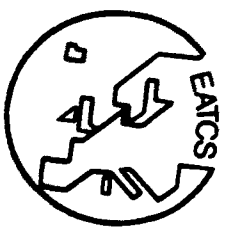
Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

226

Automata, Languages and Programming

13th International Colloquium
Rennes, France, July 15–19, 1986
Proceedings



Edited by Laurent Kott

Springer-Verlag
Berlin Heidelberg New York London Paris Tokyo

Editor

Laurent Kott
IRISA–INRIA Rennes
Campus de Beaulieu
35042 Rennes, France

CR Subject Classifications (1985): F.1, F.2, F.3, F.4, G.1.2, G.1.3, G.2.2

ISBN 3-540-16761-7 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-16761-7 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

PREFACE

ICALP is the annual European summer conference on Theoretical Computer Science sponsored by the European Association on Theoretical Computer Science (EATCS).

ICALP stands for International Colloquium on Automata, Languages and Programming, but this conference intends to cover all important areas of theoretical computer science such as automata theory, formal language theory, analysis of algorithms, computational complexity, computability theory, mathematical aspects of programming language definition, logic and semantics of programming languages, program specification, theory of data structures, theory of data bases, cryptology, VLSI structures.

Previous colloquia were held in Nafplion (1985), Antwerpen (1984), Barcelona (1983), Aarhus (1982), Haifa (1981), Amsterdam (1980), Graz (1979), Udine (1978), Turku (1977), Edinburgh (1976), Saarbrücken (1974) and Paris (1972).

ICALP 86 was organized by INRIA and located at INRIA Center of Rennes (called IRISA).

From a total of 140 submitted papers, 48 have been accepted by the Selection Committee, that consisted of W. BRAUER (Hamburg), J. DIAZ (Barcelona), Ph. FLAJOLET (Rocquencourt), J. GRUSKA (Bratislava), J. KARHUMAKI (Turku), L. KOTT (Rennes), K. MEHLHORN (Saarbrücken), U. MONTANARI (Pisa), M. NIVAT (Paris), D. PERRIN (Paris), G. PLOTKIN (Edinburgh), A. RESTIVO (Palermo), R. SETHI (Murray Hill), W. THOMAS (Aachen).

There were three invited lecturers : M.P. SCHUTZENBERGER (Paris) with "Les ambiguités de la Pureté", A. SCHÖNHAGE (Tübingen) with "Tape versus pointers : a study in implementing fast algorithms" and G. HANSEL (Rouen) with "Symbolics dynamics, automata theory and coding". These lectures are however not contained in this volume.

I am grateful for the support from Bull Company, City of Rennes and Regional Council of Brittany.

I also thank all those who made this conference possible, especially the members of the Program Committee who consisted of members of the selection committee plus L. GUIBAS (Palo Alto) and G. ROZENBERG (Leiden), the staff of INRIA (Rennes and Rocquencourt), especially M. GLEVEO, S. GOSSET and E. LEBRET.

Rennes, May 1986

Laurent KOTT
Program Committee Chairman

REFEREES FOR ICALP 86

AALBERSBERG IJ.J.
AMBOS-SPIES K.
ANCILOTTI P.
APT K.R.
AUSIELLO G.

BAIARDI F.
BALCAZAR J.L.
BEAUQUIER J.
BELLIA M.
BERSTEL J.
BERTONI A.
BONUCCELLI M.A.
BREBNER G.J.
BRODER A.
BUDACH L.

CARALLI A.L.
CARDELLI L.
CASAS R.
CERNY A.
COHN E.
COSTA G.
COURCELLE B.
CROCHEMORE M.
CULIK II K.

D'ARTRI R.
DEGANO P.
DELPORTE-GALLET C.
DE NICOLA R.
DEZANI-CIANCAGLINI M.
DIEKERT V.
DOBERKAT E.E.
DURIS P.

ENGELFRIET J.
EVEN S.

FEHR E.
FEIGENBAUM J.
FANTECHI A.
FONTET M.
FRAENKEL A.S.
FRIBOURG L.
FROUGNY Ch.
FRUTOS ESCRIG D.

GABARRO J.
GOLTZ U.

HAJEK P.
HARJU T.
HROMKOVIC J.

INDERMARK K.

JANSSENS D.
JERRUM M.
JOHNSON J.H.
JOSKO B.

KARLIN A.R.
KEESMAAT N.W.
KLEIJN H.C.M.
KLOP J.W.
KOREC I.
KROGER F.
KUCERA L.
KUICH W.

LAGARIAS J.C.
LANGE K.J.
LA PAUGH A.
LAVAULT Ch.
LAWRENCE J.
LAZARD D.
LEIGHTON T.
LENGAUER T.
LEVI G.
LIGHTNER M.
LUCCIO F.

MACQUEEN D.B.
MAHANEY S.
MAIN M.
MAIRSON H.
MANASSE M.S.
MAZURKIEWICZ A.
MILNER
MONIEN B.
MORGENSTERN J.
MUNRO I.

OBERSCHELP W.
OCHMANSKI E.
OLDEROG E.R.

PACHL J.
PALLOTTINO S.
PAPADIMITRIOU C.
PARTSCH H.
PATERSON M.
PERRIN G.R.
PIN J.E.
POLJAK J.
PUDLAK

RAMSHAN L.
REISIG W.
ROSENBERG A.L.
ROVAN B.
RUOHOVEN K.

SAKAROVITCH J.
SALOMAA A.
SALOMAA K.
SALTOR
SANNELLA D.
SAVAGE J.E.
SAVITCH W.
SCHAFFER A.A.
SCHOETT O.
SCHONING U.
SCHREIBER F.A.
SIEBE K.
SIFAKIS J.
SILVESTRI A.
STERN J.
STEYAERT
STIRLING C.
STROTHOTTE
SUBRAMANIAN A.
SZPANKOWSKI N.

TAUBNER D.
TENNENT R.D.

VALLEE B.
VALK R.
VAN EMDE BOAS P.
VAN LEEUWEN J.
VENTURINI ZILLI M.
VITTER J.
VOGLER W.
VOSSEN G.

WENEGER I.
WELZL E.
WIEDERMANN J.
WINKELMAN K.
WIRSING M.

YANNAKAKIS M.
YU S.

TABLE OF CONTENTS

CHARACTERIZATIONS OF PUNC AND PRECOMPUTATION

Extended Abstract

Eric W. Allender[†]
Department of Computer Science
Rutgers University
New Brunswick, N.J. 08903 (USA)

Abstract

Much complexity-theoretic work on parallelism has focused on the class NC, which is defined in terms of logspace-uniform circuits. Yet P-uniform circuit complexity is in some ways a more natural setting for studying feasible parallelism. In this paper, P-uniform NC (PUNC) is characterized in terms of space-bounded AuxPDA's and alternating Turing Machines with bounded access to the input. We also present a general-purpose parallel computer for PUNC; this characterization leads to an easy proof that NC = PUNC iff all tally languages in P are in NC. The characterizations of PUNC lead to natural methods for modelling precomputation. We show that for many classes of interest, there is a single "universal" table which can be used in place of any table of similar size and complexity, while for certain other classes, no such "universal" table exists.

1. Introduction

With the advent of VLSI, it has become feasible to construct computers which exhibit massive parallelism; chips with thousands of processors are no longer unimaginable. Motivated by the possibility of so much parallelism, complexity theory has picked up the question of determining what class of problems can be solved much more quickly in parallel than on sequential computers. Much complexity-theoretic work in this area has focused on the class NC, the class of all languages for which there exists a logspace-uniform family of circuits $\{C_n\}$ of size polynomial in n and of depth $\log^{O(1)}n$. (Definitions of uniformity, circuits, etc. will be given in a later section.)

Ruzzo [Ru-81] has shown that the class NC remains the same if stronger notions of uniformity than logspace-uniformity are used. That is, if we require not only that the functions $1^n \rightarrow C_n$ be logspace-computable, but that they must be "easily" logspace-computable, there is no effect on the class NC. Indeed, if we define NC not in terms of circuits, but rather in terms of interconnected processors (RAM's or finite-state machines) we can do away with the uniformity condition entirely, as the characterizations of NC in terms of HMM's [Co-81], SIMDAG's [Go-82], WRAM's [CSV-84], etc. [Vi-83] show. In this way, the "pre-processing" phase implicit in the logspace-uniformity condition for circuits is sidestepped. As has been observed before [Go-82, Co-81, DC-80], these machines can be thought of as building their own interconnection network during the course of the computation. In the characterization of NC in terms of HMM's, this is

---

[†]Portions of this research were carried out while the author was supported by NSF grant MCS 81-03608.

particularly evident. NC can thus be viewed as the class of problems for which fast "self-organizing" feasibly-parallel solutions exist. We argue that the "self-organizing" condition is an unnatural restriction.

Let us now take the view that it is okay to pack as much computational power into the pre-processing phase as is feasible. That is, we will be interested in any problem for which a fast circuit family $\{C_n\}$ exists, with the only stipulation being that the function $n \rightarrow C_n$ be feasible to compute. The natural formulation of this stipulation in complexity theory is P-uniformity [BCH-84]; the family of circuits $\{C_n\}$ is P-uniform if the function $n \rightarrow C_n$ is computable in time polynomial in n. This gives rise to the class P-uniform NC (PUNC), the class of languages for which there exists a P-uniform family of circuits $\{C_n\}$ of size polynomial in n and depth $\log^{O(1)}n$.

PUNC has not been studied before (although P-uniform circuits of depth $\log n$ and $\log^2 n$ were considered in [BCH-84,vzG-84]), and we list below some reasons which may partly explain why. At the same time, we present the contributions of this paper which, we believe, put PUNC on a more equal footing with NC.

First, NC has very nice characterizations in terms of general-purpose parallel computers such as SIMDAG's, WRAM's, etc. The fact that logspace-uniform circuit depth corresponds to parallel time on these machines has been taken as evidence that NC is the "right" setting in which to study parallelism. In fact, one might suppose that, because so much power has been placed in the pre-processing phase, problems in PUNC might be solvable only by "special-purpose" chips. However, in Section 5, we present a natural model of general-purpose computation on which PUNC is the class of problems solvable using $n^{O(1)}$ processors in $\log^{O(1)}n$ time.

Second, NC has many alternate characterizations in terms of other models of computation. For example, Ruzzo [Ru-81] has shown that NC can be characterized in terms of AuxPDA's and alternating Turing machines with simultaneous time and space bounds. It may have seemed unlikely that PUNC would have similar characterizations. Yet, in section 4, PUNC is characterized in terms of AuxPDA's and alternating Turing machines with simultaneous bounds on space and access to the input. For instance, a language L is in PUNC iff it is accepted by a logspace-bounded AuxPDA which moves its input head $2^{\log^{O(1)}n}$ times.

Third, many researchers seem to have considered uniformity conditions to be inelegant and ungainly. For example, Cook [Co-81], in discussing HMM's, cites as an advantage the fact that the HMM model has no uniformity condition, and Ruzzo [Ru-81] cites as undesirable the situation in which the circuit constructor is more powerful than the circuit. Uniformity conditions also cause some annoying difficulties when relating alternating time to circuit depth. Let NC^k denote the class of languages accepted by logspace-uniform circuits of depth $O(\log^k n)$. It was shown in [Ru-81] that NC^k = ASPACE,TIME($\log n$, $\log^k n$) for all $k \geq 2$. Unfortunately, equality does not seem to hold for $k = 1$. This is because deterministic logspace seems to be more powerful than alternating log time, and thus a logspace-computable function which constructs a circuit cannot be simulated in alternating log time. It was suggested in [Ru-81, Co-83] that uniformity conditions stronger than logspace-uniformity be used so the equality NC^k = ASPACE,TIME($\log n$, $\log^k n$) would hold for all k. Note that these results are obtained by essentially "overpowering" the precomputation phase by making the uniformity condition very strong. In this paper we take the

approach of "factoring out" the precomputation phase and dealing with it explicitly. Thus we show that $PUNC^k$ is the class of languages accepted by logspace-bounded alternating Turing machines which access their input only during the first $O(\log^k n)$ steps, for all $k \in \mathbf{N}$.

The type of computational power which is added by precomputation is illustrated in the following result:

$$NC = PUNC \Leftrightarrow \text{all tally languages in P are in NC.}$$

This, in turn, leads to results about exponential-time complexity classes.

The characterizations of PUNC also clarify the complexity of one-way auxiliary pushdown automata, which have been studied before in [Br-77a, Br-77b, Ch-77, WB-79, We-80, BDG-85, Hu-85]. Some restrictions of AuxPDA's have been shown not to have a severe effect on the complexity of the languages they accept; in [Ga-77] a two-way deterministic (one-head) PDA is presented which accepts a language which is hard for P under logspace reductions. Some other restrictions have been shown to be more limiting; in [Ki-81a, We-79] it was shown that logspace-bounded AuxPDA's whose pushdowns make at most a constant number of turns accept only languages in NLOG, and Ruzzo [Ru-81] showed that AuxPDA's which run in time $2^{\log^{O(1)} n}$ accept only languages in NC. Here we show that AuxPDA's which move their input heads at most $2^{\log^{O(1)} n}$ times accept exactly the languages in PUNC, and thus it seems unlikely that any such machine accepts a language which is hard for P.

In Section 6, we present some results about precomputation in general, rather than just in the context of circuit complexity. These results are intended to model the case in which one may be willing to devote considerable computational resources to a one-time task of building a precomputed table of some specified size, in order to have the table available for repeated use by more efficient routines. We show that for many cases of interest, there is a single "universal" table which can be used in place of all other tables of similar size and complexity, whereas for some other classes, no such "universal" table exists.

Due to space limitations, the results presented in this extended abstract are stated without proof. Proofs will be provided in the final version of the paper. Many of the results are also proved in [Al-85].

2. Preliminaries

A *circuit* for inputs of size n is a finite collection of AND, OR, and NOT gates, and n input nodes, along with an acyclic interconnection network linking the gatss to each other and to the input and output nodes. We will not distinguish between a circuit and its description in some suitable description language. The *size* of a circuit is the number of gates it contains. The *depth* of a circuit is the length of the longest path in the network from an input node to an output node.

A *family of circuits* is a set $\{C_n \mid n \in \mathbf{N}\}$ where C_n is a circuit for inputs of size n. $\{C_n\}$ is a *DSPACE(S(n))-uniform (DTIME(T(n))-uniform)* family of circuits if the function $n \rightarrow C_n$ is computable on a Turing machine in space $S(n)$ (time $T(n)$).

Background, and a more detailed discussion of circuit complexity, may be found in [Ru-81].

A language is *sparse* if $|\{w \in L \mid n \geq |w|\}|$ is $n^{O(1)}$; L is a *tally language* if $L \subseteq \{0\}^*$.

In order to use strings in $\{0.1\}^*$ to represent numbers and vice-versa, we use the standard method of letting the string w denote the number whose binary representation is $1w$. We shall often refer to languages $L \subseteq \{0, 1, \#\}^*$. This is merely a notational convenience; such an L should be thought of as a subset of $\{00,11,01\}^*$.

3. PUNC

The class of problems (languages) for which extremely fast parallel algorithms can be efficiently constructed is a class of some interest. PUNC is an attempt to capture this class in complexity-theoretic terms.

Although PUNC is defined as a class of languages, we may also say that a function f is in PUNC. It is clear what is meant by this.

PUNC is a robust class in the sense that it is not overly dependent upon idiosyncracies of the circuit model. In particular, if we allow circuits with unbounded fan-in, or if we consider P-uniform networks of RAM's or finite-state machines, the same class of languages results. Similarly, we could have defined PUNC in terms of aggregates [DC-80] or conglomerates [Go-82] with P-uniform interconnection networks.

Proposition 3.1:
L is in PUNC iff
L is accepted in time $\log^{O(1)}n$ by an aggregate with a P-uniform interconnection network iff
L is accepted in time $\log^{O(1)}n$ by a conglomerate with a P-uniform interconnection network.

PUNC is also closed under a broad class of reducibilities:

Proposition 3.2:
PUNC is closed under logspace reductions (defined in [Jo-75]) and NC_1 reductions (defined in [Co-83]).
If f is in PUNC, L is in PUNC, and for all w, $w \in L' \Leftrightarrow f(w) \in L$, then L' is in PUNC.

Thus PUNC has a robust definition in terms of circuit-based models of parallel computation. In the next two sections we will present characterizations of PUNC in terms of some sequential models of computation, and also in terms of general-purpose parallel computers.

4. An Alternate Characterization of PUNC

In order to consider alternating Turing machines of sublinear time complexity (which is necessary in order to characterize NC in terms of alternating time) a special "random-access" feature has to be contrived which allows alternating Turing machines to access specified bits of the input in unit time. This is a powerful feature, and it makes sense to restrict its use. In this section, we show that such a restriction in fact gives one way to characterize PUNC. Another characterization is given by restricting how often AuxPDA's may move their input heads.

Theorem 4.1: The following are equivalent:

(1) $L \in$ PUNC

(2) L is acceptsd by a logspace-bounded deterministic AuxPDA which moves its input head $O(2^{\log^{O(1)}n})$ times.

(3) L is accepted by a logspace-bounded nondeterministic AuxPDA which moves its input head $O(2^{\log^{O(1)}n})$ times.

It is clear that restricting motion of the input head is the natural way to restrict access to the input for an AuxPDA. It is far less clear what the correct way is to restrict an alternating Turing machine. Simply restricting the number of accesses to the input along any computation path is not sufficient, since *any* alternating Turing machine which uses at least logspace can be simulated efficiently by a machine which never accesses its input more than *once* on any computation path. The reasonable ways of restricting access to the input seem to be to restrict the computation which *precedes* any access to the input, and to restrict the total number of nodes in the alternation tree which access the input.

Theorem 4.2:
$L \in PUNC^k \Leftrightarrow$ L is accepted by a logspace-bounded alternating Turing machine which accesses its input only during the first $O(\log^k n)$ steps, for all $k \geq 1$.

It is interesting to compare Theorem 4.2 to the characterization of SC given by Sudborough in [Su-83]. Sudborough considered both one-way and two-way loglogspace-bounded alternating Turing machines, and he showed that SC is the class of all problems which are logspace-reducible to languages accepted by one-way loglogspace-bounded alternating Turing machines. Thus both SC and PUNC are characterized in terms of space-bounded altsrnating Turing machines with restricted access to the input.

We close this section with two further characterizations of PUNC, the proofs of which follow easily using the techniques used here and the results proved in [Ru-80] relating AuxPDA's and alternating Turing machines.

Theorem 4.3: The following are equivalent:
(1) $L \in PUNC$
(2) L is accepted by a logspace-bounded alternating Turing machine which accesses its input only during the first $O(\log^{O(1)}n)$ alternations.
(3) L is accepted by a logspace-bounded alternating Turing machine which, if it accepts an input, accepts via an alternation tree which contains $O(2^{\log^{O(1)}n})$ nodes which access the input.

5. A General-Purpose Parallel Computer for PUNC

NC is the class of problems using $n^{O(1)}$ processors in time $\log^{O(1)}n$ on a SIMDAG, on a WRAM, and on almost all of the many approximately-equivalent models of parallel computation which have been proposed in the past few years. (For surveys, see [Co-81,Vi-83].) This has been taken as evidence that NC truly captures the notion of efficient parallel computation. However, it was argued earlier in the paper that PUNC, and not NC, better captures that notion, at least when one is trying to model the class of problems solvable efficiently by special-purpose chips.

Note that SIMDAG's, WRAM's, and the like are *general-purpose* parallel computsrs, in the sense that one SIMDAG will efficiently compute all problems in NC; all one need do is swap one program out of the CPU and swap another one in. One might therefore suspect that NC models general-purpose parallel computation, and PUNC models special-purpose parallel computation. We show here that that is not the case.

Models of general-purpose parallel computers such as SIMDAG's and WRAM's all share the characteristic that they have infinitely many processors. These processors are nearly identical, but each processor has a register which stores its "name", or address. Let us now take the position that it makes just as much sense to provide infinitely many bits, *where the bits are no harder to produce than are the processors*; i.e., the n-th bit can be produced in time polynomial in n. This section explores the consequences of taking that position.

A *SIMDAG augmented with a sequence* s consists of a SIMDAG along with an infinite sequence of read-only global registers B_1, B_2, ..., where each B_i contains the i-th bit of the sequence. When counting the number of processors used by a SIMDAG during a computation, we include the number of registers B_i accessed during the computation. (Equivalently, we could let B_i reside in processor P_i.)

A sequence s is *P-printable* if the language $\{0^n \mid$ the n-th bit of s is 1$\}$ is in P; that is, if the n-th bit of s can be obtained in time polynomial in n.

The reader should already suspect that L is in PUNC iff L is accepted using $n^{O(1)}$ processors in time $\log^{O(1)}n$ on a SIMDAG augmented with a P-printable sequence. However, we will prove more. There is a universal P-printable sequence s_U such that L is in PUNC iff L is accepted using $n^{O(1)}$ processors in time $\log^{O(1)}n$ on a SIMDAG augmentsd with a s_U. Furthermore, s_U has a natural and appealing definition.

For any language L, the *characteristic sequence for L*, s_L, is an infinite sequence of 0's and 1's such that the r-th character of s_L is 1 iff $r \in L$.

Now let U be any language complete for EXPTIME under log-lin reductions. (See, e.g., [St-74].) For a concrete example, let $U = \{M\#w\#^{l|w|} \mid M$ accepts w in time $2^{(l+1)|w|}\}$. The characteristic sequence s_U is P-printable; to obtain the n-th bit, see if $n \in U$ in time $2^{O(|n|)} = 2^{O(\log n)} = n^{O(1)}$.

Theorem 5.1:
$L \in$ PUNC $\Leftrightarrow$ L is accepted using $n^{O(1)}$ processors in time $\log^{O(1)}n$ time on a SIMDAG augmentsd with s_U.

Theorem 5.1 says that there is a single general-purpose parallel computer on which all problems in PUNC can be solved quickly; thus this has somewhat the same flavor as the results presented in [GP-83, Vi-84], in which efficient general-purpose parallel computers are studied. Unfortunately, the practical utility of this result is limited since, as Rackoff has pointed out [Ra-85], the number of registers B_i needed is exponential in the size of the program M.

Theorem 5.1, together with the relationship between P-printable sequences and tally languages, leads to an easy proof of the following result, which helps to explain the nature of the computational power added by P-Uniformity over logspace-uniformity.

Theorem 5.2: [Al-86]
NC = PUNC $\Leftrightarrow$ all tally languages in P are in NC $\Leftrightarrow$ $\{0^i \mid i \in U\}$ is in NC.

Theorems 5.1 and 5.2 make obvious the connection between P-uniform circuits and characteristic sequences of languages in EXPTIME. Other closely-related concepts are P-recognizable real numbers (using the "standard left cut" definition in [Ko-83]) and P-printable sets (sets S such that the function $n \rightarrow S \cap \{w \mid n \geq |w|\}$ is computable in time polynomial in n [HY-84]). Also, P-

printable sequences may be defined as sequences s such that, for some k and M_v, every prefix of s is in $K_v(\log\ n,\ n^k)$, where $K_v(\log\ n,\ n^k)$, is the "generalized Kolmogorov complexity" measure of [Ha-83]. In addition, a set is P-printable iff it is sparse and has an easy ranking function, as considered in [GS-85]. For a related discussion, see [Ko-84,Al-86].

Tally languages have often been studied in conjunction with other types of sparse sets. Theorem 5.2 led to some other results about the complexity of sparse sets in P, which are covered in [Al-85,Al-86]

Techniques presented in [Bo-74] enable us to interpret results about classes of tally languages as results about exponential-time complexity classes. Thus Theorem 5.2 has the following corollary:

Corollary 5.3: NC = PUNC iff ASPACE,TIME(n, $n^{O(1)}$) = EXPTIME.

Since NC = ASPACE,TIME(log n, $\log^{O(1)}n$), Corollary 5.3 can be interpreted as saying that the NC = PUNC question is equivalent to the exponential-time analog of the NC = P question.

6. Precomputation

The results of Section 5 show how to augment a parallel computer with a precomputed table, where the table is feasible to compute and of polynomial size. There are other aspects of precomputation, however. For instance, when augmenting a space-bounded computation, it makes sense to bound the size of the precomputed table as well. Furthermore, it may happen that one is willing to use more than polynomial time to precompute a table. This section deals with modelling these aspects of precomputation.

Of particular interest will be the existence of "universal" precomputed tables, analogous to the "universal" sequences of section 5. Universal tables exist in many interesting cases, but do not exist for all classes.

Computations augmented by short strings of "nonuniform" data were studied in [KL-82], as a generalization of nonuniform circuit complexity. The principal difference between the definitions of [KL-82] and the definitions considered here is that the "advice" bits of [KL-82] did not need to be effectively or efficiently constructible. Here, we are trying to model "feasible" precomputation. Note that there is no chance of having "universal" tables for nonuniform advice, since given any sequence of advice bits, (1) any language accepted relative to that sequence is r.e. in that sequence, and (2) there are languages which are not r.e. in the sequence which can be recognized efficiently with very little advice.

Definition: Let C be a class of languages, and let S and T be sets of functions. (S and T should be thought of as bounds on the size of the precomputed table and on the amount of time spent in precomputation, respectively.) Then C/S,pretime(T) is the class of languages L such that there is a function h (a *help function* for C/S,pretime(T)) such that $n \rightarrow h(n)$ is computable in time bounded by some function in T, where $|h(n)|$ is bounded by some function in S, and there is some language $L' \in C$ such that for all strings w of length $\leq n$, $w \in L$ iff $w\#h(n) \in L'$. (We will denote this language L' by $L/h(n)$.

For example, SC/O(log n),pretime($n^{O(1)}$) is the class of all languages which can be accepted in polynomial time and polylog space with the help of a precomputed table of size O(log$n$), where

the table can be built in polynomial time. For another example, note that Adleman showed in [Ad-78] that R $\subseteq$ P/$n^{O(1)}$,pretime($2^{n^{O(1)}}$).

We could also make similar definitions for the case when there is a space bound on the precomputation. Thus, for example, we also have that R $\subseteq$ P/$n^{O(1)}$,prespace($n^{O(1)}$) [Ad-78].

Definition: A help function h is *universal* for C/S,pretime(T) if for every language L in C/S,pretime(T) there is a function g and some language L′ $\in$ C such that the function $n \rightarrow h(g(n))$ is a help function for C/S,pretime(T), and for all strings w of length $\leq g^{-1}(n)$, $w \in$ L iff $w\#h(n) \in$ L′.

In other words, the help function h is universal for C/S,pretime(T) if it can be used in place of all other help functions.

The main lemma which allows us to show a relationship between precomputation and characteristic sequences, and eventually to produce universal help functions, is the following.

Lemma 6.1: Assume DLOG $\subseteq$ C, $n^{O(1)} \subseteq$ T, and all functions in S are monotone and are computable in logspace. Define $s^{-1}(r) = \min\{n \mid s(n) = r\}$. If for all functions $s \in$ S and $t \in$ T, the function $t(s^{-1}(s(n)+1)) \in$ T, then for every help function h for C/S,pretime(T) there is a help function h' for C/S,pretime(T) such that if L/$h(n) \in$ C, then L/$h'(n) \in$ C, and for all $r < n$, $h'(r)$ is a prefix of $h'(n)$.

Note that the help function h' defines a sequence; this sequence can, in turn, be interpreted as the characteristic sequence of a language. For appropriate classes C, S, and T, the characteristic functions of complete languages give rise to universal help functions. Rather than present necessary conditions on C, S, and T for this to happen, it is more enlightening to consider some examples.

In what follows, assume an encoding of Turing machines such that no encoding contains "00" as a substring.

Let $L_1 = \{M\#w\#^{l|w|} \mid M \text{ accepts } w \text{ in time } 2^{(l+1)|w|}\}$.
Let $L_2 = \{M\#w\#^{l|w|} \mid M \text{ accepts } w \text{ in time } 2^{2^{(l+1)|w|}}\}$.
Let $L_3 = \{M(00)^s1w \mid M \text{ accepts } w \text{ in time } 2^{2^{s+|w|}}\}$.
Let $L_4 = \{M(00)^s1w \mid M \text{ accepts } w \text{ in time } 2^{2^{2^{s+|w|}}}\}$.

Theorem 6.2:
Let S, T, and h be given by any row in the following table. then h is a universal help function for C/S,pretime(T), where C is any of the classes DLOG, NLOG, SC, NC, PUNC, P.

S	T	h
$n^{O(1)}$	$n^{O(1)}$	$n \rightarrow s_1,s_2,\ldots,s_n$, where s is the characteristic sequence for L_1.
$n^{O(1)}$	$2^{n^{O(1)}}$	$n \rightarrow s_1,s_2,\ldots,s_n$, where s is the characteristic sequence for L_2.
O(log n)	$n^{O(1)}$	$n \rightarrow s_1,s_2,\ldots,s_{\log n}$, where s is the characteristic sequence for L_3.
O(log n)	$2^{n^{O(1)}}$	$n \rightarrow s_1,s_2,\ldots,s_{\log n}$, where s is the characteristic sequence for L_4.
O(loglog n)	$2^{\log^{O(1)}n}$	$n \rightarrow s_1,s_2,\ldots,s_{\text{loglog } n}$, where s is the characteristic sequence for L_4.

In some cases, it can be shown that universal help functions do not exist.

Theorem 6.3:
There is no universal help function for P/O(loglog n),pretime($2^{n^{O(1)}}$).

There are other cases where the existence of universal help functions is unlikely, but cannot

be proved without proving DLOG $\neq$ P. In these cases, the existence of universal help functions is usually equivalent to the collapse of some higher space and time complexity classes.

Theorem 6.4: The following are equivalent:

(1) There is a universal help function for DLOG/O(loglog n),pretime($n^{O(1)}$)

(2) DLOG/O(loglog n),pretime($n^{O(1)}$) $=$ DLOG

(3) DSPACE($(2^{2^n})^{O(1)}$) $=$ DTIME($(2^{2^{2^n}})^{O(1)}$)

7. Open Problems The results of Sections 5 and 6 show how to replace any precomputed table with a "universal" table of similar asymptotic size and complexity. Unfortunately, the universal table will be larger by a constant factor, where that constant factor has size exponential in the size of the program which performs the precomputation. Can these results be improved so the constant factor is not so overwhelming?

Are there any "natural" problems which seem to be in PUNC - NC?

Acknowledgements

Larry Ruzzo, in considering "U_B-uniformity" [Ru-81], has recently proved some results with a flavor similar to Theorems 5.2 and 6.4 [Ru-85]. Discussions with a number of people influenced the direction in which this work developed: my thesis advisor: Kim King, Marc Graham, Rich DeMillo, Craig Tovey, Ray Miller, Steve Cook, Charles Rackoff, Jim Hoover, Pat Dymond, Larry Ruzzo, and Steve Mahaney.

References

[Ad-78] L. Adleman, *Two theorems on random polynomial time*, Proc. 19th IEEE Symposium on Foundations of Computer Science, pp. 307-311.

[Al-85] E. W. Allender, *Invertible functions*, Doctoral Dissertation, Georgia Institute of Technology.

[Al-86] E. W. Allender, *The complexity of sparse sets in P*, Paper presented at the Structure in Complexity Theory Conference, Berkeley, to appear in Lecture Notes in Computer Science.

[BDG-85] J. L. Balcázar, J. Diaz, J. Gabarró, *On some "non-uniform" complexity measures*, 5th Conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 199, pp. 18-27.

[BCH-84] P. W. Beame, S. A. Cook, and H. J. Hoover, *Log depth circuits for division and related problems*, Proc. 25th IEEE Symposium on Foundations of Computer Science, pp. 1-11.

[Bo-74] R. V. Book, *Tally languages and complexity classes*, Information and Control 26, 186-193.

[Br-77a] F.-J. Brandenburg, *On one-way auxiliary pushdown automata*, Proc. 3rd GI Conference, Lecture Notes in Computer Science 48, pp. 133-144.

[Br-77b] F.-J. Brandenburg, *The contextsensitivity of contextsensitive grammars and languages*, Proc. 4th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 52, pp. 272-281.

[CSV-84] A. K. Chandra, L. J. Stockmeyer, U. Vishkin, *Constant depth reducibility*, SIAM J. Comput. 13, 423-439.

[Ch-77] M. P. Chytil, *Comparison of the active visiting and the crossing complexities*, Proc. 6th Conference on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 53, pp. 272-281.

[Co-81] S. A. Cook, *Towards a complexity theory of synchronous parallel computation*, L'Enseignement Mathematique 27, 99-124.

[Co-83] S. A. Cook, *The classification of problems which have fast parallel algorithms.* Proc. 4th International Conference on Foundations of Computation Theory, Lecture Notes in Computer Science 158, pp. 78-93.

[DC-80] P. W. Dymond and S. A. Cook, *Hardware complexity and parallel computation*, Proc. 20th IEEE Symposium on Foundations of Computer Science, pp. 360-372.

[vzG-84] J. von zur Gathen, *Parallel powering*, Proc. 25th Annual ACM Symposium on Theory of Computing, pp. 31-36.

[Ga-77] Z. Galil, *Some open problems in the theory of computation as questions about two-way deterministic pushdown automaton languages*, Mathematical Systems Theory 10, 211-228.

[GP-83] Z. Galil and W. Paul, *An efficient general-purpose parallel computer*, J. ACM 30, 360-387.

[GS-85] A. V. Goldberg and M. Sipser, *Compression and ranking*, Proc. 17th Annual ACM Symposium on Theory of Computing, pp. 440-448.

[Go-82] L. M. Goldschlager, *A universal interconnection pattern for parallel computers*, J. ACM 29, 1073-1086.

[Ha-83] J. Hartmanis, *Generalized Kolmogorov complexity and the structure of feasible computations*, Proc. 24th IEEE Symposium on Foundations of Computer Science, pp. 439-445.

[HY-84] J. Hartmanis and Y. Yesha, *Computation times of NP sets of different densities*, Theoretical Computer Science 34, 17-32.

[Hu-85] D. T. Huynh, *Non-uniform complexity and the randomness of certain complete languages*, Technical Report TR 85-34, Computer Science Department, Iowa State University.

[Jo-75] N. D. Jones, *Space-bounded reducibility among combinatorial problems*, J. Computer and System Sciences 11, 68-85.

[KL-82] R. M. Karp and R. J. Lipton, *Turing machines that take advice*, L'Enseignement Mathematique 28, 191-209.

[Ki-81a] K. N. King, *Measures of parallelism in alternating computation trees*, Proc. 13th Annual ACM Symposium on Theory of Computing, pp. 189-201.

[Ko-83] K.-I. Ko, *On the definition of some complexity classes of real numbers*, Mathematical Systems Theory 16, 95-109.

[Ko-84] K.-I. Ko, *A definition of infinite pseudorandom sequences*, manuscript, University of Houston.

[Pi-81] N. Pippenger, *Pebbling with an auxiliary pushdown*, J. Computer and System Sciences 23, 151-165.

[Ra-85] C. Rackoff, personal communication.

[Ru-81] W. L. Ruzzo, *On uniform circuit complexity*, J. Computer and System Sciences 21, 365-383.

[Ru-85] W. L. Ruzzo, personal communication.

[St-74] L. J. Stockmeyer, *The complexity of decision problems in automata theory and logic*, Doctoral Dissertation, M.I.T.

[Su-83] I. H. Sudborough, *Bandwidth constraints on problems complete for polynomial time*, Theoretical Computer Science 26, 25-52.

[Vi-83] U. Vishkin, *Synchronous parallel computation - a survey*, preprint, Courant Institute, New York University.

[Vi-84] U. Vishkin, *A parallel-design distributed-implementation (PDDI* general-purpose computer), Theoretical Computer Science 32, 157-172.

[We-79] G. Wechsung, *The oscillation complexity and a hierarchy of context-free languages*, Proc. 2nd Proc. 2nd International Conference on Fundementals of Computation Theory, Akademie-Verlag, Berlin, GDR, pp. 508-515.

[We-80] G. Wechsung, *A note on the return complexity*, Elektronische Informationsverarbeitung und Kybernetik 16, 139-146.

[WB-79] G. Wechsung and A. Brandstadt, *A relation between space, return and dual return complexities*, Theoretical Computer Science 9, 127-140.

COMPARISON OF ALGORITHMS CONTROLLING CONCURRENT ACCESS TO A DATABASE : A COMBINATORIAL APPROACH

D. ARQUES, J. FRANÇON, M.T. GUICHET, P. GUICHET
I.S.E.A. Université de Haute Alsace
4, rue des Frères Lumière 68093 MULHOUSE, FRANCE

Abstract In this paper we compare the performances of concurrency control algorithms using the combinatorics of words. We characterize such control algorithms by a triple of values, respectively estimating the frequency of accepted executions of concurrent transactions, the degree of authorized parallelism, and the load factor (a value related to the average number of elementary operations needed for the serialization of an execution). We compute these values for timestamp ordering and two phases locking in the case of the concurrent execution of two transactions. We obtain the exact and asymptotic frequences of deadlocked executions of two transactions, and compare these two algorithms in various asymptotic situations.

I. INTRODUCTION

A database is a set of objects called entities whose values must at any time verify some relations characteristic of the situation described by the database. These relations are called the integrity constraints of the database.

A transaction is any sequence of atomic actions operating on the entities, whose global effect preserves these integrity constraints.

In practice several transactions can access concurrently to the same database. Such a concurrent execution can lead to a new database state distinct from the one given by a serial execution, without concurrency of the same transactions (cf [1], [8], [16]).

The aim of an algorithm controlling access to a database - or controller - is to rearrange the sequence of access to the database entities coming from the transactions, in such a way that the resulting execution is equivalent (in some sense) to a serial execution.

An important problem, both in theory and practice, is to compare the performances of these algorithms. This has been done in two strongly different ways. The system composed of the controller, the transactions and the entities is modelized in the first approach by a queueing network, in the second by a shuffle of words. The

first approach leads to quantitative results (cf [5], [9], [10], [14], [15]), while the second gives a qualitative measure of the quantity of parallelism (cf [11], [12], [13], [4]).

In this paper we make use of the theoretic tools of the combinatorics of words, and show how to characterize a controller by a triple of values:

The first is a measure of the ratio of executions not rejected by the algorithm. The second, called parallelism ratio is a quantitative analog of the parallelism measure by set inclusion introduced by Papadimitriou. The last is a measure of the average number of modifications made by the concurrency control algorithm to make correct a concurrent execution, it has not rejected. In other words it is a measure of the "quantity of control" to apply, and can be seen as the average delay imposed to an arbitrary execution.

This quantitative approach of qualitative notions is studied here for the timestamp ordering, and two phases locking concurrency control algorithms with concurrent execution of two transactions. The three above described parameters are explicitely computed, and we can infer from theorem 4 and proposition 2 the exact value enumerating the deadlocked executions of two transactions.

We can then quantitatively compare timestamp ordering and two phases locking in various situations. For example in the case of the execution of two transactions of total length far bigger than the database size, we show a complete opposition of the behaviour of these two algorithms (cf section IV, 4).

II. DEFINITIONS

II.1. Execution of a set of concurrent transactions

The entities constituting the database, are the atomic objects accessed by the transactions. In the following they will be represented by the integers $1,\ldots,r$. The number r is said to be the size of the database.

A transaction T is an activation date followed by a finite chronological sequence of access to these entities. We do not distinguish between the type of these access : reading or writing.

Let $\mathcal{C}_k$ be a set of k transactions enumerated $T^{(1)},\ldots,T^{(k)}$ in the order of their respective activations. The order of these activations is supposed to be independant of the order given by the first access of each transaction (for example someone can connect with the database, and then stays idle for a while).

The access of the transaction $T^{(\ell)}$ to the entity j is noted by the

letter $t_j^{(\ell)}$. In a natural way we associate to a transaction $T^{(\ell)}$ the word $w(T^{(\ell)}) = t_{i_1}^{(\ell)} t_{i_2}^{(\ell)} \ldots t_{i_m}^{(\ell)}$, coding the finite sequence of successive access to the entities $i_1, i_2, \ldots, i_m$ by the transaction $T^{(\ell)}$.

Given a set $\mathcal{C}_k = \{T^{(1)}, \ldots, T^{(k)}\}$ of k transactions we call execution of $\mathcal{C}_k$ every word e belonging to the set $w(T^{(1)}) \sqcup\!\sqcup w(T^{(2)}) \sqcup\!\sqcup \ldots \sqcup\!\sqcup w(T^{(k)})$, of the shuffle products of the words $w(T^{(1)})$, $w(T^{(2)})$, ..., $w(T^{(k)})$.

The length of an execution is its length as a word.

We note $E_{r,k}$ the set of the executions associated with the set of all families of k transactions acting on a size r database.

Two transactions are said to be in conflict, if they access the same entity during their execution.

II.2. Serializability

As remarked in the introduction, an arbitratry execution of the set $\mathcal{C}_k$ of k transactions does not necessarily lead to a consistent state of the database, (cf [8], [12], [13], [16]).
A particular example of a consistency preserving execution of $\mathcal{C}_k$ is the serial execution s given by the word
$w(T^{(\sigma(1))}) \cdot w(T^{(\sigma(2))}) \cdot \ldots \cdot w(T^{(\sigma(k))})$, where σ is a permutation of $\{1,\ldots,k\}$.
Taking into account the possibility of concurrent access to the entities leads to introduce the notion of serializable (or correct) execution :

An execution is called serializable if it is equivalent to a serial one in the following sense. Two executions e_1 and e_2 are equivalent if and only if the projections of these two words on $T^{(i)}$, for $1 \leqslant i \leqslant k$, and on the entity j, for $1 \leqslant j \leqslant r$, are identical (cf [1]). Intuitively, two executions are equivalent if and only if they preserve for each transaction the order of its access to entities, and if they imply the same order of access of the transactions to each entity. (cf [3], [4], [6], [16] for other definitions).

We note $S_{r,k}$ the subset of $E_{r,k}$ consisting of the serializable executions.

II.3. Schedulers

A concurrency control algorithm, called scheduler, or controller, in this paper, is an algorithm, which as input receives an execution e of $E_{r,k}$; and gives back as output, when possible, and after reordering a serializable execution e'.

To such a scheduler $\mathcal{C}$ we associate the language $C_{r,k,\mathcal{C}}$, subset of $S_{r,k}$, consisting of the serializable executions it can output.