have indegree zero. Associated with each input node v is a *register* R_v consisting of a sequence of $[\log n]$ distinct nodes, which is used to specify which input x_i will be assigned to v. (Distinct nodes have disjoint registers.) There is also a distinguished pair of nodes (v_0, v_1) called *output* nodes. A *configuration* C of β_n is an assignment of 0 or 1 to each node v of β_n (the assigned value is called the *output* of v). A *computation* of β_n is a sequence $C_0, C_1, ..., C_T$ of configurations which satisfies:

(a) All nodes in C_0 have output 0 except for nodes computing the constant function 1.

(b) If a node v has label $f \in B_i$, then in C_{t+1} it has output equal to f 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 $t \leqslant [\log n]$ and, in general, in $C_{t+[\log n]}$ v has output x_i, 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, *time*(β_n), is the maximum over all inputs of the above index T. The hardware size, *hardware*(β_n), is the number of nodes in β_n. (For aggregates viewed as transducers, i.e., computing a function, we extend this definition slightly,

see Cook, 1980.) We shall only consider aggregates which are well defined in the sense they have an output as defined above for every input of length n.

The unusual input convention we have chosen needs justification. The reason that inputs x_i are not fed directly into aggregates (as they are for boolean circuits) is that this would entail $hardware(\beta_n) \geqslant n$, whereas we are interested in sublinear hardware bounds. In fact, the value of an input node v could be computed from the index in R_v using a boolean selection circuit of size $O(n)$ and depth $O(\log n)$. (This is the reason for the delay of $[\log n]$ time units for R_v to affect v). Our convention of not counting the size of the selection circuit is similar to the convention of not counting the input tape in measuring space used by an off-line Turing machine. In any case, this convention does not significantly affect cases in which the hardware bound is $\Omega(n)$.

An aggregate β_n recognizes a set $A^n \subseteq \{0, 1\}^n$ if β_n outputs 1 on a input $x = x_0 x_1 \cdots x_{n-1} \Leftrightarrow x \in A^n$. A family of aggregates $\{\beta_n\}$ recognizes a set $A \subseteq \{0, 1\}^*$ if β_n recognizes A^n for all n. We define

$AG\text{–}TIME(T) = \{L \mid L$ is recognized by a family of aggregates $\{\beta_n\}$ and $time(\beta_n) = O(T(n))\}$.

$AG\text{–}HARDWARE(H) = \{L \mid L$ is recognized by a family of aggregates $\{\beta_n\}$ and $hardware(\beta_n) = O(H(n))\}$.

According to the definitions above, it is possible for an aggregate family to recognize a non-recursive set (just as a circuit family can) because of the fact that a different machine is allowed for each input size. Since we wish to relate aggregate and circuit resources to those of Turing machines, we have to either add power to our Turing machines or restrict the class of circuits and aggregates under consideration. The approach used by Borodin (1977) is to require that the family be uniform in some sense; that is, that the members of the family be enough alike so that there would be an efficient algorithm for describing the circuit for a given input size. Unfortunately, there is no clear best choice for a precise definition of "efficient algorithm" in the above sentence, and a variety of possibilities have been suggested. Here we will use the following definition for uniform circuit, due to Borodin and Cook.

DEFINITION. A circuit family $\{\alpha_n\}$ is *uniform* if the transformation $1^n \to \bar{\alpha}_n$ can be computed by a deterministic Tm in space $O(\log(size(\alpha_n)))$. (Here the notation $\bar{\alpha}_n$ is used to denote a binary string coding the circuit α_n in a straightforward way; e.g., by specifying a distinct number for each node along with its type and the numbers of its input. It is useful to require in addition that the node numbers be topologically ordered, so that a node has inputs only from lower numbered nodes. Since the exact coding

scheme does not play a significant role in the proofs to be outlined below, we do not further specify it here. (See Ruzzo (1981) for a more extensive discussion.) Uniformity of aggregate families is defined in a manner similar to that of circuit families:

An aggregate family $\{\beta_n\}$is uniform if the transformation $1^n \to \bar{\beta}_n$ can be computed by a deterministic Tm in space $\log(\mathit{hardware}(\beta_n) \cdot \mathit{time}(\beta_n))$.

The uniform aggregate and circuit complexity classes *UAG–TIME*, *UAG–HARDWARE*, *USIZE*, *UWIDTH*, *UDEPTH* are defined in the same way as the already defined non-uniform classes, except that now the relevant circuit or aggregate family must be uniform. Analogously to Properties T and C, the following can be assumed to hold for aggregates:

PROPERTY A. If $\{\beta_n\}$ is a family of aggregates working in time $T(n)$ and hardware $H(n)$ then

$$T(n) \cdot H(n) = \Omega(n),$$
$$T(n) = \Omega(\log(H(n)),$$
$$H(n) = \Omega(\log(T(n)).$$

The first property holds by assuming that the aggregate examines all inputs; the second may be assumed because if the hardware is larger than an exponential in the time, then some gates can never affect the output (by bounded fanin) and so may be discarded; the third is because an instantaneous configuration of an aggregate of hardware $H(n)$ can be described in $O(H(n))$ bits and no halting computation repeats an instantaneous configuration.

We first note that the parallel time classes defined by aggregates are the same as those defined by circuits.

THEOREM 1. (a) $AG\text{–}TIME(T) = DEPTH(T)$
(b) $UAG\text{–}TIME(T) = UDEPTH(T)$.

Proof Sketch. An aggregate can be converted to a circuit by implementing each input node by the selection circuit mentioned earlier. Then each gate v is replaced by a set $\{\langle v, t\rangle \mid 0 \leqslant t \leqslant T(n)\}$ 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 imputs to v in the aggregate. The circuit output is $\langle v_0, T(n)\rangle$ (we 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 $T(n) + \log n$ chain of identity gates having the constant function 1 at the beginning. In fact no assumption of constructibility is needed in part (a) of this theorem,

since for both circuits and aggregates a different member of the family is used for each input size. For part (b), T must be constructible to ensure uniformity. ▮

COROLLARY 2. $DSPACE(S) \subseteq UAG\text{-}TIME(S^2) \subseteq DSPACE(S^2)$.

Proof. By a result of Borodin (1977), $DSPACE(S) \subseteq UDEPTH(S^2) \subseteq DSPACE(S^2)$. ▮

While aggregate time is related by the above to circuit depth and sequential space, there is an even closer relationship among uniform aggregate hardware, uniform circuit width, and sequential space.

THEOREM 3. $DSPACE(S) = UAG\text{-}HARDWARE(S) = UWIDTH(S)$.

Proof. To show $UAG\text{-}HARDWARE(S) \subseteq UWIDTH(S)$, the construction of Theorem 1 is modified by implementing each input node by a selector circuit of width $O(\log n)$. This selector subcircuit computes $\bigvee_{0 \leq i \leq n-1} \{x_i \wedge \text{value in } R_v \text{ is } i\}$, where the last inner term is computed by ANDing the (possibly inverted) bits of the gates representing R_v. Once the output of this selector subcircuit has been obtained, it can be carried down to the layer where it will be needed $[\log n]$ layers below. Since there are at most $cS(n)/[\log n]$ input nodes, at most this many lines need be carried down from this level, so that the total width needed is $O(S)$. It can be noted that this simulation can be carried out by a *synchronous* circuit, in which the circuit is divided into levels such that each gate on a level has all its inputs from gates on the previous level, without increasing the width bound.

To show $UWIDTH(S) \subseteq DSPACE(S)$, we first discuss the simulation of a synchronous circuit. A Turing machine M simulates a circuit α_n level-by-level starting at level 1, by keeping a list of $W = cS(n)$ bits, one bit for each gate on that level. The order of the W bits corresponds to the order in which the W gates appear in the circuit description. To determine a gate's type and the positions of its inputs on the previous level, M uses extra space $O(\log(size(\alpha_n))) = O(S)$ (see Property C) to generate a description of the circuit. (In fact, this description may require too much space to write down, but M ignores most of it and retains only the information needed for the gate of current interest). If the circuit family is not synchronous, M evaluates the gates successively in the topological order specified by the encoding. As before a list of up to W bits is kept, representing the values held by exactly those previously evaluated gates which are used as inputs to the current gate or to any gate after it in the topological ordering. This set of gates can be enumerated in space log of the size of the circuit to match positions in the list with gate numbers.

To show $DSPACE(S) \subseteq UAG\text{-}HARDWARE(S)$, an aggregate simulates a space S Tm M with state set Q and worktape alphabet $\{0, 1\}$ by having a "module" of gates to represent each worktape square, in the style of (Pippenger, 1973). Each module remembers a "worktape square configuration" which consists of a pair (q, σ), $q \in Q \cup \{\phi\}$, $\sigma \in \{0, 1\}$. If M's worktape head is currently visiting a particular square with symbol σ and M is in state q, the corresponding module is remembering (q, σ); each module corresponding to any other square remembers (ϕ, σ') where σ' is the appropriate tape symbol. Every module is connected to its right and left neighbors, to the input node v, to an associated binary counter which serves as R_v, and to the "clock" which ensures that the modules pause $[\log n]$ steps between simulation steps (to allow time for update of the input node).

The module currently remembering a square configuration with a non-null state (i.e., one of the form (q, σ)) has all the information it needs to simulate one step of the machine (current input symbol, current worktape symbol σ, current state q) and so can calculate its own new symbol, send the appropriate signal to update the input counter, and notify either its left or right neighbor of the new state (depending which way the head moves). Each module can be constructed from a constant number of gates (i.e., depending on M but not the length of the input), the input mechanism requires $O(\log n)$ gates and so the total hardware used is $O(S)$. The uniformity of the aggregate family follows from the fact that S is strongly constructible. ▮

As an immediate consequence of the theorem and Corollary 2, we observe a striking relationship between uniform aggregate time and uniform aggregate hardware.

COROLLARY 4. (i) $UAG\text{-}HARDWARE(H) \subseteq UAG\text{-}TIME(H^2)$

(ii) $UAG\text{-}TIME(H) \subseteq UAG\text{-}HARDWARE(H)$

The relationship between aggregate hardware and Turing machine space can also be investigated in the non-uniform setting, and again a correspondence becomes evident, see (Dymond and Cook, 1980). The close relationships between uniform aggregates, Turing machines and uniform circuits demonstrated above indicate that uniform aggregate resources do not define new complexity classes taken separately. We will now consider new complexity classes obtained by simultaneously bounding both time and hardware on uniform aggregates.

3. Simultaneous Classes

To discuss simultaneous complexity classes we use the following notation: for a particular model of computation X (such as the Tm), particular resources A, B (such as time, space) and bounding functions f and g, let $X-A, B(f, g)$ denote the class of languages recognized by machines of model X using no more than $O(f(n))$ of resource A, and simultaneously using no more than $O(g(n))$ of resource B. (We will usually omit X when the resources make the model clear.)

For example, $TIME, SPACE(n^3, \log^2 n)$ denotes the class of languages which are each accepted by a (deterministic) Tm which works in time $O(n^3)$ and space $O(\log^2 n)$, and $UAG\text{–}TIME, HARDWARE(T, H)$ denotes the class of languages each accepted by a uniform aggregate family which works in time $O(T(n))$ and uses hardware $O(H(n))$. Note that these simultaneous classes are different in general from the intersection of the corresponding single resource classes; for example, the set $A = \{0^n 1^n \mid n \geqslant 0\}$ can be recognized in either space $O(\log n)$ or reversal $O(1)$, but it can be shown that the product of space and reversal used by any Tm accepting A must be $\Omega(n)$, and hence that A is not in the class $REVERSAL, SPACE(1, \log n)$.

We will continue to require that our complexity bounding functions be strongly constructible and grow at least as fast as $\log n$. In addition, when considering the simultaneous complexity class $X-A, B(f, g)$, we shall assume without significant loss of generality that:

(a) $f(n) = \Omega(\log(g(n))$

(b) $g(n) = \Omega(\log(f(n))$

(c) $f(n) \cdot g(n) = \Omega(n)$.

For the resources and models under consideration here, (a) and (b) and (c) follow directly from properties A, T, and C, except in one case. For Turing machines with resources (reversal, space), the proof that the restriction (c) imposes no unacceptable loss of generality follows from Hong (1984) where it is shown that a Tm violating (c) accepts only a regular set.

Two complexity resources A, B are said to be *polynomially related* if for any bounding function T, the class $A(T)$ is a subset of $B(T^{O(1)})$ and conversely, $B(T) \subseteq A(T^{O(1)})$. (The superscript $O(1)$ notation is as used in Pippenger (1979) to denote an implicit union over all constant exponents.) Similarly, a pair A_1, A_2 of resources on a model X and another pair B_1, B_2 on a model Y are *simultaneously polynomial related*, if for any bounding functions T, H,

$$X-A_1, A_2(S, T) \subseteq Y-B_1, B_2(S^{O(1)}, T^{O(1)})$$

and

$$Y - B_1, B_2(S, T) \subseteq X - A_1, A_2(S^{O(1)}, T^{O(1)}).$$

Pippenger (1979) has investigated simultaneous relationships between Tm's and uniform circuit families and has presented four theorems, two relating time, reversal and size, depth, and two relating time, space and size, width. These theorems can be stated as follows:

$$TIME, REVERSAL(T, R) \subseteq USIZE, DEPTH(T^{O(1)}, R \log^4 T) \quad (1)$$

$$TIME, SPACE(T, S) \subseteq USIZE, WIDTH(T^2, S) \quad (2)$$

$$USIZE, DEPTH(L, D) \subseteq TIME, REVERSAL(L^{O(1)}, D \log^2 L) \quad (3)$$

$$USIZE, WIDTH(L, W) \subseteq TIME, SPACE(L^{O(1)}, W \log L) \quad (4)$$

These results can be combined and extended to show:

THEOREM 5. $REVERSAL, SPACE(R^{O(1)}, S^{O(1)}) = UDEPTH, WIDTH(R^{O(1)}, S^{O(1)})$.

Proof. The theorem follows from (1)–(4) by first observing the following:

$$TIME, REVERSAL(T, R) \subseteq UWIDTH, DEPTH(T^{O(1)}, R^{O(1)}) \quad (5)$$

$$TIME, SPACE(T, S) \subseteq UDEPTH, WIDTH(T^2, S) \quad (6)$$

$$USIZE, DEPTH(L, D) \subseteq SPACE, REVERSAL(L^{O(1)}, D^{O(1)}) \quad (7)$$

$$USIZE, WIDTH(L, W) \subseteq REVERSAL, SPACE(L^{O(1)}, W^{O(1)}). \quad (8)$$

These follow from (1)–(4) because (1) the original inclusions hold when the right-hand sides are increased by changing size to width or depth, and time to space or reversal; and (2) the original inclusions still hold when $\log T$ is replaced by S or R and $\log L$ by W or D. (We are using the properties that $\log T = O(R)$, $\log L = O(W)$, $\log L = O(D)$.) By restriction (c) above, $S \cdot R = \Omega(n)$, so the larger of the two Tm resources space, reversal (for any Tm in the class $REVERSAL, SPACE(R, S)$) must be as large as $\sqrt{n}$ and also as large as $\sqrt{T/n}$ (since $T = O(n \cdot R \cdot S)$. If $S = \Omega(R)$, we apply (5), and if $R = \Omega(S)$ (6) to obtain

$$\begin{aligned} SPACE, REVERSAL(S, R) &\subseteq TIME, REVERSAL(S^{O(1)}, R) \\ &\subseteq UWIDTH, DEPTH(S^{O(1)}, R^{O(1)}) \\ REVERSAL, SPACE(R, S) &\subseteq TIME, SPACE(R^{O(1)}, S) \\ &\subseteq UDEPTH, WIDTH(R^{O(1)}, S). \end{aligned}$$

Similarly for circuits, one of the two resources W, D must be as large as the square root of the size. If $W = \Omega(\sqrt{L})$ apply (7), if $D = \Omega(\sqrt{L})$ (8):

$$\begin{aligned} UWIDTH, DEPTH(W, D) &\subseteq USIZE, DEPTH(W^{O(1)}, D) \\ &\subseteq SPACE, REVERSAL(W^{O(1)}, D^{O(1)}) \\ UDEPTH, WIDTH(D, W) &\subseteq USIZE, WIDTH(D^{O(1)}, W) \\ &\subseteq REVERSAL, SPACE(D^{O(1)}, W^{O(1)}). \end{aligned}$$

The two cases dealt with above do not exhaust all possibilities; it may be, for example, that neither the width nor the depth of a circuit family is asymptotically as large as $\sqrt{L}$. Nevertheless, for any specific value of n, one of the two resources must be as large as $\sqrt{L}$. It follows that any language A in $UWIDTH, DEPTH(W, D)$ can be expressed as the union of two disjoint sublanguages A_0 and A_1, satisfying $A_0 \in USIZE, DEPTH(W^{O(1)}, D)$, $A_1 \in USIZE, WIDTH(D^{O(1)}, W)$ and for any n, each sublanguage contains either all or none of the members of A of that length. By the arguments given above, these sublanguages are each members of the class $SPACE, REVERSAL(W^{O(1)}, D^{O(1)})$ and hence their union is also in this class (because of the strong constructibility of W and D). A similar argument for space, reversal classes completes the proof. ▮

COROLLARY 6. $TIME, REVERSAL, SPACE(T^{O(1)}, R^{O(1)}, S^{O(1)}) = USIZE, DEPTH, WIDTH(T^{O(1)}, R^{O(1)}, S^{O(1)})$.

Proof. Both the time and the size are polynomially related to the larger of S and R, hence to each other. ▮

This simultaneous polynomial correspondence can be extended to include uniform aggregate families.

THEOREM 7. *The following classes are equal*:

(i) $REVERSAL, SPACE(T^{O(1)}, H^{O(1)})$

(ii) $UDEPTH, WIDTH(T^{O(1)}, H^{O(1)})$

(iii) $UAG\text{–}TIME, HARDWARE(T^{O(1)}, H^{O(1)})$.

Proof. The proof that the third class is equal to the other two divides into cases, as above. When T exceeds H, the construction in the second half of the proof of Theorem 3 is used. When H exceeds T, the conversion of a circuit to an aggregate described in the proof of Theorem 1 suffices. ▮

These theorems establish polynomial relationships between parallel time and reversal, as well as between hardware and space. We have. of course, observed the relationship between hardware and space earlier, but the

correspondence between aggregate time (or circuit depth) and reversal is new, although it could be inferred from Section 3 of (Pippenger, 1979). We obtain a more precise bound for the simulation of aggregate parallel time by sequential reversal in the next theorem.

THEOREM 8. $UAG\text{–}TIME(T) \subseteq REVERSAL(T^2)$.

In fact, we will also show that the space required in this simulation is polynomially related to the product of the hardware of the aggregate and T. We will first need the following "data processing" lemmas, which provide a method of sorting using small reversal and a method for obtaining a small reversal computation from a small space one.

LEMMA 9. *A file of n records of length m each with a key of length p can be sorted by a* Tm M *in simultaneous space $O(n \cdot m)$ and reversal $O(p)$.*

Proof Sketch for Lemma 9. The idea is to use a "radix sort", least significant digit first. To simplify the discussion, assume that the records consist only of the keys of p bits, and that the records are written one after another on a worktape W, separated by the symbol #. Four additional tapes will be used, denoted by F, X_0, X_1, and C. To begin, M initializes tape C by marking off an area of size $n(p+1)$ with every p consecutive blanks separated by a #. This can be done in constant reversal by copying from W. Tape C will be used to provide n unary counters which will hold values from 1 to p (right-justified), indicating which digit position of the keys is currently of interest for the radix sort. Prior to each of the stages described below all counters of C will hold the same value. Supposing that at some stage every area between the #'s on C has exactly i ones right-justified, it is easy to see that with a constant number of reversals M can make a complete pass through the records on tape W, copying every key followed by a mark bit followed by #, onto tape F, such that the mark bit is 1 iff the ith bit of the key is 1; and updating the counters on tape C so that each now contains $i+1$ ones. Now with another (backwards) pass over tape F, keys can be classified and copied onto either tape X_0 or X_1 depending on whether the mark bit is 1 or 0. Finally, another pass permits copying the records back to tape W with the records from X_0 preceding those from X_1. Thus using constant reversal M accomplishes one stage of a radix sort. p such stages suffice for a complete sort. ▮

Pippenger observed that a Tm working in space $S(n)$ could be simulated in reversal $S(n)^{O(1)}$ by reduction to sorting. Using his technique with Lemma 9 we can get bounds of reversal $O(S^2(n))$ and space $2^{O(S(n))}$.

LEMMA 10. $DSPACE(S(n)) \subseteq REVERSAL(S^2(n))$.

Proof Sketch. Let M be a Tm which works in space $S(n)$. A configuration of M is the state, contents of the worktape (with head position marked), and a binary string of $[\log n]$ bits specifying the position of the input head. All space S configurations can be recorded in total space $2^{O(S)}$. These configurations can be generated and recorded on a tape in reversal $O(S)$ by a "repeated doubling" technique. We describe the similar task of generating all S bit binary numbers, the above task involves a few straightforward additional details. The induction hypothesis is that given a list of all i binary numbers separated by $\#$, a list of all $i+1$ bit binary numbers can be generated in constant reversal. This is done of course by making two copies of the list, on one of which each number is followed by a zero, and on the other by a one; and then concatenating these two lists.

Given a list of all space S configurations, by sorting on input value each configuration can be given the actual bit of the input it refers to in $O(\log n)$ reversal. Next, each configuration record is modified so that is also holds the representation of its one step successor configuration (or itself as successor if it is the unique accepting configuration.) At this point we have a list of pairs (u, v) in which every possible configuration u appears in a pair (u, v) where the corresponding v is the 2^0 step successor of u. Assuming for induction each pair contains a configuration and its 2^i step successor, we show below how to obtain a list where every configuration appears with its 2^{i+1} step successor using reversal logarithmic in the length of the list, i.e., $O(S)$. So after $O(S)$ such phases the list will contain the $2^{O(S)}$ step successor of the initial configuration, with total reversal cost $O(S^2)$.

The idea for obtaining the 2^{i+1} successor from the list of 2^i successors is to view the list as a table containing function domain-range pairs, and to perform a functional composition. Using sorting and an extra copy of the list, we can bring together as a group all pairs $(u_1, v), (u_2, v), \cdots$ with the same range value v along with a copy of the pair (v, w) with v as the domain value. The range value w of this last pair must become the new range value for all other pairs in the group. To achieve this with small reversal, first w is duplicated $2^{O(S)}$ times, so that enough copies will exist to allow the replacement of the value in one more pass. The duplication can be accomplished for all groups simultaneously in $O(S)$ reversal, and this also bounds the sorting and grouping. ∎

Lemma 10, together with the fact that $UAG\text{-}TIME(T(n)) \subseteq DSPACE(T(n))$ from Corollary 2, is enough to establish Theorem 8. The more direct proof to be presented here also shows the time T, hardware H aggregate can be simulated in space $(H \cdot T)^{O(1)}$, whereas direct application of Lemma 10 and Corollary 2 would use space exponential in T.

Proof of Theorem 8. There are two phases to the simulation, computation of a description of the aggregate, and step-by-step simulation of

the aggregate. The uniformity of the aggregate ensures that its description can be computed in space $\log H \cdot T$, which is $O(T)$ by Property A. Using Lemma 10, the bits of the description of the aggregate can be generated within reversal $O(T^2)$ and space $(H \cdot T)^{O(1)}$. (Care must be taken here so that the TM works to generate all the bits of the output simultaneously in order that the reversals necessary for each bit are not multiplied by the number of bits.) In fact, if the time T, hardware H aggregate satisfies the more stringent uniformity criterion that its description can be computed in reversal T^2 and simultaneously space $O(H \log H)$, then space $O(H \log H + n \log n)$ would suffice for the entire simulation. The aggregate description is a set of H records, each of length $O(\log H)$ with keys (identification numbers for the gates) of length $\log H$. An additional n records, each containing a bit x^i of the input and having a binary key i, $0 \leqslant i \leqslant n-1$, can be obtained in $O(\log n)$ reversal by repeated doubling and a pass through the input.

Given values for all the gates at any time t, the values of all gates at time $t+1$ can be computed in reversal $O(\log(H+n))$ by sorting as in Lemma 9. The basic idea is that a master tape holds the number of each gate with its current value. The description of all gates can be first sorted by the number of each gate's left input. Then the values of each gate's left input can be obtained using constant reversal. Similarly, values of right inputs can be obtained with a second sort. Following this, a single pass through the records can be used to compute the new output values, and the master tape can be updated by means of one more sort. Values of inputs may be obtained and kept in a queue for each register. Thus one parallel step of β_n can be simulated in reversal $O(T)$, and T steps in reversal $O(T^2)$. ∎

A specific polynomial upper bound for the converse simulation of reversal by aggregate time and, simultaneously, space by hardware is not known because of the apparent need for the Tm to be oblivious (i.e., head motions are the same for all inputs of a particular length), which causes the exponent of the time bound to depend on the number of tapes being simulated. The main idea of the direct proof of polynomial correspondence is Pippenger's observation that a computation of an oblivious Tm between reversals is similar to that of a finite state transducer, and thus can be simulated by a circuit of small depth using the parallel prefix algorithm of (Ladner and Fisher, 1980; Offman, 1963.)

It follows by adapting Proposition 3.1 of Pippenger (1979) that a k-tape Tm working in space $S(n) = \Omega(n)$ and reversal $R(n)$ can be simulated by a Tm which runs in space $O(S^{k+2})$ and reversal $O(R^4)$ and which is "uniformly phase oblivious"; i.e., the machine first lays out $O(S^{k+2})$ squares on its worktape using reversal $O(R^2)$ and then simulates each phase (interval between reversals) of the original machine using the same

easily computed sequence of head motions for every input of length n and for every phase. (By easily computed here is meant that the head position at any time in the phase can be computed in space $O(\log R \cdot S)$.) Finally, a uniformly phase oblivious Tm working in space $S(n) = \Omega(n)$ and reversal $R(n)$ can be simulated by a uniform family of aggregates in time $O(R^2)$ and hardware $O(S)$ (Dymond, 1980).

4. Extended Parallel Computation Thesis

These results provide evidence for an extension to the original parallel computation thesis of Goldschlager (1977) and Chandra and Stockmeyer (1976).

Original Parallel Computation Thesis. *Space and parallel time are polynomially related.*

The extended version was proposed originally by Dymond (1980).

Extended Parallel Computation Thesis. (i) *Parallel time and hardware requirements are simultaneously polynomially related to sequential* (Tm) *reversal and space requirements.*

(ii) *Parallel time and hardware are polynomially related.*

Like Church's thesis and the original parallel computation thesis, this thesis cannot be proved, because it relates an intuitive concept (parallel computation) to a mathematically precise one (Turing machine). We can, however, obtain evidence for the thesis by showing that it holds for any "reasonable" model of a parallel machine.

Theorems 7 and 8 and the seminal ideas of (Pippenger, 1979) provide such evidence for part (i), and Corollary 4 and Lemma 10 suport part (ii). Other support for the thesis is contained in the subsequent work of Hong (1980) who both extended the class of machine models that provide evidence, to include hardware modification machines (Cook, 1980; and others), and generalized the thesis to consider nondeterministic and other types of machines.

The original thesis follows from the extended thesis by observing that space is related to parallel hardware (by part (i)) and hardware to parallel time (by part (ii)). Like the original thesis, this extended thesis provides no guarantee that the time of an arbitrary sequential computation may be reduced by parallelizing it; it merely states that the resulting hardware and parallel time will be polynomially related to the space and reversal of the original sequential computation. However, if the reversal of the original

computation is small then we can obtain a fast parallel computation using only polynomially more hardware than the original computation uses space. In contrast to this, the original thesis guarantees a fast parallelization if the original space is small, but does not bound the hardware required to achieve this in any way. In fact, we conjecture that no polynomial bound on hardware is possible, in general, when transforming space to parallel time and conversely; and that the correspondence presented here is the best possible (up to polynomial factors) when both hardware and parallel time are considered. To state this another way, we are conjecturing that it is not in general the case that

$$SPACE, REVERSAL(S, R) \subseteq parallel\text{-}time, hardware(S^{O(1)}, R^{O(1)}).$$

Note that by our thesis, this is equivalent to conjecturing that

$$parallel\text{-}time, hardware(T^{O(1)}, H^{O(1)}) \neq parallel\text{-}time, hardware(H^{O(1)}, T^{O(1)}).$$

5. Characterizations of NC and SC

It is widely believed that the "feasible" sets for sequential computers form a subclass of P. The belief that parallel algorithms which use a non-polynomial amount of hardware are as impractical as algorithms which require non-polynomial time suggests that the feasible sets for parallel machines are also a subclass of P, because polynomial hardware running for polynomial time can be simulated by a polynomial time sequential computation. While we know of no class which exactly characterizes the feasible sets for parallel machines, the simultaneous class $NC = USIZE, DEPTH(n^{O(1)}, (\log n)^{O(1)})$ proposed by Pippenger (1979) appears to be a very useful and stable characterization of those sets which can be accepted extremely quickly on parallel machines which use a feasible (i.e., polynomial) amount of hardware.

Pippenger showed that $NC = TIME, REVERSAL(n^{O(1)}, (\log n)^{O(1)})$, and Ruzzo (1981) has obtained results which show both that the class remains unchanged under a variety of definitions of circuit uniformity, and that it can be defined in terms of other machines including auxiliary pushdown machines (AUXPDMs) (Cook, 1971) and alternating Turing machines (ATMs) (Chandra, Kozen, and Stockmeyer, 1981.) More precisely, by closely relating alternating Tm's to uniform circuits, Ruzzo shows that

$$\begin{aligned} NC &= ATM\text{-}TIME, SPACE((\log n)^{O(1)}, \log n) \\ &= ATM\text{-}ALTERNATION, SPACE((\log n)^{O(1)}, \log n) \\ &= AUXPDM\text{-}TIME, SPACE(2^{(\log n)^{O(1)}}, \log n). \end{aligned}$$

For sequential computations a similar class of interest is $SC = TIME, SPACE(n^{O(1)}, (\log n)^{O(1)})$. The question of whether the two simultaneous classes NC and SC are equal was first considered (in a different form) by Borodin (1977) and the question is still open. We provide new characterizations for NC and SC in terms of the aggregate model as follows:

THEOREM 11.

$$NC = UAG\text{–}TIME, HARDWARE((\log n)^{O(1)}, n^{O(1)})$$

$$SC = UAG\text{–}TIME, HARDWARE(n^{O(1)}, (\log n)^{O(1)}).$$

Proof. Clearly $TIME, REVERSAL(n^{O(1)}, (\log n)^{O(1)})$ is a subset of

$$SPACE, REVERSAL(n^{O(1)}, (\log n)^{O(1)}).$$

Equality follows since even allowing polynomial space does not permit a substantial increase in time when the reversal is bounded. (Recall that time can be no more than $(n + \text{space}) \cdot \text{reversal}$.) Similar reasoning shows that

$$SC = SPACE, REVERSAL((\log n)^{O(1)}, n^{O(1)}).$$

The theorem then follows from Theorem 7. ∎

This shows that the question $SC = NC$ mentioned above is a particular case of the question of whether the parallel resources $TIME, HARDWARE$ are polynomially related to the resouces $HARDWARE, TIME$. Arguments based on pebbling lower bounds (Pippenger, 1980; Lengauer and Tarjan, 1982) suggest that neither NC nor SC is a subset of the other.

SC was studied in (Cook, 1979), where it was shown to contain all languages log space reducible to deterministic context-free languages. It follows from (Borodin, 1977) that $NL = NSPACE(\log n)$ is contained in NC (in fact, in NC^2, which is $USIZE, DEPTH(n^{O(1)}, \log^2 n)$). From the AUXPDM characterizations given above, Ruzzo was able to show that context-free language recognition problems are also in NC^2. We can use the same characterization to show that a problem first considered by Goldschlager, the monotone planar circuit value problem, is in NC^2.

In a monotone circuit, the only gate types are *and* and *or*, as well as input nodes. In a monotone planar circuit in standard form, nodes of the circuit are divided into *levels*, where a level is a sequence of gates and input nodes, satisfying the following planarity property. Let g_i^l denote the ith node on level l. Then g_i^{l+1} is either some input node x_j, or an *and* or *or* gate with left and right inputs (g_j^l and g_k^l, respectively) of two nodes on the lth level where $j \leqslant k$ and no gate after g_i^{l+1} in level $l+1$ has an input preceding g_k^l. Now define $\text{MPCVP} = \{x \mid x$ encodes a monotone planar

circuit in standard form and a set of inputs such that the output gate evaluates to *true*}.

Let *LOGCFL* denote the class of sets accepted by log space bounded (nondeterministic) auxiliary pushdown machines operating in polynomial time. Ruzzo (1980) has shown that $LOGCFL \subseteq NC^2$.

THEOREM 11. MPCVP $\in LOGCFL$, *so* MPCVP $\in NC^2$.

Proof Sketch. In a monotone circuit of n nodes with output gate true, there are paths from true inputs to the output gate, such that all gates along the paths are true. If the circuit is planar, such paths may still intersect because two true gates on a level may both depend on a common true gate from a previous level. However, as Goldschlager observed, there is a "proof tree" which attests to the fact that the output gate is true. A vertex $\langle g_k^l, g_m^l \rangle$ of this tree corresponds to a sequence of one or more adjacent true gates $g_k^l, g_{k+1}^l, \ldots, g_m^l$ on a level l of the circuit. A sequence of gates (i.e., a vertex of the proof tree) can be proved all true from the truth of non-overlapping sequences of gates (i.e., other vertices of the proof tree) at the previous level, or by being a sequence of true input nodes.

Formally, a proof tree is defined inductively to include the vertex corresponding to the output gate and, if $\langle g_k^l, g_m^l \rangle$ is a vertex in the tree, then either

(a) the kth through mth nodes at level l are true input nodes of the circuit, or

(b) there exists a set of p vertices (called the children of $\langle g_k^l, g_m^l \rangle$) $\{\langle g_{k_1}^{l-1}, g_{m_1}^{l-1} \rangle, \langle g_{k_2}^{l-1}, g_{m_2}^{l-1} \rangle, \ldots, \langle g_{k_p}^{l-1}, g_{m_p}^{l-1} \rangle\}$, $1 \leqslant p < n$, satisfying:

(1) $m_i < k_{i+1}$, i.e., the sequences of corresponding gates do not overlap, and

(2) the structure of the circuit (i.e., the types and connections of the gates involved) guarantees that if the gates corresponding to the sons all have output true, then the gates corresponding to $\langle g_k^l, g_m^l \rangle$ must all output true.

A nondeterministic pushdown machine M with auxiliary space $\log n$ can guess and verify the existence of a proof tree as follows:

Initially (after verification of the form of the input), the vertex corresponding to the output gate is placed on the stack. Throughout the computation, the vertices on M's stack will be vertices which are to be *verified.* (A vertex is verified when it is established that the corresponding nodes of the circuit are true.) As each vertex is to be verified it is removed from the stack; if the latest verification is successful and the stack becomes empty, M accepts. So M loops, removing the top vertex from the stack and

trying to verify it. If the current vertex corresponds to input nodes of the circuit, M verifies it by directly by consulting the input. Otherwise, M guesses in turn the children of the vertex, pushing each onto the stack. M must be sure that the vertices it pushes onto the stack could in fact be children of V in a proof tree for the circuit; given this assurance, the original vertex can be regarded as verified, subject to the verification of its children.

If there is a computation of M which guesses the proof tree correctly, in this computation M performs a depth-first search of the proof tree, so M clearly works in polynomial time. It is only necessary to be sure that M guesses the children insuch a way that no more than $O(\log n)$ space on the auxiliary worktape is needed (although there may be $\Omega(n)$ children), and so that exactly the possibilities with legal structure to form a proof tree can be guessed. Here we make the observation that M may guess the children and stack them one at a time working through the sequence of nodes on the level using $O(\log n)$ space to keep track of how far it has progressed. We leave the remaining details as an amusing exercise for the interested reader. ∎

The corresponding result in (Goldschlager, 1980) shows that MPCVP $\in DSPACE(\log^2 n)$, although our result is based on the ideas in an earlier version in which a $\log^3 n$ space bound was obtained. The theorem is interesting partially because Goldschlager (1977) had earlier shown that both the planar circuit value problem and the monotone circuit value problem were log-space complete for P and thus very unlikely to be in $\log^2 n$ depth or $\log^2 n$ space (see results of Cook and Sethi, 1973). Our theorem points out that when circuits are restricted to be both planar and monotone they can be evaluated very quickly in parallel, using a polynomial amount of hardware.

Acknowledgments

We thank Allan Borodin, Les Goldschlager, Jia-wei Hong, Nicholas Pippenger, Charles Rackoff, Walter Ruzzo, and Martin Tompa for valuable discussions about parallel computation, and an anonymous referee for helpful suggestions.

References

Burks, A. W., and Wright, J. B. (1964), Theory of logical nets, *in* "Sequential Machines: Semected Papers" (E.P. Moore, ed.), pp. 193–212, Addison–Wesley, Reading, MA.

Borodin, A. (1977), On relating time and space to size and depth, *SIAM J. Comput.* **6**, 733–744.

CHANDRA, A. K., KOZEN, D. C., AND STOCKMEYER, L. J. (1981), Alternation, *J. Assoc. Comput. Mach.* **28**, 114–133

CHANDRA, A. K., AND STOCKMEYER, L. J. (1976), Alternation, *in* "Conference Record IEEE 17th Annual Symposium on Foundations of Computer Science," pp. 98–108.

COOK, S. A., SETHI, R. (1973), Storage requirements for deterministic polynomial time recognizable languages. *J. Comput. System Sci.*, **7**, 354–375.

COOK, S. A. (1971), Characterizations of pushdown machines in terms of time-bounded computers, *J. Assoc. Comput. Mach.* **18**, 4–18.

COOK, S. A. (1979), Deterministic CFLs are accepted simultaneously in polynomial time and log squared space, *in* "Proceedings 11th Annual ACM Symposium of Theory of Computing, May 1979," pp. 338–345.

COOK, S. A. (1980), Towards a complexity theory of synchronous parallel computation, presented at "Internales Symposium über Logik und Algorithmik, Zurich, Feb. 1980"; *Enseign. Math.* **27** (1981), 99–124.

DYMOND, P. W. (1980), "Simultaneous Resource Bounds and Parallel Computation, Ph.D. thesis supervised by A. B. Borodin and S. A. Cook, Dept. of Computer Science, Tech. Report 145/80, University of Toronto.

DYMOND, P. W., AND COOK, S. A. (1980), Hardware complexity and parallel computation, *in* "Proceedings IEEE 21st Annual Symposium on Foundations of Computer Science," pp. 360–372.

FORTUNE, S., AND WYLLIE, J. (1978), Parallelism in random access machines, *in* "Proceedings 10th ACM Symposium on Theory of Computing," pp. 114–118.

GOLDSCHLAGER, L. M. (1977), "Synchronous Parallel Computation," Ph.D. thesis and Tech. Report 114. Dept. of Computer Science, University of Toronto; A universal interconnection pattern for parallel computers, *Assoc. Comput. Mach.* **29** (1982), 25–29.

GOLDSCHLAGER, L. M. (1980), A space efficient algorithm for the monotone planar circuit value problem, *Inform. Process. Lett.* **10**, 25–27.

HOPCROFT, J. E., AND ULLMAN, J. D. (1979), "Introduction to Automata Theory, Languages, and Comp'tation," Addison-Wesley, Reading, MA.

HARTMANIS, J. (1968), Tape reversal bounded Turing machine computations, *J. Comput. System Sci.* **2**.

HONG, J. W. (1980), The similarity and duality of computation, *in* "Conference Record IEEE 21st Annual Symposium on Foundations of Computer Science," pp. 348–359, On similarity and duality of computation (I), *Inform. and Control* **62** (1984), 109–128.

HONG, J. W. (1984), A trade-off theorem for space and reversal, *Theoret. Comput. Sci.* **32**, 221–224.

HOOVER, H. J. (1979), "Some Topics in Circuit Complexity," M.Sc. thesis and Tech. Report 139/80, Dept. of Computer Science, University of Toronto.

LADNER, R. E., AND FISCHER, M. J. (1980), Parallel prefix computation, *J. Assoc. Comput. Mach.* **27**, 831–838.

LENGAUER, T., AND TARJAN, R. E. (1982), Asymptotically tight bounds on time-space trade-offs in a pebble game, *J. Assoc. Comput. Mach.* **29**, 1087–1130.

OFMAN, YU. P. (1963), On the algorithmic complexity of discrete functions, *Soviet Phys. Dokl.* **7**, 589–591.

OFMAN, YU. P. (1965), A universal automation, *Trans. Moscow Math. Soc.* **14**, 200–215.

PIPPENGER, N. J. (1973), A relationship between machine time-complexity and network gate complexity, manuscript.

PIPPENGER, N. J. (1977), "Fast Simulation of Combinational Logic Networks by Machines without Random-Access Storage," IBM Research Report RC6582.

PIPPENGER, N. J. (1978), Course notes for CSC2429. Topics in the theory of computation, Dept. of Computer Science, University of Toronto.

PIPPENGER, N. J. (1979), On simultaneous resource bounds (preliminary version), *in* "Proceedings, 20th Annual Symposium on Foundations of Computer Science," pp. 307–311.

PIPPENGER, N. J. (1980), Comparative schematology and pebbling with auxiliary pushdowns (Preliminary version), *in* "12th Annual ACM Symp. on Theory of Computing," pp. 351–356.

RUSSELL, R. M. (1978), The CRAY-1 computer system, *Comm. ACM* **21**, 63–72.

RUZZO, W. L. (1980), Tree-size bounded alternation, *Comput. System. Sci.* **21**, 218–235.

RUZZO, W. L. (1981), On uniform circuit complexity, *J. Comput. System Sci.* **22**, 365–383.

SAVITCH, W., AND STIMSON, M. (1979), Time bounded random access machines with parallel processing, *J. Assoc. Comput. Mach.* **26**, 103–118.

SAVITCH, W. J. (1970), Relations between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4**, 177–192.

SAVITCH, W. J. (1979), Parallel and nondeterministic complexity classes, *in* "Proceedings, 5th Conf. Automata, Languages and Programming," Lecture Notes in Computer Science, pp. 411–424, Springer–Verlag, New York/Berlin.

SCHÖNHAGE, A. (1980), Storage modification machines, *SIAM J. Comput.* **9**, 490–508.

SEIFERAS, J. I. (1976), "A Note on Notions of Tape Constructibility," Tech. Report 187, Pennsylvania State University.

SCHNORR, C. P. (1976), The network complexity and the Turing machine complexity of finite functions, *Acta Inform.* **7**, 660–674.

WYLLIE, J. C. (1979), "The complexity of Parallel Computations," Ph.D. thesis and TR79-387, Dept. of Computer Science, Cornell University.