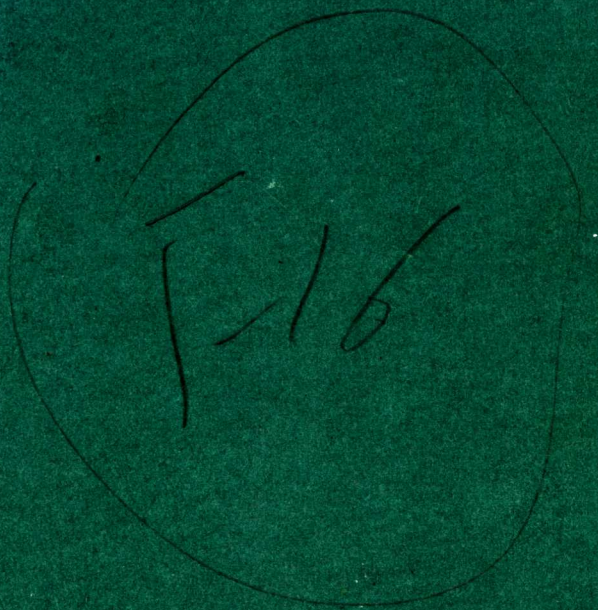
SOFTWARE FOR COMPUTER CONTROL 1982

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

CONTROL SOFTWARE ENGINEERING AND MANAGEMENT I

CONTROL SOFTWARE ENGINEERING AND MANAGEMENT II

CONTROL SOFTWARE ENGINEERING AND MANAGEMENT III

INDUSTRIAL APPLICATIONS I

INDUSTRIAL APPLICATIONS II

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 multiple 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 $x_1, x_2, \ldots, x_n$ 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: $F_{n-1} < L_n \le F_n$ 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 $n \lg 1.6 / \lg((m+1+\sqrt{m^2+1})/2)$ 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 $m=15$.

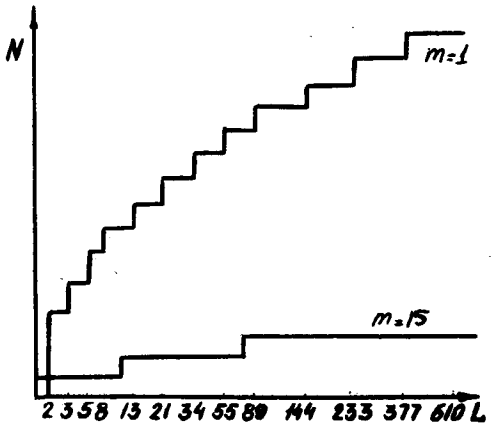


Fig. 1 Number of iterations vs. length of section.

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 multiprocessor 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

$$\sum_{i=1}^{n} c_i x_i \to max$$
$$\sum a_{ij} x_i \le b_j, \quad x_i \in \{0,1\} \qquad (1)$$

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 combinational problem is minimax (Groppen, 1982):

$$\max_{1 \le j \le V} t_{2j} \to min$$
$$\sum_{t=t_1}^{t_2} \sum_{j=1}^{V} [Q_j(m_{t,j}) + Q_j(m_{t,q})] = Q$$
$$\sum_{j=1}^{V} m_{t,j} \le m; \quad t = t_1, t_1+1, \ldots, t_2$$
$$m_{t,j} \ge 0 \quad j = 1,2,\ldots,V$$

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; $t_1 = \min_j t_{1,j}$; $t_2 = \max_j t_{2,j}$.

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 alloting all processor elements to one branch and providing the movement along the branching tree by rushes of $h = \log_2 P$ on each iteration, where P is the number of processor elements in the system, and the second alloting m/V processor elements to each branch where V is the number of control devices. The number of variables varied within the range $4 \le n \le 17$ and the number of constraints $1 \le l \le n$, $m=16$, $V=4$.

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 ($n=13$, $2 \le l \le n$).

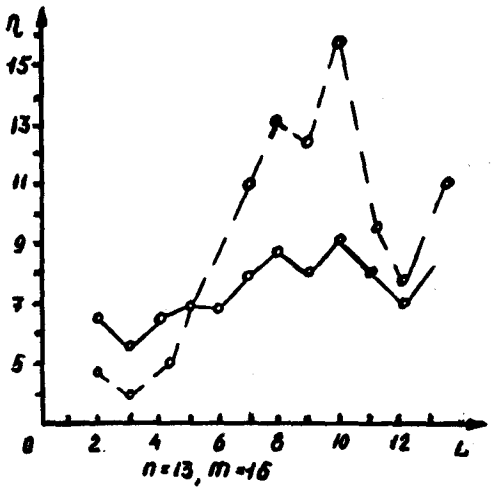


Fig.2 Computation time vs. number of constraints.

The full broken line on the picture corresponds to the "trustful" parallelization strategy, when all the processor elements are alloted 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 alloted 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 ℓ 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 $(i=1,2,\ldots,I)$. 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 τ , 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 (Kutin, 1981):

$$x_I \to min \qquad (2)$$

Introduce the following designations:

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

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

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

To trace the availability of processors introduce the function

$$\delta_t(\theta_1,\theta_2) = \begin{cases} 1 & \text{if } \theta_1 \le t \le \theta_2 \\ 0 & \text{otherwise} \end{cases}$$

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:

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

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

$$c_{i\kappa}(y_e(a_{ie}x_e + t_e + \tau) - a_{\kappa s}x_s) \le 0 \qquad (4)$$

Another constraint to be satisfied is the resourse limitation:

$$\sum_{j=1}^{J} \delta_t(x_j, x_j + t_j + \tau) \le Q \quad \forall t \in \overline{1,T} \qquad (5)$$

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

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

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

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

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

$$\sum_{t=1}^{T} z_{Jt} \to max \qquad (7)$$

under constraints

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

$$c_{i\kappa} a_{ie} z_{et} y_e y_s \Big(\sum_{t=1}^{T} a_{\kappa s} z_{st} - t_s\Big) \ge 0 \qquad (9)$$

$$\sum z_{jt} \le Q \qquad (10)$$

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

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.

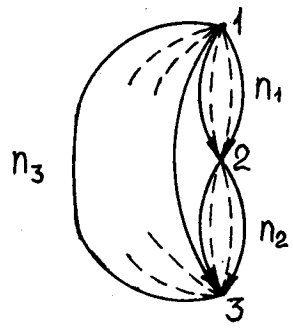


Fig.4 Structure of a program fragment

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 $\sum C^{i}_{n_3}$, on route 1-2 $\sum C^{i}_{n_1}$ and on route 2-3 $\sum C^{i}_{n_2}$. The number of unions of routes on route 1-2-3 is then $I_2 = \sum C^{i}_{n_1} \cdot \sum C^{i}_{n_2}$. The execution time of the union on the section 1-3 is $T_i = \sum a_{i\kappa} t_\kappa + \tau_i$, $i = 1,2,\dots,I_3$. Since we consider unions of the branches on section 1-2-3

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

The problem is to find x_i such that

$$\max x_i T_i \to \min$$

Introduce a variable $W = \max x_i T_i$. Then the goal function (2) takes up the form

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

under condition that

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

Besides, the following constraints should be satisfied:

$$\sum_{i=1}^{I_1, I_2} a_{ik} x_i = 1 \quad (14)$$

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

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

In other words, one should find Q unions of branches of an approximately equal length which comprise all initial branches.

For programs with an arbitrary initial structure an algorithm for the determination of generalized branches is suggested in (Kutin, 1981) which employs an iterative technique to design the structure of the program with a good approximation to minimal execution time.

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

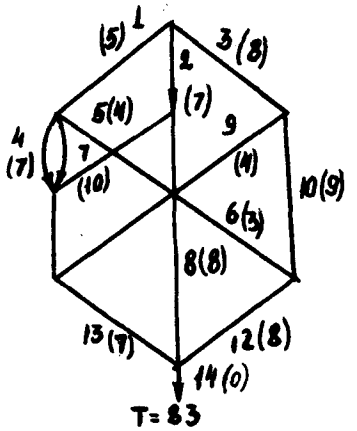


Fig.5. Initial structure of the program.

The structure was designed with the resourse $Q=2$ 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 structure obtained is presented in Fig.6.

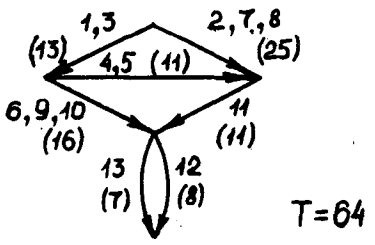


Fig.6. Transformed structure of the program.

Thus the use of the algorithm of the rational unification of branches reduces the execution time by 22 percent. 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 organization of serial-parallel computations.

They differ from "traditional" programming languages in that they contain additional constructs to provide 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 expressions. Normally, specification of subarrays of various types and operations over them is allowed. The conditional logical operator over arrays permits operations only over such array elements to which the value "TRUE" in its logical expression corresponds. In this way operations over vectors are masked which is normally hardware realized.

To organize operations with parallel branches special statements are introduced for the description and initiation of branches. In any point of the program one or several branches 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 branches announced in a given point is determined in the course of program execution.

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

The timing of the computational process requires introduction of variables or arrays of the "event" type as well as wait and event termination 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.

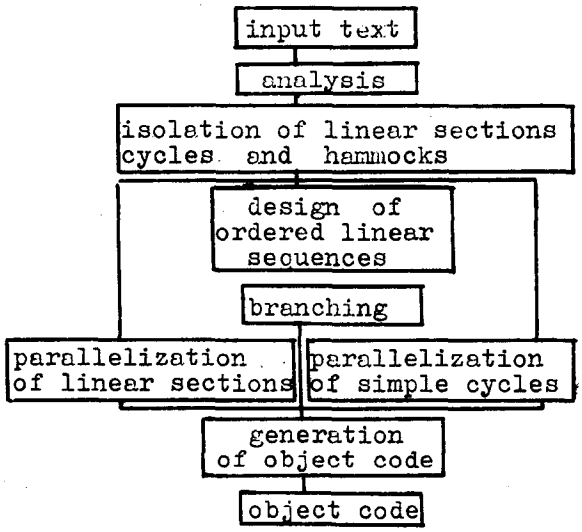


Fig.7. Parallelization process diagram

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 α_i and arc (α_i) will be referred to as essential for vertex α_s if any other path in graph G from vertex α_i along arc (α_i) toward the output vertex α_k passes through vertex α_s, and if a path exists in graph G from vertex α_i along arc (α_i) to the output vertex α_k escaping vertex α_s. 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 programms 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 $\Sigma a_j I_j = const$ 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 ILLIAK-IV type of systems the method of coordinates (Lamport, 1974) may be employed; for systems with sets of asynchronously operating processors, the hyperplane method (Lamport, 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 debth 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.

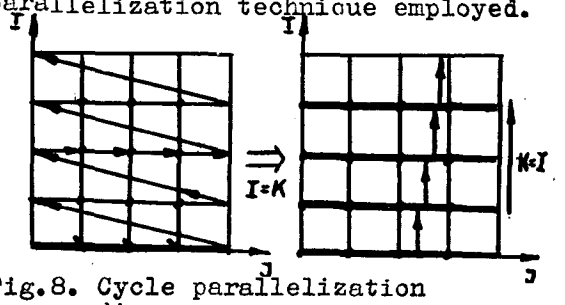


Fig.8. Cycle parallelization diagram.

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 $\varphi=\begin{pmatrix}1&0\\0&0\end{pmatrix}$. 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 $I=1$, $1\le J\le M=5$, then for $I=2$, $1\le J\le M=5$ and so on.