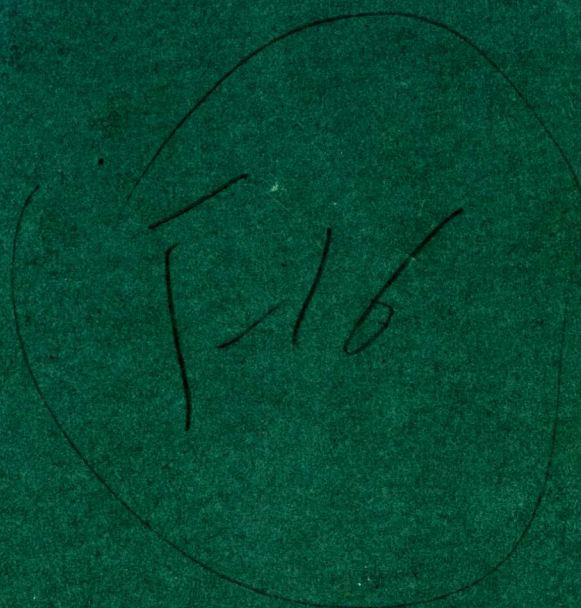


**SOFTWARE FOR COMPUTER  
CONTROL 1982**

---

Edited by  
**G. FERRATE and E. A. PUENTE**





# SOFTWARE FOR COMPUTER CONTROL 1982

*Proceedings of the Third IFAC/IFIP Symposium  
Madrid, Spain, 5-8 October 1982*

Edited by

**G. FERRATE**

and

**E. A. PUENTE**

*E.T.S. Ingenieros Industriales, Madrid, Spain*

Published for the

**INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL**

by

**PERGAMON PRESS**

**OXFORD · NEW YORK · TORONTO · SYDNEY · PARIS · FRANKFURT**

# THIRD IFAC/IFIP SYMPOSIUM ON SOFTWARE FOR COMPUTER CONTROL

## *Sponsored by*

IFAC Technical Committee on Applications  
IFAC Technical Committee on Computers  
IFAC Technical Committee on Education

## *Co-Sponsored by*

IFIP Technical Committee on Computer Applications in Technology

## *Organized by*

Comité Español de la IFAC  
Departamento de Ingeniería de Sistemas y Automática de la  
Universidad Politécnica de Madrid

## *International Program Committee (IPC)*

G. Ferraté, Spain (Chairman)  
P. Albertos, Spain  
A. Buzo, Mexico  
C. M. Doolittle, U.S.A.  
P. Elzer, F.R.G.  
D. G. Fisher, Canada  
J. Gertler, Hungary  
R. Isermann, F.R.G.  
P. M. Larsen, Denmark  
M. Mansour, Switzerland  
J. S. Meditch, U.S.A.  
R. Mezencev, France  
M. Novak, Czechoslovakia  
E. A. Puente, Spain  
V. Strejc, Czechoslovakia  
B. Tamm, U.S.S.R.  
E. A. Trakhtengerts, U.S.S.R.  
J. D. N. Van Wyk, South Africa  
T. J. Williams, U.S.A.

## *National Organizing Committee (NOC)*

E. A. Puente (Chairman)  
M. Alique  
R. Aracil  
A. J. Avello  
L. Basañez  
E. Bautista  
J. G. Bernaldo de Quirós  
E. F. Camacho  
M. Collado  
J. M. Coronado  
R. Huber  
J. A. Martín-Pereda  
M. Mellado  
P. de Miguel  
J. A. de la Puente  
R. Puigjaner  
A. Rodríguez

## PREFACE

This volume contains the papers presented to the Third IFAC/IFIP Symposium in the series "Software for Computer Control" (Madrid, 5-8 October, 1982), which follows the symposia held in Tallin (U.S.S.R.) in 1976 and in Prague (Czechoslovakia) in 1979.

The aim of this Symposium is to present, discuss and summarize the present state of software developments for digital computer applications in science and control. Special emphasis has been given to application of software developments, where relevant.

A total of 73 papers were given in 24 technical sessions, covering the following topics:

- Real-time languages and operating systems
- Man-machine communication software
- Software for robots
- Software for distributed control systems
- C.A.D. of digital computer control systems
- Adaptive computer control systems
- Algorithms for digital computer control
- Control software engineering and management
- Industrial applications

In addition to these papers, four plenary papers presented by invited speakers, cover the theory, methods and applications of digital computer programming for control applications.

We would like to thank the members of the International Program Committee for their effort in the selection of papers and the members of the National Organizing Committee for their support in the organization.

We hope that the publication of these papers, which come from specialists of 26 different countries, will make a good contribution to the development of this important field.

October 1982

G. Ferrate  
E.A. Puente  
Editors

# CONTENTS

## PLENARY SESSION

Specifics of Real-Time Software Design for Multiprocessor Computer Systems <i>E.A. Trakhtengerts</i>	1
A Unified Presentation of Model Reference Adaptive Controllers and Stochastic Self Tuning Regulators (Concepts, Algorithms, Applications) <i>I.D. Landau</i>	11
On the Development and Implementation of Parameter-Adaptive Controllers <i>R. Isermann and K.H. Lachmann</i>	25
Interprocess Communication Primitives for Distributed Process Control <i>I.M. MacLeod and M.G. Rodd</i>	51

## REAL-TIME LANGUAGES AND OPERATING SYSTEMS I

MODEB: A Real-Time Operating System Kernel Written in the High Level Programming Language Modula-2 <i>G. Kaier</i>	61
Real-Time Operating System Software for a Multiprocessor System <i>M. Levin and A.V. Spiegelhauer</i>	69

## REAL-TIME LANGUAGES AND OPERATING SYSTEMS II

Implementation of Real-Time Facilities in Pascal <i>H. Elmquist and S.E. Mattsson</i>	77
Programmable Logic Controllers and Petri Nets: A Comparative Study <i>M. Silva and S. Velilla</i>	83
Optimization of Computations on Multiprocessor Systems <i>E.A. Trakhtengerts and Yu.M. Shuraits</i>	89

## MAN-MACHINE COMMUNICATIONS SOFTWARE

A Graphical Approach to Documentation and Implementation of Control Systems <i>H. Elmquist</i>	95
Digital System for Voice Switching Loudspeaking Telephones <i>J. Rivero-Laguna</i>	101

Interactive System to Help Program Design <i>J.M. Faba</i>	107
SOFTWARE FOR ROBOTS	
Automatic Planning for Robots: Review of Methods and Some Ideas about Structure and Learning <i>J. Cuenca and C. Salmeron</i>	113
Trajectory Calculation of a Robot which Manipulates Moving Objects <i>C. Balaguer</i>	121
The Sweep Mapping in Robot Vision: Properties and Computational Savings <i>L. Basañez and C. Torras</i>	127
SOFTWARE FOR DISTRIBUTED CONTROL SYSTEMS I	
Control Hierarchy in a Distributed Process Control System <i>M.J. Shah and F. Schneider</i>	133
Decomposition on the Basis of Structure of Discrete Systems. Application to Hierarchical Approach of a Water Quality Control Problem <i>J.L. Calvet and A. Mano</i>	139
Software and Protocols in REBUS. A Distributed Real-Time Control System <i>J.M. Ayache, J.P. Courtiat, M. Diaz and J. Michelena</i>	147
SOFTWARE FOR DISTRIBUTED CONTROL SYSTEMS II	
A Set of Tools for Designing and Evaluating Communication Protocols in Industrial Computer Networks <i>L. Motus and J. Vain</i>	155
Evaluation of a Transport Protocol for its Implementation in Distributed Multimicrocomputer Control Systems <i>J. Figueras and A. Alabau</i>	165
SOFTWARE FOR DISTRIBUTED CONTROL SYSTEMS III	
Synchronization of Concurrent Activities in a Local Area Computer Network <i>H. Rzehak</i>	171
CL80: A High Level Programming Language for Distributed Control Systems <i>R. Galan and A.J. Avello</i>	179
COMPUTER AIDED DESIGN OF DIGITAL COMPUTER CONTROL SYSTEMS I	
CATPAC- An Interactive Software Package for Process Analysis and Control <i>M. Barthelmes, P. Bressler, D. Blinz, K. Gutschow, J. Heeger, R. Kersic and U. Schreiber</i>	183
ICONGRAPH- Programpackage for Interactive Controller Design by Graphical Plotting <i>R. Tuschak, R. Bars, R. Haber, M. Habermayer, T. Kovacs, I. Vajk, M. Vajta and L. Keviczky</i>	191

An Interactive Simulation System - A New Component of the Development Support System EPOS <i>E. Joho</i>	197
COMPUTER AIDED DESIGN OF DIGITAL COMPUTER CONTROL SYSTEMS II	
An Algorithm for Design of Low Order Dynamic Compensators for Large Scale Systems <i>M. Rakic and Dj. Petkovski</i>	203
A Package for Computer Aided Design of Multivariable Discrete Control Systems <i>V. Feliu and A.J. Avello</i>	209
Multivariable Control Scheme Design for Discrete Systems Employing Multirate Sampling <i>F.M. Hughes and B.O. Awobamise</i>	217
A Software Package for Computer Aided Design of Multivariable Control Systems <i>P. Gonzalez de Santos, J. No, M. Armada and J.A. Cordero</i>	223
COMPUTER AIDED DESIGN OF DIGITAL COMPUTER CONTROL SYSTEMS III	
A Package for Multivariable Adaptive Control <i>G. Bartolini, G. Casalino, F. Davoli and R. Minciardi</i>	229
An Overview of the Computer Aided Design of Control Systems <i>T.P. Wang</i>	237
COMPUTER AIDED DESIGN OF DIGITAL COMPUTER CONTROL SYSTEMS IV	
A Package for Computer Design of Concurrent Logic Control Systems <i>J. Martinez and M. Silva</i>	243
Computer Aided Generation of Microprocessor Software for Controlling Static Power Converters <i>F. Aldana, J. Peire, C.M. Penalver and J. Uceda</i>	249
OVIDE: A Software Package for Verifying and Validating Petri Nets <i>E. Le Mer</i>	255
ADAPTIVE COMPUTER CONTROL SYSTEMS I	
On PID Self-Tuners <i>R. Ortega</i>	261
Multivariable Adaptive Control - Comparative Studies of Some Algorithms <i>J.M. Dion, L. Dugard and I.D. Landau</i>	267
Organized Learning Models (Pursuer Control Optimisation) <i>R. Pla</i>	273
ADAPTIVE COMPUTER CONTROL SYSTEMS II	
Quasi-Direct Adaptive Control for Non-Minimum Phase Systems <i>R. Lozano L. and I.D. Landau</i>	279

Adaptive Control of Non-Minimum Phase Systems with Independent Tracking and Regulation Specifications <i>R. Lozano L.</i>	285
A Generalised Minimum Variance Self-Tuning Controller Incorporating Pole Assignment Specification <i>A. Y. Allidina and F.M. Hughes</i>	289
Self-Tuning Controller for Plants Distributed by Nonstationary Stochastic Disturbance <i>M.A. Magdy, J. Hrusak and M. Simandl</i>	295
ALGORITHMS FOR DIGITAL COMPUTER CONTROL I	
Process Parameter Identification: A Total Least Square Approach <i>M.F. Senning</i>	301
Recursive Algorithm for Identification of Dynamic Processes <i>J. Quevedo and J. Riera</i>	307
ALGORITHMS FOR DIGITAL COMPUTER CONTROL II	
On-Line Use of a Linear Programming Controller <i>P.O. Gutman</i>	313
Algorithms for Time Continuous Quadratic Performance Functions in Sampled-Data Control Systems <i>B. Lennartson, B. Qvarnstrom and S. Zeng-Qi</i>	319
ALGORITHMS FOR DIGITAL COMPUTER CONTROL III	
On a Procedure for the Calculation of Sensitivity Functions for Optimal Control Systems <i>D.M. Marganovic and D.P. Petrovacki</i>	325
Algorithms for Interactive Multicriteria Optimal Design of Control Systems <i>A. Ollero and R. Marin</i>	331
Algorithms of Optimization of Linear and Non-Linear Systems <i>R. Gabasov, V.S. Glushenkov, F.M. Kirillova, A.V. Pokatayev, A.A. Senko and A.I. Tyatyushkin</i>	339
ALGORITHMS FOR DIGITAL COMPUTER CONTROL IV	
On Decentralized Non-Linear Feedback Stabilization of Interconnected Systems <i>A. Hmamed and L. Radouane</i>	345
On the Stochastic Realization Problem for Continuous-Time Nonstationary Stochastic Processes <i>F.A. Badawi</i>	351
Computation of Matrix-Fraction Description for Transfer Function Matrices <i>M. Alvarez and J.M. Sainz de Baranda</i>	357
An Algorithm for Obtaining the Jordan Canonical Form of a Matrix <i>E. Jimenez Moreno</i>	361



## CONTROL SOFTWARE ENGINEERING AND MANAGEMENT I

A Method of Specifying and Designing Data Acquisition Systems <i>K. Kaaramees</i>	365
DATED: A Language to Describe Plans and Strategies in Automatic Deduction Systems <i>F.J. Garijo Mazarío</i>	371
Functional Programming for Discrete Process Control <i>C. Walter</i>	377

## CONTROL SOFTWARE ENGINEERING AND MANAGEMENT II

A Statistical Method for the Detection of Software Errors <i>P. Puhr-Westerheide and B. Krzykacz</i>	383
Highly Reliable Software for Use in Fail-Safe Control <i>J.S.D. Lambrechts and M.G. Rodd</i>	387
The Application of Modern Software Engineering Techniques in the Development of a Process Control Package <i>I. H. MacLeod</i>	393
Evaluation of Statistical Reliability Measures for a Program for the Limitation of Power Density in a Reactor Vessel <i>M. Kersken and B. Krzykacz</i>	399

## CONTROL SOFTWARE ENGINEERING AND MANAGEMENT III

The Production of Program Families Based on Abstract Data Types <i>A. Perez Riesco and J. Ym Cabrera</i>	403
Simulation Program (BASIM) for Personal Computers <i>R.P. Offereins and J.W. Meerman</i>	409
Control Software to Combine Batch Control with Enhanced Continuous Control <i>R.M. Henry, S. Abdelhay, G. Bailley, C. Gray and C. Cable</i>	415

## INDUSTRIAL APPLICATIONS I

Studies for the Application of an Adaptive Controller to Hydroturbine Generators <i>F.R. Rubio, E.F. Camacho, J. Aracil, L.G. Franquelo and J.M.G. Provost</i>	423
Fault Surveillance and Back-Up Policy for a Load-Frequency Digital Controller <i>J.G. Ayza and J.M. Fuertes</i>	429
An On-Line Algorithm for Power System Observability Determination <i>P. Albertos and C. Alvarez</i>	433
Multiprogrammed Stepping Motor Controller <i>M. Levin</i>	441

## INDUSTRIAL APPLICATIONS II

An Interactive Simulation Model of Batchwise, Chemical Production - A Tool to Improve the Communication between the Plant Operators and the Production Process <i>K. Heide, P.M. Jorgensen and H. Soeberg</i>	447
--	-----

Computer Based Process Control System which is Distributed, Hierarchical and Fully Redundant <i>E.H.L. Derksen and K. Buring</i>	453
Implementation and Comparison of Two Extreme Types of Optimal Control <i>H. Soeberg</i>	461
Interactive Software Packages for Polymerisation Reactor Control <i>A.F. Johnson, B. Khaligh and J. Ramsay</i>	467
INDUSTRIAL APPLICATIONS III	
Microprocessor Based Grain Dryer Control <i>U.J. Jaaksoo, E.M. Talvis and J.M. Ummer</i>	473
Two Microprocessor Based Adaptive On-Off Controller Concepts for Practical Application <i>S. Dormeier and B. Jahn</i>	479
Software for Quality Control Applications in Industrial Equipment <i>C. Rey and F. Aldana</i>	489
INDUSTRIAL APPLICATIONS IV	
An Adaptive Control Application to a Lathe <i>A. Vizan and A.M. Sanchez</i>	493
Static Adaptive Algorithms for Computer Control of Diesel Engine Tests <i>F. Sastron</i>	499
Software for a Microcomputer Controlled Laser Synchronization System <i>M. Nechmadi and D. Tabak</i>	505
Microprocessor Based Multifunction Telephone Terminal <i>J. Miñambres-Puig and J. Rivero-Laguna</i>	511
ROUND-TABLE Discussions	
Trends in Computer Control Education <i>Chairman: B. Tamm</i>	517
Computer Control Systems in Industry <i>Chairman: R.W. Gellie</i>	523
Author Index	527

## SPECIFICS OF REAL-TIME SOFTWARE DESIGN FOR MULTIPROCESSOR COMPUTER SYSTEMS

E. A. Trakhtengerts

*Institute of Control Sciences, Moscow, USSR*

**Abstract.** The paper discusses the realization of programs on multiprocessor computer systems. The major attention is paid to the design of systems for automatic program parallelization. Consideration is given to the problem of finding the optimal program structure solved via unification of parallel branches and to the method of program cycle parallelization. Specifics of operation system design for multiprocessor computers of different structure are discussed.

**Keywords.** Multiprocessing system; computer software; data processing; optimal systems; parallel processing.

### INTRODUCTION

Application of multiprocessor real-time control systems gave birth to a number of new problems whose solution requires

- design of parallel algorithms;
- parallelization of the computing process in systems with multiple flow of instructions for parallel sections of programs;
- development of new language tools for effective parallel program execution and debug tools for real-time programs;
- design of translators for an automatic parallelization of user programs to obtain effective program realizing real-time algorithms;
- design of an operation system providing the control of the parallel computing process and its own parallel functioning.

### PARALLEL ALGORITHMS

Algorithm parallelization for multiprocessor systems is a relatively new field of mathematics. Several principally different algorithm parallelization techniques for multiprocessor computer system exist.

1. Widely used is the modification of sequential algorithms and their transformation into parallel. Such modifications may be easily obtained, for instance, by the substitution of sequential processing of individual components of a vector in problems of linear algebra with parallel (vector) processing of all or some of the vector components. A system with a mul-

tiple flow of instruction permits several iterations to be computed in parallel. For example, solving a system of partial differential equations on one processor one may begin computing variable values for the first iteration, then using the first obtained values give a start-up to the second processor and turn to computing the second iteration, and so on.

2. Another way to design parallel algorithms is to isolate a family of parameter-dependent algorithms, the parameter being the degree of computational parallelization. In many cases such a family may be isolated proceeding with a single known algorithm which corresponds to the degree of parallelization equal to one. A family of algorithms for the search of the extremum of a unimodal function may serve as an example. The number of steps required to solve the problem is non-linear dependent on the volume of information processed at each step.

Let  $f(x)$  is a unimodal function defined at the interval  $(0, L)$ . It is required to find its unity-long subinterval  $(0, L_n)$  which contains the point of the extremum of function  $f(x)$ .

It is well known (R. Bellman, 1965) that consecutive computation of the function values in points  $x_1, x_2, \dots, x_n$  yields the solution of this problem if  $L_n \leq F_n$  where  $F_n$  is the Fibonacci number. Estimate the efficiency of the above parallel computational process with a consecutive one (Bronstein, 1978).

For a section of length  $L_n: F_{n-1} < L_n \leq F_n$   $n$  consecutive steps will be required while the number of steps in the parallel process will be approximately  $n \lg 1.6 / \lg ((m+2)/2)$  for an even  $m$ , and  $n \lg 1.6 / \lg ((m+1+2(m-1)(m-2))/4)$  for an uneven  $m$ , where  $m$  is the number of processors. Fig. 1 presents the maximal number of iterations versus the length of the section for  $m=1$  and  $m=15$ .

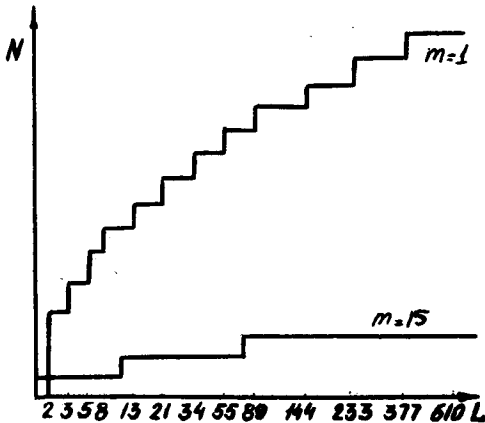


Fig. 1 Number of iterations vs. length of section.

3. Instead of a family of uniform parameter-dependent algorithms one may construct a set of diverse algorithms and find within this set the most effective algorithm in terms of performance speed, minimal memory or frequency of exchanges with external devices. For example, algorithm of recursion computation or reverse of matrices.

4. Finally, special parallel algorithms have appeared lately oriented toward a specific type of a multi-processor computer system. Such are, for instance, algorithms in (Maruyama, 1973) and (Hyalif, 1977). It should be noted that the same algorithms may feature different efficiency under different strategies of parallel computations.

Let us perform the following experiment (Groppen, 1982). Our task is to solve an extremal combinatorial problem of the form

$$\begin{aligned} \sum_{i=1}^n c_i x_i &\rightarrow \max \\ \sum_{j=1}^m a_{ij} x_i &\leq b_j, \quad x_i \in \{0, 1\} \end{aligned} \quad (1)$$

Let the entire set of solutions for problem (1) be divided into  $V$  subsets each surveyed by its own "branch" realizing any of the techniques which guarantee the global optimal solution (direct exhaustive search, Balasch's algorithm, branch-and-boundary methods). In the course

of this survey the branches exchange information on the record and processor elements. The problem is solved when the last branch terminates operation. Formally, the problem of minimizing the search time for the solution of the extremal combinatorial problem is minimax (Groppen, 1982):

$$\begin{aligned} \max_{1 \leq j \leq V} t_{2j} &\rightarrow \min \\ \sum_{t=t_1}^{t_2} \sum_{j=1}^V [Q_j(m_{t,j}) + Q_j(m_{t,q})] &= Q \\ \sum_{j=1}^V m_{t,j} &\leq m; \quad t = t_1, t_1+1, \dots, t_2 \end{aligned}$$

where  $m$  is the total number of the MCS processor elements;  $m_{t,j}$  is the number of the processor elements isolated by the  $j$  branch in the  $t$  instant of time;  $V$  is the number of branches (control devices);  $Q$  is the number of surveyed solutions of the problem;  $Q_j$  is the number of solutions surveyed by the  $j$  branch;  $Q_j(m_{t,j})$  is the number of solutions surveyed at the time  $t$  by the processor elements belonging to the  $j$  branch;  $Q_j(m_{t,q})$  is the number of solutions of the  $j$  branch cut off at the time  $t$  by the  $q$  branch which feature the "best" record at this instant of time;  $t$  is the current solution time;  $t_{1,j}$  is the time of starting the  $j$  branch;  $t_{2,j}$  is the time of terminating the  $j$  branch;  $t_1 = \min t_{1,j}$ ;  $t_2 = \max t_{2,j}$ .

To solve the problem (1) use Balasch's technique (Korbut, 1969). Compare three approaches: a traditional one which realizes the Balasch algorithm on a single-processor computer, and two parallel methods, which require a multi-processor computer system, the first allotting all processor elements to one branch and providing the movement along the branching tree by rushes of  $h = \rho g, \rho$  on each iteration, where  $\rho$  is the number of processor elements in the system, and the second allotting  $m/V$  processor elements to each branch where  $V$  is the number of control devices. The number of variables varied within the range  $4 \leq n \leq 17$  and the number of constraints  $1 \leq l \leq n$ ,  $m=K, V=4$ .

Fig. 2 shows typical dependencies of the gain in the computation time  $q$  on the number of constraints  $l$  under the fixed number of variables  $n$  ( $n=13, 2 \leq l \leq n$ ).

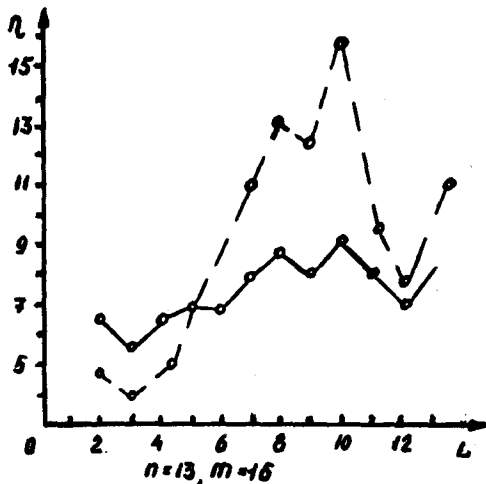


Fig. 2 Computation time vs. number of constraints.

The full broken line in the picture corresponds to the "trustful" parallelization strategy, when all the processor elements are allotted to one branch having the best record. The dotted broken line corresponds to a "distrustful" search to which the following is true:  $\forall t, j$ ,  $m_{t,j} = m/V = 4$  ( $m_{t,j}$  is the number of processor elements allotted to the  $j$  branch at the time  $t$ ). The "trustful" search is obviously more effective under a small number of constraints. The increase of this number makes asynchronous search by several branches with identical number of processor elements more effective. This result, in particular, speaks in favour of applying multiprocessor computer systems with several control devices each having its own vector processor element for the solution of such combinatorial problems featured by a large number of constraints.

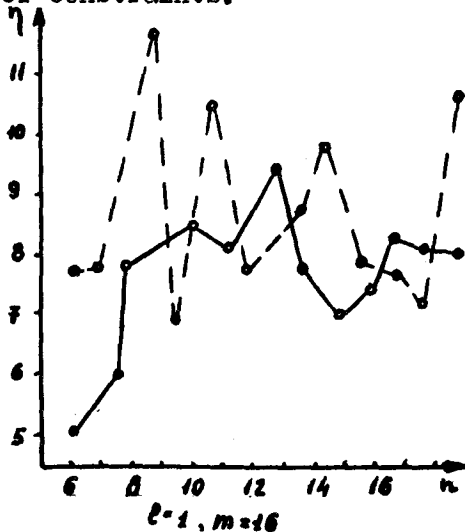


Fig. 3 Computation time vs. number of variables.

The gain in computation time  $\tau$  versus the number of variables of problem (1) is presented in Fig. 3 where the full and the dotted lines correspond to the same algorithms as in Fig. 2. It is easy to see that the increase of the number of variables for problems of the dimensions studied has a less effect on the strategy of multiprocessor systems usage than the increase of the number of constraints.

Thus this example already shows that the use of multiprocessor computer systems for the solution of extremal combinatorial problems proves effective only if the structure of the computer system is adequate to the algorithm-problem combination. For real-time systems the design of algorithms which may be effectively realized on specific computer systems is particularly important.

#### PARALLELIZATION OF THE COMPUTING PROCESS INTO SIMULTANEOUSLY EXECUTED SECTIONS OF PROGRAMS

Simultaneous solution of a problem on several computer devices results in a significant speed-up of the computational process. It is well known however that with the use of  $p$  processors the speed of computation is higher by the factor much less than  $p$ . This is so, in particular, due to time losses resulting from the initiation of parallel program sections execution (herein after referred to as branches) and their synchronization i.e. making one branch wait the execution of some other branches to terminate. Therefore the use of branches with short execution times is often impractical. It should also be said that the availability of branches the number of which exceeds that of the processors does not speed up program execution. This is certainly true if branches contain only processor instructions. Thus appears a problem of generating such branches whose application minimizes the execution time of the program. Let there are  $I$  different sections (primary branches) which comprise a graph of the program with the known execution time for each branch  $t_i$  ( $i=1, 2, \dots, I$ ). The transformation of the graph is in unification of some of its branches. It is implied that all the branches executed prior to these unified branches should be terminated by the time the execution of the unified branches starts. The same order should be maintained for branches executed after the unified ones. Unification of two branches into one reduces the total operation time by  $\tau$ , the time needed



to realize the branch generation and unification instructions. Such branches of the program should be generated that would minimize its time of realization on a finite number of processors,  $Q$ . Assume the last  $I$  branch consists only of one program termination instruction and cannot be unified with other branches of the program. Then the problem is to minimize the start time of  $I$  branch operation (Kut'in, 1981):

$$x_I \rightarrow \min \quad (2)$$

Introduce the following designations:

$$c_{i_1 i_2} = \begin{cases} 1 & \text{if the } i_2 \text{ branch follows} \\ & \text{after } i_1 \\ 0 & \text{otherwise} \end{cases}$$

Let us make all possible unions of branches. Assume their number is  $J$ . To characterize the versions of branch unification introduce the values

$$a_{ij} = \begin{cases} 1 & \text{if the } i \text{ branch is a member} \\ & \text{of the } j \text{ union} \\ 0 & \text{otherwise} \end{cases}$$

To trace the availability of processors introduce the function

$$\delta_i(t_1, t_2) = \begin{cases} 1 & \text{if } t_1 \leq t \leq t_2 \\ 0 & \text{otherwise} \end{cases}$$

It is required to find such unions of branches  $y$  and start times of their operation  $x_j$  ( $j=1, 2, \dots, J$ ) which minimize (2). It is also necessary that the following is true:

$$\sum_{j=1}^J a_{ij} y_j = 1, \quad i=1, 2, \dots, I \quad (3)$$

i.e. all the primary branches are realized. Besides the order of the branches should be maintained set by the matrix with elements  $c_{i_1 i_2}$ :

$$c_{i_k} (y_{i_k} (a_{i_k} x_i + t_i + \tau) - a_{i_s} x_s) \leq 0 \quad (4)$$

Another constraint to be satisfied is the resource limitation:

$$\sum_{j=1}^J \delta_i(x_j, x_j + t_j + \tau) \leq Q \quad \forall i \in \bar{I} \quad (5)$$

where  $\tau$  is a sufficiently large number greater than  $x_I$ . It follows from the problem statement that

$$y_j = 0 \vee 1; \quad x_j \geq 0 \quad (6)$$

The problem (2)-(6) is a non-linear integer problem of mathematical programming. The constraint (5) may be presented explicitly using the variables  $x_{jt}$ :

$$x_{jt} = \begin{cases} 1 & \text{if the } j \text{ union is active at} \\ & \text{the time instant } t \\ 0 & \text{otherwise} \end{cases}$$

The problem (2)-(6) may then be rewritten as the problem of maximization

$$\sum_{j=1}^J x_{jt} \rightarrow \max \quad (7)$$

under constraints

$$\sum a_{ij} y_j = 1 \quad (8)$$

$$c_{i_k} a_{i_k} x_{i_k} y_{i_k} (\sum_{i_s \in \bar{I}} a_{i_s} x_{i_s} - t_{i_k}) \geq 0 \quad (9)$$

$$\sum x_{jt} \leq Q \quad (10)$$

$$x_{jt} = 0 \vee 1; \quad y_j = 0 \vee 1 \quad (11)$$

In contrast to (2)-(6), the problem (7)-(11) is featured by a greater number of variables but it does not contain any implicit functions. It should be noted that the number of unknowns and constraints in a specific problem may be essentially reduced if only the neighbouring branches are united i.e. those outcoming from one point or incoming into one point. To obtain the optimal solution an iterative procedure should be designed using the found value as the initial approximation in the subsequent run of problem (7)-(11). The solution of this problem determines the unification of branches to be used and the start times of their execution. Unfortunately the solution of this problem is greatly hindered by the non-linearity of constraints (9). Only for program structures simple enough whose fragment is shown in Fig. 4 the problem may be solvable.

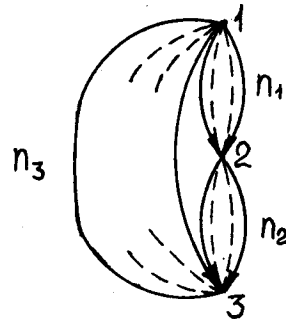


Fig. 4 Structure of a program fragment

Let there are  $n_1$  branches on route 1-2,  $n_2$  branches on route 2-3 and  $n_3$  branches on route 1-3. Let us make all possible unifications of branches on the sections whose number  $I_3$  on route 1-3 is equal  $\sum c_{i_3}$ , on route 1-2  $\sum c_{i_1}$  and on route 2-3  $\sum c_{i_2}$ . The number of unions of routes on route 1-2-3 is then  $I_2 = \sum c_{i_1} \sum c_{i_2}$ . The execution time of the union on the section 1-3 is  $T_i = \sum a_{i_k} t_k + \tau$ ,  $i=1, 2, \dots, I_3$ . Since we consider unions of the branches on section 1-2-3

$$T_i = \sum_{k=1}^{n_1} a_{i_k} t_k + \sum_{k=1}^{n_2} a_{i_k} t_k + \tau, \quad i=1, 2, \dots, I_2.$$

The problem is to find  $x_i$  such that  
 $\max x_i T_i \rightarrow \min$   
 Introduce a variable  $W = \max x_i T_i$ .  
 Then the goal function (2) takes up  
 the form

$$W \rightarrow \min \quad (12)$$

under condition that

$$x_i T_i \leq W \quad (13)$$

Besides, the following constraints  
 should be satisfied:

$$\sum_{i=1}^n a_{ik} x_i = 1 \quad (14)$$

$$\sum x_i \leq Q \quad (15)$$

$$x_i = 0 \vee 1 \quad (16)$$

In other words, one should find  $Q$   
 unions of branches of an approxima-  
 tely equal length which comprise all  
 initial branches.

For programs with an arbitrary ini-  
 tial structure an algorithm for the  
 determination of generalized bran-  
 ches is suggested in (Kutin, 1981)  
 which employs an iterative technique  
 to design the structure of the pro-  
 gram with a good approximation to mi-  
 nimal execution time.

The work of this algorithm may be  
 exemplified with the transformation  
 of the program whose initial struc-  
 ture was shown in Fig.5.

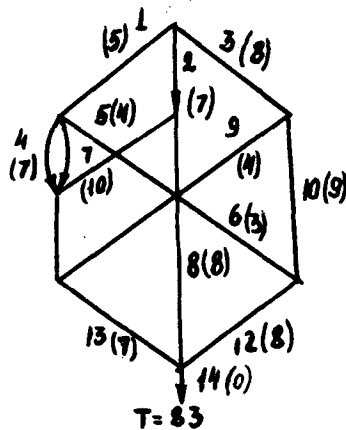


Fig.5. Initial structure of the  
 program.

The structure was designed with the  
 resource  $Q=2$  and  $T=5$ . The program  
 execution time was 83. After the  
 transformation of its structure in  
 accord with the suggested algorithm  
 it was reduced down to 64. The struc-  
 ture obtained is presented in Fig.6.

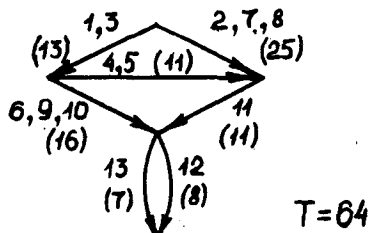


Fig.6. Transformed structure of  
 the program.

Thus the use of the algorithm of the  
 rational unification of branches re-  
 duces the execution time by 22 per-  
 cent. It should be noted that the  
 more the relative execution time for  
 branch generation and unification  
 instructions, the higher the need  
 to unify the branches.

#### PARALLEL PROGRAMS: NEW LANGUAGE AND DEBUG TOOLS

Language tools in multiprocessor  
 systems are intended for the organi-  
 zation of serial-parallel computa-  
 tions.

They differ from "traditional" pro-  
 gramming languages in that they con-  
 tain additional constructs to provi-  
 de parallel realization of program  
 fragments and their timing.

These tools include

- vector and matrix operations and  
 means for masking the operations  
 over vector elements;
- tools for the creation of sections  
 of parallel execution herein after  
 referred to as "branches";
- tools for timing the branches.

The expressions over arrays (vector,  
 matrices) usually make use of the  
 same operations as scalar expres-  
 sions. Normally, specification of  
 subarrays of various types and ope-  
 rations over them is allowed. The  
 conditional logical operator over  
 arrays permits operations only over  
 such array elements to which the va-  
 lue "TRUE" in its logical expres-  
 sion corresponds. In this way ope-  
 rations over vectors are masked which  
 is normally hardware realized.

To organize operations with parallel  
 branches special statements are in-  
 troduced for the description and  
 initiation of branches. In any point  
 of the program one or several bran-  
 ches may be initiated which may be  
 executed in parallel. Usually both  
 static and dynamic determination of  
 parallel branches is provided. In  
 the latter case the number of bran-  
 ches announced in a given point is  
 determined in the course of program  
 execution.

The body of a branch is created sta-  
 tically i.e. in the course of the  
 translation and, normally, does not  
 lend itself to be shaped dynamically.  
 Special statements designate the be-  
 ginning and the end of a branch.

The timing of the computational pro-  
 cess requires introduction of vari-  
 ables or arrays of the "event" type  
 as well as wait and event termina-  
 tion statements. The operands for  
 these are variables or arrays of the  
 "event" type. To provide operation  
 of several branches with the same  
 data statements of the "semaphor"  
 type are introduced.

The level of synchronizing primitives is raised with the help of mechanisms of conditional critical intervals (Hoare, 1972), monitors (Hoare, 1974; Hansen, 1973; Hansen, 1975), sentinels (Keller, 1978), control expressions (Campbell, 1974) and rendezvous (in the Ada language). The major feature of parallel programs which hinder their debug as compared to serial programs is asynchronous execution of sections of a parallel program. This feature makes it difficult to reproduce the situation which leads to an error. Because parallel processes are asynchronous they may access to the same data in different succession. The order of processing may effect the result while the programmer has no means to restore the order of data processing to localize the error. This feature of asynchronous programs is particularly harmful in complex debug of large programs.

#### MEANS OF TRANSLATION PROVIDING AUTOMATIC PARALLELIZATION OF USER PROGRAMS

The analysis and parallelization of programs in the course of its translation may be performed according to the scheme shown in Fig.7.

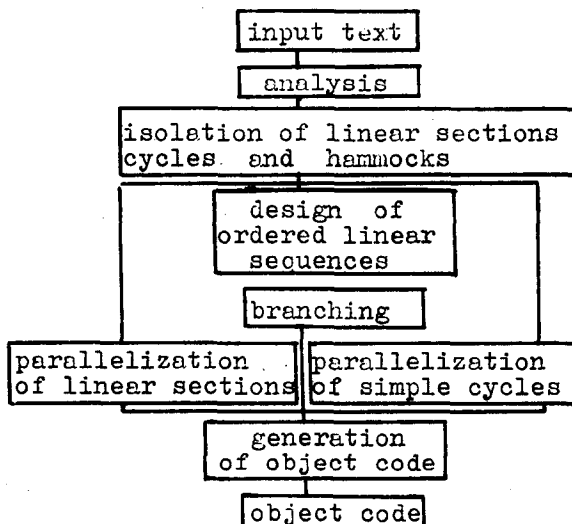


Fig.7. Parallelization process diagram

The process of syntax analysis may also be made parallel (Trakhtengerts, 1981). The analysis helps one to isolate linear sections, simple cycles and hammocks. An algorithm for the isolation of linear section may easily be designed proceeding from its definition. Let us call a simple cycle the program fragment which consists of one or several statements of a cycle or cycle body not containing the transfer of

control beyond this cycle. The boundaries of cycles are found by formal indicators of cycle description in appropriate programming languages. A subgraph with a single input and single output will be referred to as a hammock.

The process of the program graph analysis terminates with the isolation of linear sections, simple cycles and hammocks. After that takes place parallelization inside linear sections, parallelization of simple cycles and design of ordered linear sequences. An ordered linear sequence consists of simple cycles, hammocks and linear sections.

Parallelization inside linear sections, design of ordered linear sequences and parallelization of simple cycles may be performed simultaneously.

Parallelization of linear sections. Inside a section parallelization is performed by statements, while inside a statement arithmetic expressions may be executed in parallel. The transformation of scalar arithmetic expressions for parallel computation is in the reduction of the number of steps required to compute the arithmetic expression. For instance, the computation of the expression

$a + b * c + d$  requires two steps. At the first step,  $b * c$  and  $a + d$  are computed. At the second step the results of the first step are summed up.

Such parallelization of arithmetic expressions and parallel execution of linear section statements unrelated informationally is possible only when a computer system realizes pipeline processing and/or is provided with special ALU's for addition, multiplication, shift and/or several multi-purpose standard processor elements. In this way a substantial speed-up of the computational process is achieved (Trakhtengerts, 1981). Generation and local parallelization of ordered linear sequences. Introduce the concept of an essential arc and an essential statement. Let a control graph of the program  $G$  is given. Each vertex of the graph corresponds to a linear section, simple cycle or hammock. Vertex  $\alpha_i$  and arc  $(\alpha_i)$  will be referred to as essential for vertex  $\alpha_i$  if any other path in graph  $G$  from vertex  $\alpha_i$  along arc  $(\alpha_i)$  toward the output vertex  $\alpha_k$  passes through vertex  $\alpha_j$ , and if a path exists in graph  $G$  from vertex  $\alpha_i$  along arc  $(\alpha_i)$  to the output vertex  $\alpha_k$  escaping vertex  $\alpha_j$ . Proceeding from this definition one may easily design an algorithm for finding essential arcs and vertices. As soon as a set of essential arcs is found this means that the set of

ordered linear sequences is available since each arc defines a certain ordered linear sequence. The ordered linear sequences are generated in the following manner. Assume  $(n_i)$ , an arc outcoming from vertex  $n_i$ , is essential for the set of vertices  $\{n_p\}$ . Then a direct follower of vertex  $n_i$  from the set  $\{n_p\}$  taken along arc  $(n_i)$  is the first element of the ordered linear sequence generated with the help of the arc  $(n_i)$ . A sequence of look-through is established for the elements of the set  $\{n_p\}$  which corresponds to the order of their appearance in graph  $G$ . Within each ordered linear sequence information and logical links between its components are analyzed in a manner similar to the analysis of linear sections. As a result of parallelization of elements for every ordered linear sequence, its elements are displaced with respect to one another within this ordered linear sequence, i.e. they are distributed by local levels.

Cycle parallelization in the course of translation.

Vector computations prove highly efficient for computer systems equipped with vector registers or sets of processor elements. Operations over the elements of vectors in vector operations on such systems are performed an order of magnitude faster than the same operations over scalars. Therefore the transformation of cycle bodies of serial programmes into vector operations significantly reduces program execution time.

The aim of transformation of a serial program cycle body into the vector operation is to perform a vector operation (parallel computation) over those elements of the vector the coordinates of all points of which are parallel to some plane. For instance, such plane for which the condition

$$\sum a_j I_j = \text{const}$$

holds. The value of the constant should change after each cycle body run until all points of the cycle are passed.

As a rule far not every cycle lends itself to parallelization; this may be done only with those which satisfy a number of limitations. Normally the following limitations are imposed upon a cycle body: it should not contain I/O statements, transfer-of-control statements and references to subprogramms and functions whose parameters are generated variables. Certain limitations on the form and order of index expressions should also be satisfied.

The structure of the computer system dictates the use of a particular parallelization technique. Thus, for the ILLIAC-IV type of systems the method of coordinates (Lampport,

1974) may be employed; for systems with sets of asynchronously operating processors, the hyperplane method (Lampport, 1974), the method of parallelipipeds (Val'kovski, 1979), etc.

These methods differ not only in the way parallelization is carried out, but also in the depth of limitation imposed upon the cycles to be transformed.

Parallel execution of the cycle body requires determination of the entire range of values for every index variable within which the vector operation may be performed. In doing so, one should make sure that the execution of the vector operation is equivalent to the initial cycle. Normally this requires the solution of a rather complicated system of integer equations and inequalities (Lebedev, 1979). Therefore parameters of the cycles to be transformed should be specified in the form of constants rather than variables. The entire preparatory work preceding parallelization may then be carried out in the course of the translation rather than execution of the program.

It should be noted that according to the analysis of cycles in Fortran programs, 30 to 60 percent of cycles used in programs lend themselves to automatic parallelization, depending on the computer system structure and, consequently, on the parallelization technique employed.

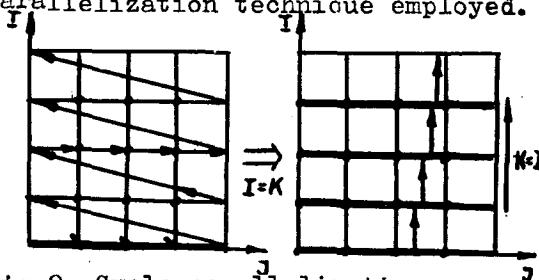


Fig.8. Cycle parallelization diagram.

Figure 8 shows transformation of a simple sequential Fortran cycle

```
DO 1 I=1,N
DO 1 J=1,M
1 A(I,J)=A(J,I)
```

into a parallel cycle

```
DO 1 K=1,N
DO CONC FOR ALL (I,J), 1 < J <= M & I=K
1 A(I,J)=A(J,I)
```

with the transformation matrix  $\varphi \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ . The lefthand part of Fig.8 presents the traditional sequential execution of the cycle. The righthand part shows the parallel way of execution. At first the cycle body is executed for  $I=1$ ,  $1 \leq J \leq M=5$ , then for  $I=2$ ,  $1 \leq J \leq M=5$  and so on.