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].