7. Simultaneous Resource Bounds

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.