MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

SCHEDULING SPLITTABLE TASKS ON PARALLEL PROCESSORS TO MINIMIZE SCHEDULE LENGTH

Jacek Błażewicz, Wojciech Cellary, Jan Weglarz
Institute of Control Engineering, Technical University of Poznań,
Poznań, Poland

1. Introduction

In deterministic scheduling problems, knowledge of all task execution times, task arrival times and the precedence relations among tasks is assumed. In practice, these assumptions are rarelly directly met; in particular it is usually not possible to know a priori the task execution times; one would at best only have probability distributions for these times. But the analysis of deterministic problems allows the system designer to determine how well the operational scheduler does in comparison to an optimal one and to provide guigance in improving the scheduler. Moreover, the execution time τ_j of task T_j has at least two practical interpretations [3]. Firstly, it may be the upper bound of the task execution time. Scheduling using these values is equivalent to worst case analysis and is applicable to the "hard-real-time" environment with strict deadlines to be observed. Secondly, τ_j may denote the expected value of the execution time of task T_j considered as a random variable. In this paper the problem of scheduling splittable tasks to minimize schedule lenght is considered. An algorithm for finding the optimal schedule is presented for the case of dependent tasks and identical processors. The idea of solving the proble, for heterogeneous processors is also given.

2. Basic assumptions

It will be assumed that the set of tasks $T_1, T_2, \ldots, T_n$, the set of processors $P_1, P_2, \ldots, P_m$, the structure of precedence relations among tasks, and task execution times τ_{ij}, $i = 1,2, \ldots, m$, $j = 1,2, \ldots, n$, where τ_{ij} denotes the execution time of task T_j on processor P_i, are given.

The structure of precedence relations among tasks is usually given in the form of a precedence graph, where nodes correspond to tasks and arcs to precedence relations. A simple example of a precedence graph is shown in Fig. 1.

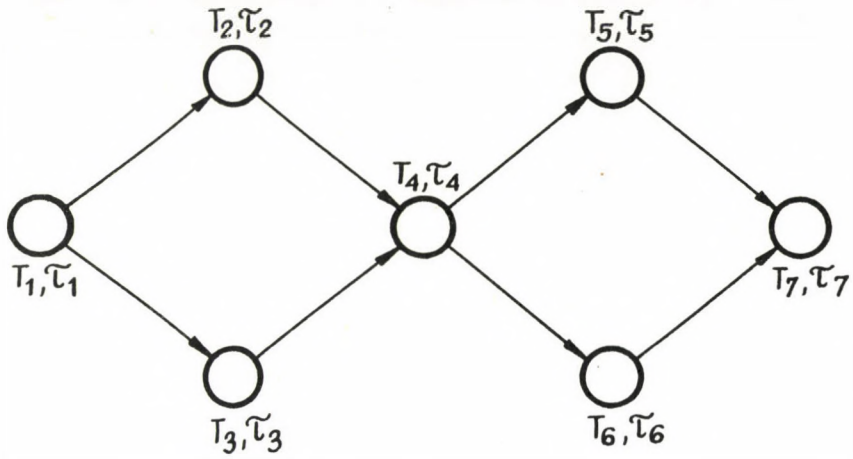

Fig. 1. An example of a precedence graph

Following the most commonly used convection $T_j < T_j$ denotes the fact that task T_i precedes task T_j i.e. task T_j cannot begin processing until task T_i is completed. If no precedence relation among tasks exists, then the tasks are called *independent*, otherwise they are called *dependent* or *precedence related.*

The set of tasks is to be processed on m parallel processors i.e. every processor is capable of processing every task. If the execution times of every individual task are equal one to another for all the processers, then the processors are called *idèntical.* Otherwise, i.e. when there are differences among the processing speeds of the processors, the processsors are called *hęterogeneous.*

A very important feature of a task is whether it may be preempted and then completed, not necessarily on the same processor. Generally, the possibility of task preemption is profitable for some measures for schedule evaluation. If preemption is allowed, it is usually assumed that the processing of the preempted task may resume where it left off without any extra work or time being occasioned by the preemption. This is called a preempt-resume discipline [5], In computer applications, this will not generally be the case, for preemptions may imply the removal of tasks from core storage and subsequent reloading of these tasks. Moreover, in those cases when input/ output operations cannot be overlapped sufficiently, the time delays introduced will be so high that consideration of preemption can frequently be ruled out a priori.

On the other hand examining the preempt-resume discipline is of great importance in systems of parallel processors using a common operating store. Such systems have increasingly many applications in the control of such processes as traffic, telephoneswit control organiza-

tion [6, 12] in which several processors using a common data base computational procedures are being used. It is easy to verify that the possibility of preemption is profitable for improving the schedule lenght.

3. Scheduling splittable tasks to minimize schedule lenght

In this Section scheduling splittable tasks to minimize schedule lenght will be detailed.

Scheduling such tasks has been considered in [9, 10, 11]. The algorithms presented in the these papers concern homogeneous processors and relatively simple precedence relations among tasks. The problem of scheduling independent tasks on processors that are consistently fast or consistently slow for all the tasks was considered in [4, 8]. In the papers mentioned above, non-enumerative algorithms were presented. However, the problem of scheduling dependent, splittable tasks, in the general case, is known to be polynomial complete [4] and hence unlikely to admint a non-enumerative. Thus, for this case, the direct use of scheduling strategies in an oparating system has rather restricted applications. Finding such strategies has, however, practical significance for the following reasons. Firstly, one can use them to estimate an operational scheduler. Secondly, the distance between an optimal solution and a suboptimal one for a heuristic, non-enumerative approach, may be found. Thirdly, enumerative algorithms may be used in computer centres taht perform large and complex numerical computations, but not in a real-time environment.

Below, such scheduling strategies will be presented, which yield some particular advantages [1].

Let us assume, that the processors are identical. Thus, task T_j is characterized by execution time τ_j, $j = 1,2, \ldots, n$. The precedence relations among tasks are given in a form of an activity network (i.e. an acyclic, directed graph with only one origin and only one terminal node) in which arcs correspond to tasks and nodes to events. The precedence graph from Fig. 1. is presented in Fig. 2. in this form.

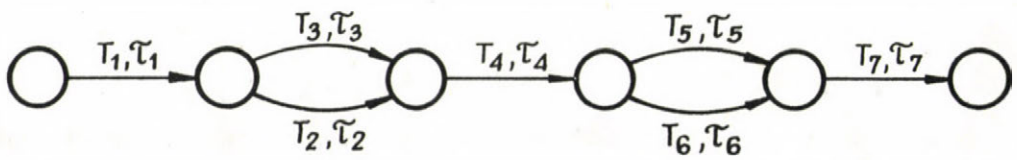

Fig. 2. The set of tasks from Fig. 1. in the form of an activity network

Let the number of nodes of the network be equal to $r + 1$. It is assumed that events are numbered so that event i is not later than event j, when $i < j$. The set of all tasks which can be processed between the occurrence of event k and $k + 1$ will be called the main set and denoted by S_k, $k = 1,2, \ldots, r$. The number of elements of set S_k will be denoted by $|S_k|$.

Now, let us number from 1 to N the feasible sets, i. e. those subsets of all main sets, in which the number of elements is not greated than m.

Let Q_j denote the set of all numbers of the feasible sets in which task T_j may be processed and x_i the duration of set i. Thus the linear programming problem is obtained [1]:

Minimze

$$y = \sum_{i=1}^{N} x_i$$

Subject to

$$(1) \qquad \sum_{i \in Q_j} x_i = \tau_j \qquad j = 1,2, \ldots, n$$

or in matrix notation

$$(1') \qquad \underline{A}\ \underline{x} = \underline{\tau}$$

where $\underline{A}$ is the matrix of coefficients:

$$a_{ij} = \begin{cases} 1 & \text{if} \quad i \in Q_j \\ 0 & \text{otherwise} \end{cases}$$

Obviously, the columns of matrix $\underline{A}$ correspond to the feasible sets. The number of variables in the above problem for m processors is given by the formula:

$$\sum_{k=1}^{r} \left(\sum_{i=1}^{m} \binom{n}{i} k \right) - \sum_{i=1}^{m} \binom{n'}{i} k \Bigg) ,$$

where n_k is the number of elements of the main set S_k;

n_k' is the number of elements of the main set S_k, which composed at least one set $S_1, S_2, \ldots, S_1$, where $1 = k - 1$.

It is assumed in the above formula, that $\binom{n}{i} = 0$ for $n < i$.

It is clear that the number of variables increases rapidly when the number of tasks and processors increases. For example for 5 processors and not very complicated networks containing to task the number of variables is equal to 50, 30 tasks – $2 \cdot 10^3$, 60 tasks – $3 \cdot 10^4$, 100 tasks $2 \cdot 10^5$. If we want to use one of the simplex methods directly for solving the

last problem, 10^7 cells of memory will be needed because of the necessity of memorizing the matrix $\underline{A}$.

As follows from the above, the direct use of simplex methods has a very restriced application.

Bellow, an approach which allows for the great reduction of storage requirements will be shown [1]. On the other hand, in the revised simplex method [7] the elements of the matrix of coefficients (i.e. matrix $\underline{A}$) are not changed during complication. This allows for the generation of single columns of this matrix (or in the other words of feasible subsets of the main set S_k, $k = 1,2, \ldots, r$) in every simplex iteration. Thus, at every moment only one main set and one of its feasible subsets has to be memorized. The use of revised simplex method is also more efficient [7], because in the described problems the number of variables is more than three times greater than the number of constraints.

Now the concept of the generation of feasible subsets will be presented. The number of processors – m, and the *network structure vector* are read as input data. The network structure vector contains the consecutive tasks as the ordered pairs of the nodes. Using the above data, consecutive main sets are generated. First, the tasks which were members of the main sets generated earlier (we will call them old tasks) are added to the main set currently being generated, and then the tasks which composed only this set (we will call them new tasks) are added. Using this information, all feasible and different subsets of all main sets are created. In Fig. 3, the block diagram of the generation of the main set S_k is shown, and in Fig. 4, the block diagram of the generation of the feasible and different subsets of the main set S_k is shown, and in Fig. 4., the block diagram of the generation of the feasible and different subsets of the main set S_k is shown. The point in which the generated subset is used in a simplex iteration is denoted by the block "revised simplex procedure".

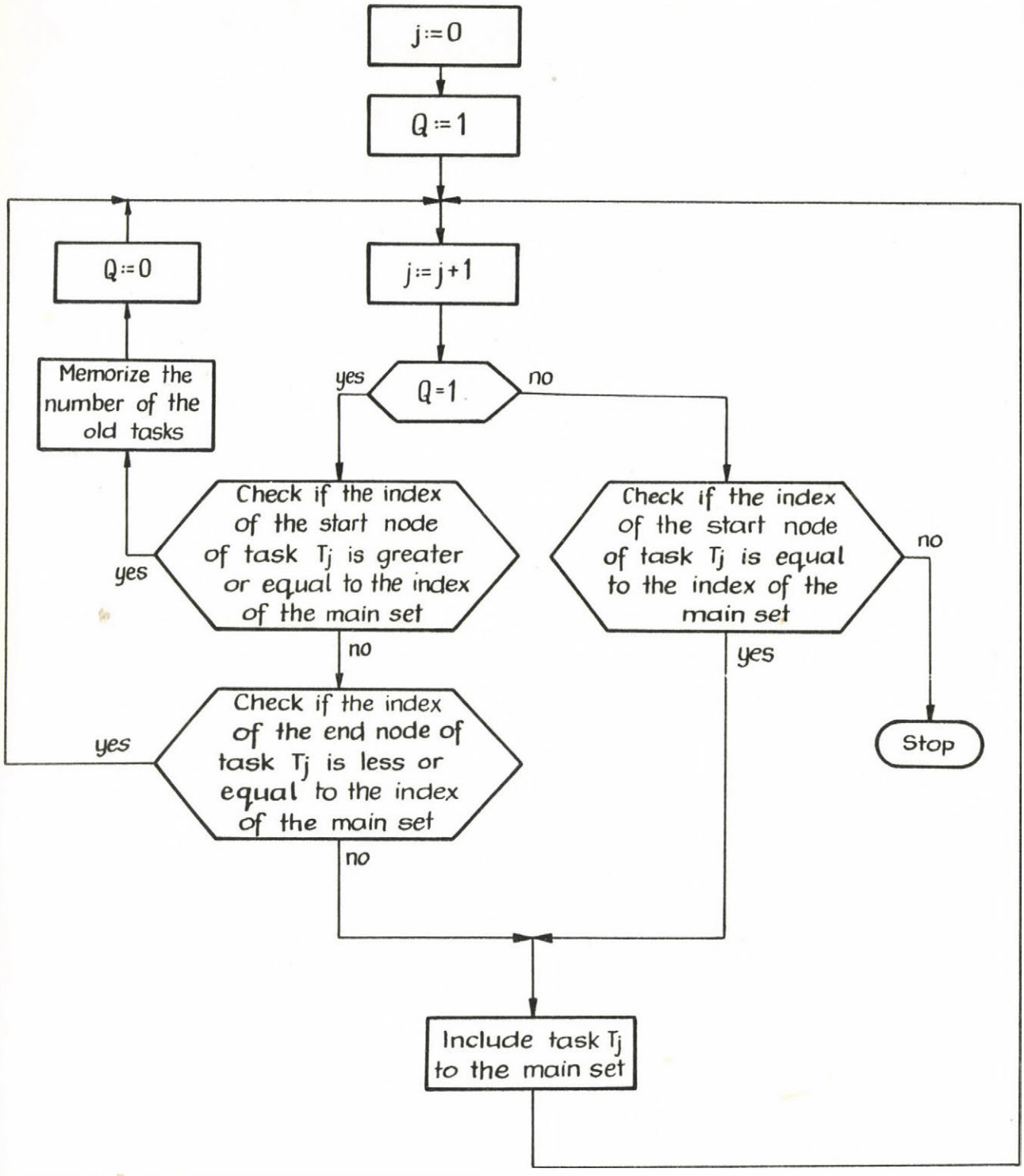

Fig. 3. Block diagram of generation of main set

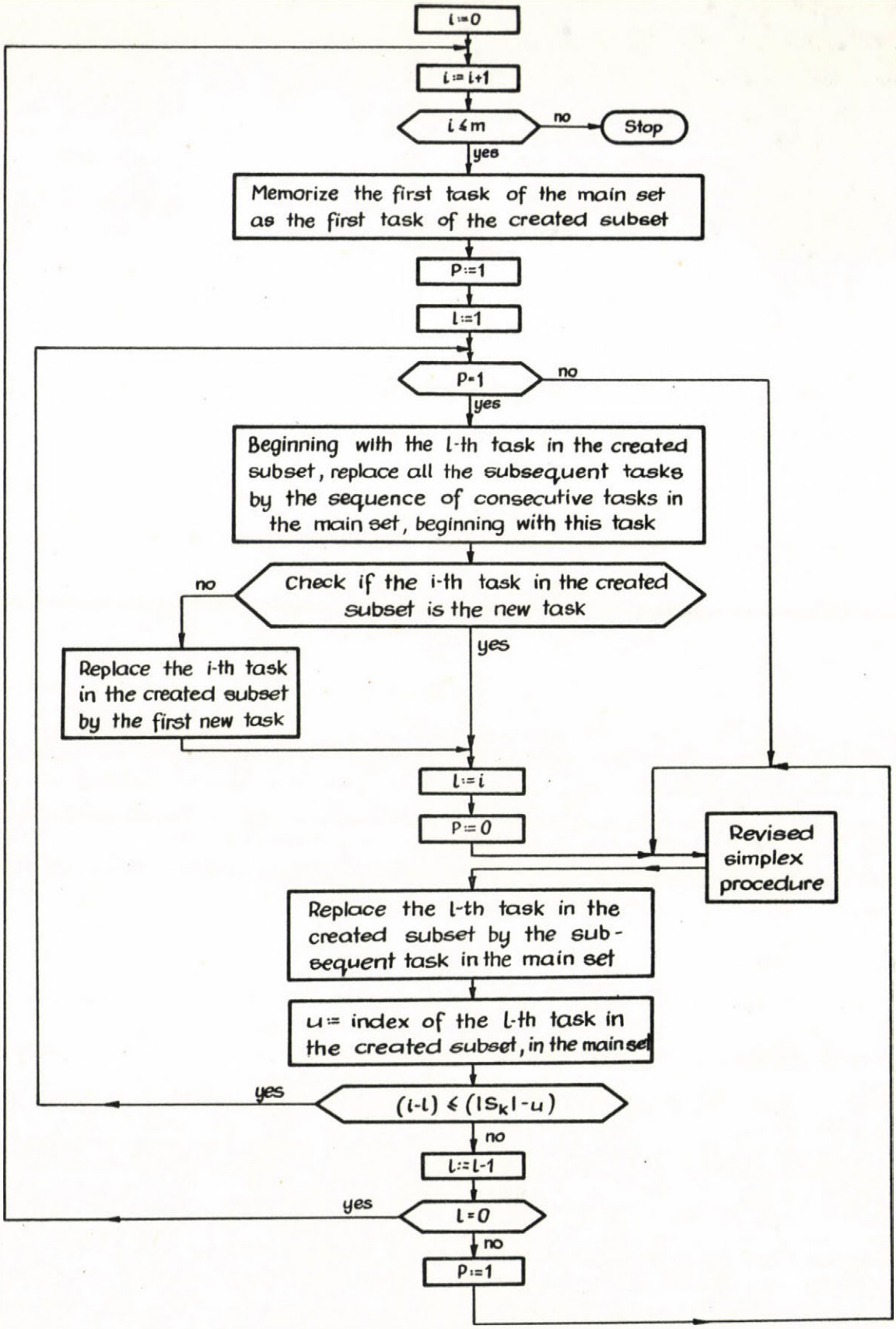

Fig. 4. Block diagram of generation of subsets of main set

Of courge, if the network, node ordering is not given, the obtained schedule is in general a suboptimal one. The optimal schedule may be obtained by choosing the best one from among optimal solutions for all possible orders.

Now let us consider the case of heterogeneous processors [2]. We introduce the following additional denotations:

- K_j, $j = 1,2, \ldots, n$, the set of indices of these main sets in which task T_j may be processed;
- $x_{ijk\epsilon} <0,1>$, $i = 1,2, \ldots, m$; $k\epsilon K_j$, a part of task T_j processed on processor P_i in S_k;
- y_k, $k = 1,2, \ldots, r$, the shedule lenght in S_k;
- $y = \sum_{k=1}^{r} y_k$, the schedule lenght.

Using the above denotations, the following linear programming (LP) problem may be formulated:

Minimize y

Subject to

$$(2) \qquad \sum_{k \epsilon k_j} \sum_{i=1}^{m} x_{ijk} = 1 \qquad j = 1,2, \ldots, n,$$

$$(3) \qquad y_k - \sum_{j \epsilon S_k} x_{ijk} \tau_{ij} \geqslant 0 \qquad \begin{matrix} j = 1,2, \ldots, m \\ k = 1,2, \ldots, r \end{matrix}$$

$$(4) \qquad y_k - \sum_{i=1}^{m} x_{ijk} \tau_{ij} \geqslant 0 \qquad \begin{matrix} j = 1,2, \ldots, n, \\ k \epsilon K_j \end{matrix}$$

Equation (2) guarantees that every task will be processed; inequality (3) defines $y_k's$ as the schedule lenght; inequality (4) assures that it will be possible to obtain a feasible schedule, i.e. one such that no task is processed simulatneously on more than one processor. As the result of solving the described LP probelm, the optimal values y_k^* x_{ijk}^*, $i = 1,2, \ldots, m$; $j = 1,2, \ldots, n$; $k\epsilon K_j$, are obtained. The starting points of parts of tasks x_{ijk}^* are then found [2].

References

[1] J. Blazewicz, Cellary, W. J. Weglarz, Scheduling preemptable tasks on hetegrogeneous processors (submitted for publication).

[2] J. Blazewicz, W. Cellary, J. Weglarz, Some computational problems of scheduling dependent and preemptable tasks to minimize schedule lenght, Foundations of Control Engineering 1, 2(1975).

[3] Jr. E. G. Coffman, P. J. Denning, Operating Systems Theory, Prentice Hall, Englewood Cliffs, N. J. (1973)

[4] Jr. E. G. (ed.) Coffman, Computer and job (shop scheduling theory, Wiley-Interscience (1976)

[5] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of Scheduling. Addison-Wesley, Reading, Mass. (1967)

[6] A. A. Covo, Analysis of multiprocessor control organizations with partial program memory replication, IEEE Trans. Computers C-23, 2 (1974), 113-120.

[7] S. J. Gass, Linear programming , McGraw-Hill, N.Y., (1969)

[8] E. C. Horváth, R. Sethi, Preemptive schedules for independent tasks, Technical Report No 162, Computer Science Dep., Pennsylvania State Univ. (1975).

[9] Mc Naoghton , R., Scheduling with deadlines and loss functions, Management Sci. 6,1 (1959), 1-12.

[10] R.R. Muntz, Coffman E.G., Jr., Optimal preemtive scheduling on two-processor systems, IEEE Trans. Computers C-18, 11 (1969), 1014-1020.

[11] R.R. Muntz, Jr. E.G. Coffman, Preemtive scheduling of real-time tasks on multiprocessor systems, J. Assoc. Comput. Mach. 17,2 (1970), 324-338.

[12] J. Torres, Test and evaluation of computer traffic control system, vol. 9 of Diamond Interchange Traffic Control, Pb-224160, (1973)

Összefoglaló

Jacek Błażewicz, Wojciech Cellary, Jan Weglarz

Felbontható taskok optimális ütemezése párhuzamos processzorokon

A dolgozat n megszakitható task m processzoron való determinisztikus ütemezésével foglalkozik. A minimális terminálási idő meghatározását lineáris programozási feladatra vezeti vissza. Foglalkozik nem azonos teljesitményű processzorok esetével is.

РЕЗЮМЕ

ПРОБЛЕМА РАСПИСАНИЯ РАЗЛАГАЕМЫХ ПРОГРАММ НА ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОРАХ

Блазениц - Келлари - Велгларз

В работе рассматривается детерминическая проблема n и m процессоров. Задача преобразования в проблему линейного программирования.