ELSEVIER

Parallel Computing 20 (1994) 15–28

PARALLEL COMPUTING

Scheduling independent multiprocessor tasks on a uniform k-processor system †

J. Błażewicz [a,*], M. Drozdowski [a], G. Schmidt [b], D. de Werra [c]

[a] *Instytut Informatyki Politechniki Poznańskiej, Poznań, Poland*
[b] *Universität des Saarlandes, Wirtschaftsinformatik II, Saarbrücken, Germany*
[c] *Département de Mathématiques, Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland*

(Received 2 September 1992; revised 12 February 1993)

Abstract

The problem to be addressed is one of scheduling multiprocessor tasks, some of which require more than one processor at a time. We extend this model by introducing a uniform k-processor system consisting of k-tuples of processors having the same speeds. A low order polynomial-time preemptive scheduling algorithms is proposed when schedule length is the performance measure.

Key words: Deterministic scheduling; Multiprocessor task systems; Uniform processors; Preemptive schedule; Scheduling on a hypercube; Complexity analysis; Polynomial-in-time algorithms

1. Introduction

While classical scheduling models assume that each task requires one processor at a time [2,5,7], it turns out that in systems of microprocessors one often has tasks requiring several processors simultaneously [3,16]. It is for instance the case of self-testing multi-microprocessor systems in which one processor is used to test others or in fault detection-systems in which test signals stimulate the elements to be tested; then outputs are analyzed simultaneously [1,8]. New parallel algorithms and corresponding future task systems create another domain of application for this kind of scheduling [9,14]. It is not difficult to give examples of the computa-

* *Corresponding author.*

† Research partially supported by grant KBN and by project CRIT.

SSDI 0167-8191(93)E0064-3

tional problems from mathematics, physics, electronics and computer graphics (e.g. computations on matrices) which can be easily divided into subproblems solvable 'almost' independently in parallel. 'Almost' means that copies of the program solving the problem must communicate from time to time. No matter what kind of communication medium is used, whether it is a shared memory or packet switching system, the advantage of parallelism can only be taken if copies are running in parallel in real time. Otherwise one running module of the program may wait for communication with a module which is temporarily idle. In such a situation the speed of execution depends mainly on the work of the scheduling algorithm swapping tasks on the processor [9]. Thus, in general it is desirable to run in parallel copies of the program requiring more than one processor simultaneously.

When modeling task sets for the above applications, the set of tasks $\mathcal{T}$ is divided into subsets $T^1, T^2, \ldots, T^k$ with $|T^i| = n_i$, $(i = 1, \ldots, k)$ and $n_1 + n_2 + , \ldots, + n_k = n$. Each task $T_l^i \in T^i$ requires exactly i arbitrary processors (from among m identical ones) simultaneously during a prespecified period. All the tasks are independent and each processor can be assigned only one task at a time. The objective is to find a feasible schedule with the minimum length. Our definition of task processing follows [3] while another way of processing including the dependence of a processing time on a number of processors executing a particular task was given in [10] (the so-called parallel task system – PTS). Both models are closely related and dependent on each other. The model considered here is called multiprocessor task system (MTS) and is a special case of PTS. However, NP-hardness results obtained for MTS are then extended to the PTS [10]. Our model (MTS) is as useful as PTS both from theoretical and practical point of view. *Practically* – it is often the case that it is difficult to change the number of processors used by a task. This reflects for example the level of the scheduler in the operating system. *Theoretically* – it is a useful model to derive complexity results and devise scheduling algorithms.

In particular in [10] it has been shown that for nonpreemptive scheduling and precedence constraints consisting of chains MTS problem is strongly NP-hard. For independent tasks and nonpreemptive scheduling the problem is strongly NP-hard for five processors ([10]). Some polynomial-time algorithms for special cases of MTS are also known [3].

For the preemptive MTS case and identical processors already some results have been obtained. For general independent multi-processor task sets with m fixed (where m is the number of processors) the problem can be solved in polynomial time using a linear programming formulation [3]. If there are T^1-tasks and T^k-tasks in a system only, an optimal schedule can be constructed in $O(n)$ time [3]. The special case of the preemptive MTS is scheduling on a hypercube of processors [6,15]. An $O(m^2n^2)$ algorithm was proposed in [15] to schedule preemptively tasks requiring a number of processors which is a power of two.

In this paper, we extend the MTS model by considering a uniform k-processor system. A system of uniform processors can be a model for a computer system consisting of heterogeneous processors or a system in which some processors have to do additional work 'in the background' (e.g. passing a massage in the node-to-

node communication network). A k-processor system consists of disjoint k-tuples of processors. All k processors in the same k-tuple have the same speed $s_{ik+1} = \ldots = s_{ik+k}$ $i = 0, \ldots, (m/k) - 1$. This assumption is justified in practice because of necessary synchronization of parts of the same task running simultaneously. Thus, the slowest processor speed is the speed of the tuple. We give a low order polynomial algorithm for preemptive scheduling T^1- and T^k-tasks. For simplicity, these tasks will be called T-tasks and W-tasks, respectively. This algorithm can be extended [4] to cover also the case of the task sets $T^1, \ldots, T^k$, where for every pair T^i, T^j, $\frac{j}{i} \in Z^+$ (when $j > i$). It is worth mentioning that this algorithm covers also the problem of scheduling on the hypercube described in [6,15] but the methods used here are different.

To set up the subject more precisely let us denote *processing requirements* of T^1-tasks by a vector of standard processing times $\underline{t}^1 = [t_1^1, t_2^1, \ldots, t_{n_1}^1]$. Thus, the time needed to process T_j^1 on a processor of speed s_r is t_j^1/s_r, $j = 1, \ldots, n_1$. Similarly, tasks from sets $T^2, \ldots, T^k$ are characterized by vectors of standard processing times $\underline{t}^2, \ldots, \underline{t}^k$, respectively, and by requirements of $2, \ldots, k$ processors, respectively, at the same time by any task of the respective set. A real processing time of task T_l^i, $i = 2, \ldots, k$, depends on a processing speed s_r of the i-tuple of processors assigned to the task, and it is calculated in the same way as above. All tasks are assumed to be *preemptable*, i.e. their processing may be interrupted at any moment and restarted later (perhaps on another processor) at no cost. The objective of scheduling is to find the shortest possible schedule, i.e. one for which $C_{\max} = \max_{i,l}\{C_l^i\}$ is at its minimum, where C_l^i is a completion time of task T_l^i, $i = 1, \ldots, k$; $l = 1, \ldots, n_i$, in the schedule.

The next section describes the algorithm for two types of tasks (T and W).

2. An algorithm for T- and W-tasks

In this section, the problem of scheduling tasks from sets T^1 and T^k will be considered. As we mentioned, for simplicity reasons these sets will be denoted by T and W, respectively, and their processing requirements by vectors $\underline{t}$ and $\underline{w}$, respectively. Now we describe briefly an idea which lies behind an approach which will be described below. Firstly a lower bound on the schedule length is proposed, and tasks are scheduled using rules that follow a standard uni-processor approach, so that this bound is observed. It appears, that sometimes this bound is exceeded and an infeasible schedule is obtained. It may be proved however, that in such a case no better (shorter) schedule exists. Then, depending on the reason for infeasibility, a new schedule length is calculated and an optimal schedule is constructed. The details of this approach are described below.

When preemptively scheduling independent W- and T-tasks on a uniform k-processor system, a lower bound on the schedule length can be calculated by considering two relaxed versions of the problem:

(1) W-tasks only and

(2) W- and T-tasks where each W-task will be treated as k independent T-tasks with identical processing requirements for each of them.

Let the set of k-processors be ordered by nonincreasing speed factors with $s_1 = \cdots = s_k \geq s_{k+1} = \cdots = s_{2k} \geq \cdots \geq s_{m-k+1} = \cdots = s_m$ and $(m/k) \in Z^+$. Let the sets of tasks for problems (1) and (2), respectively, be ordered according to nonincreasing processing requirements. The schedule lengths for problems (1) and (2) are given by the following formulae:

$$C(1) = \max\left\{\max_{1 \leq g < m/k}\left\{\sum_{j=1}^{g} w_j \Big/ \sum_{i=1}^{g} s_{ki}\right\}, \sum_{j=1}^{n_k} w_j \Big/ \sum_{i=1}^{m/k} s_{ki}\right\};$$

$$C(2) = \max\left\{\max_{1 \leq g < m}\left\{\sum_{j=1}^{g} t_j \Big/ \sum_{i=1}^{g} s_i\right\}, \sum_{j=1}^{N} t_j \Big/ \sum_{i=1}^{m} s_i\right\};$$

where $N = n_1 + kn_k$.

The above formulae follow standard uni-processor task scheduling approach [11]. Clearly $C = \max\{C(1), C(2)\}$ is a lower bound for our original problem and thus $C^*_{\max} \geq C$.

Let the *processing capacity* of each processor in the interval $[0, C]$ before scheduling any task be defined by $PC_i = s_i C$. First we schedule the set of W-tasks according to nonincreasing order of standard processing times by considering only the processor set $P' = \{P_{ki} \mid i = 1, \ldots, (m/k)\}$. After the assignment of the first task set we schedule the set of T-tasks again in a nonincreasing order of standard processing times on the processor set $P = \{P_i \mid i = 1, \ldots, m\}$ taking into account the assignment pattern resulting from scheduling all W-tasks.

The algorithm will use three rules (cf. [13]) which will be given below. It will be applied first to scheduling W-tasks and then T-tasks. For simplicity of notations the processing requirement of each task (T or W) will be denoted by t_j; when scheduling W-tasks they will be treated as T-tasks to be scheduled on (m/k) processors. At each stage of the algorithm, the processors will be ordered according to nonincreasing values PC_i (note, that initially this order coincides with the order of nonincreasing processing speeds).

The first phase of the algorithm consists in applying as long as possible the following rules 1 and 2. (We schedule one task after the other according to the order of nonincreasing standard processing times by applying appropriate rule.) Then rule 3 is used. Suppose we have to schedule T_j and we are considering the first P_l for which $PC_l \geq t_j$.

If $t_j = PC_l$ then apply Rule 1:

Rule 1. Schedule task T_j on the processor P_l in such a way that the interval $[0, C]$ is completely filled with T_j. Set $PC_l := 0$ and renumber the processor set according to nonincreasing processing capacities.

If $PC_l > t_j > PC_{l+1}$ *then apply Rule* 2:

Rule 2. Calculate the time u such that T_j is completely processed in the intervals $[0, u]$ on processor P_l and $[u, C]$ on processor P_{l+1}, respectively. Combine processors P_l and P_{l+1} to a composite P_l with $PC_l := PC_l + PC_{l+1} - t_j$. Set $PC_{l+1} := 0$ and renumber the processor set according to nonincreasing processing capacities.

When rules 1 and 2 can no longer be applied, then we are necessarily in one of the following cases:

(a) $t_j < PC_l$ with either $l = m$ or $PC_{l+1} = \cdots = PC_m = 0$ (i.e. the processing requirements of T_j will not entirely fill the smallest positive remaining capacity of a single processor);

(b) $PC_{l-1} > t_j > PC_l$ and no u (as in Rule 2) can be found. This case can occur only if Rule 3 has already been applied: processors are then loaded in some time intervals in $[0, C]$.

Then we apply the following:

Rule 3. Schedule task T_j and the remaining tasks in any order in the remaining free processing intervals from left to right starting with processor P_l and use a processor P_i, $i < l$, only if P_{i+1} is completely filled.

As $C \geq C(1)$ we know that a feasible schedule for the set of W-tasks must exist. It remains to show that T-tasks can be scheduled in the remaining processing intervals and if not, that no feasible schedule for the given problem instance with schedule length C will exist.

Let us consider the following example first.

Example 1. $k = 3$, $n_1 = 4$, $n_3 = 2$, $\underline{t} = [16,16,16,2]$, $\underline{w} = [16,7]$, $s_1 = s_2 = s_3 = s_4 = s_5 = s_6 = 2$, $s_7 = s_8 = s_9 = 1$. $C(1) = 8$ $C(2) = 8$ $C = 8$. We obtain the schedule of Fig. 1 with rules 1, 2, 3. We see that T_4 cannot be scheduled.

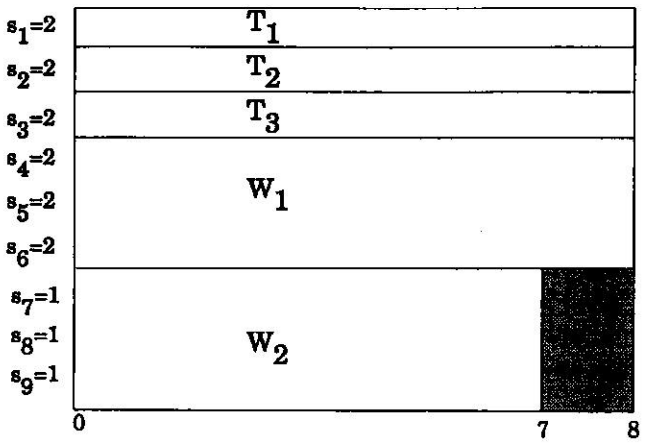

Fig. 1. A partial schedule for Example 1.

From the calculation of $C(2)$ we know that there is enough processing capacity in the interval $[0, C]$ to schedule all the tasks on the given set of processors. In case of infeasibility it might happen that the length of some T-task will prevent the construction of a feasible schedule. To check this, we calculate the processing capacities in the interval $[0, C]$ for the remaining processor system after scheduling the set of W-tasks. Let PC_i^w be the remaining processing capacity of an original or composite processor P_i in the interval $[0, C]$ after the assignment of all W-tasks following the above rules. Remember that these processors are ordered according to nonincreasing *remaining* processing capacities.

From [13] we know that a feasible schedule for the remaining set of T-tasks exists if and only if

$$\sum_{i=1}^{g} PC_i^w \geq \sum_{i=1}^{g} t_i \quad \text{for } g = 1, \ldots, m-1 \text{ and } \sum_{i=1}^{m} PC_i^w \geq \sum_{i=1}^{n_1} t_i \tag{1}$$

and that we can construct it by applying Rules 1–3 to the set of T-tasks using the processor system resulting from the assignment of the W-tasks. Now, assume that no feasible schedule can be found in this way. First, we will show that no other assignment of the set of W-tasks than the one generated by Rules 1–3 can achieve feasibility for the set of T-tasks. Let pc_i^w be the remaining processing capacity of processor P_i in any feasible W-task schedule.

Claim 1. *Using Rules 1–3 we can always guarantee that*

$$\sum_{i=1}^{q} PC_i^w \geq \sum_{i=1}^{q} pc_i^w \text{ for } q = 1, \ldots, m.$$

Proof. Using the above rules we schedule the set of W-tasks one by one. Having selected the first task W_j, assume we are using rules 1 or 2. Let l be the index such that $PC_l > w_j > PC_{l+1}$; then the composite processor has a remaining processing capacity which satisfies $PC_{l+2} \leq PC_l + PC_{l+1} - w_j < PC_{l-1}$ and no reordering of the processors is necessary.

On the other hand, if we combine P_i and P_q $(i < q)$ we will have $PC_i + PC_q - w_j < PC_i$ since $PC_q < w_j$. Let r be the new index of the composite processor after reordering; then we will have $\sum_{h=1}^{r} pc_h < \sum_{h=1}^{r} PC_h$ (and $r > i$). In general, P_i or P_q could be any feasible composition of processors other than P_l and P_{l+1}. Important is that some PC_i has been used. From rule 3, the conclusion is immediate.

After scheduling W_j we have the problem to schedule $n-1$ W-tasks on $m-1$ processors (W was scheduled by applying rules 1 or 2) or on m processors (W was scheduled by applying Rule 3). For the next W-task to be scheduled the same argument applies. Induction on the number of tasks proves the claim. □

From Claim 1 it can be concluded that if there is no feasible schedule for the set of T-tasks after scheduling all W-tasks according to Rules 1–3, then also no other

assignment of the set of W-tasks could result in a feasible schedule in $[0, C]$ for both task types. Assume, that the original set of processors is now transformed into a set where composite processors created in the assignment of W-tasks also appear. Partially filled and empty processors have been combined to composite totally filled and totally empty ones. Let the transformed processor system now be numbered by $1,\ldots,m$ according to nonincreasing processing capacities. In the following discussion by the index of a composite processor we mean index of processor for time $\tau = 0$.

There exists a feasible schedule for the ordered set of T-tasks, if and only if (1) holds. In case of infeasibility there will be at least one task T_j which causes *dead processing capacity* (DP). Dead capacity for T_j means that in order to process this task it should be assigned to more than one processor at a time (which is forbidden), because there is enough processing capacity in the whole system but not enough on any available processor (precise definition of DP is given in Eq. (2) below). Suppose inequalities (1) are not satisfied for l tasks. For the ease of notation we name tasks $T_j,\ldots,T_{j+l-1}$ that cause dead processing capacity by $T_1^*,\ldots,T_l^*$, respectively. Let for some j $(T_1^* = T_j)$ $\Sigma_{i=1}^{j} PC_i^w - \Sigma_{i=1}^{j-1} t_i - t_j^* < 0$. Dead processing capacity for T_1^* is, thus, $DP_1 = \Sigma_{i=1}^{j} t_i - \Sigma_{i=1}^{j} PC_i^w$ and for $T_2^*,\ldots,T_l^*$ respectively

$$DP_2 = \sum_{i=1}^{j+1} t_i - \sum_{i=1}^{j+1} PC_i^w - DP_1,\ldots, \quad DP_l = \sum_{i=1}^{j+l-1} t_i - \sum_{i=1}^{j+l-1} PC_i^w - \sum_{i=1}^{l-1} DP_i. \quad (2)$$

There are two cases of infeasibility.

Case 1 (Fig. 2). Here the dead processing capacity follows from the fact that in bound $C(2)$ splitting of W-task is assumed and tasks assigned according to Rule 2 are assumed to share a processor capacity fairly. Let P_j be a processor on which

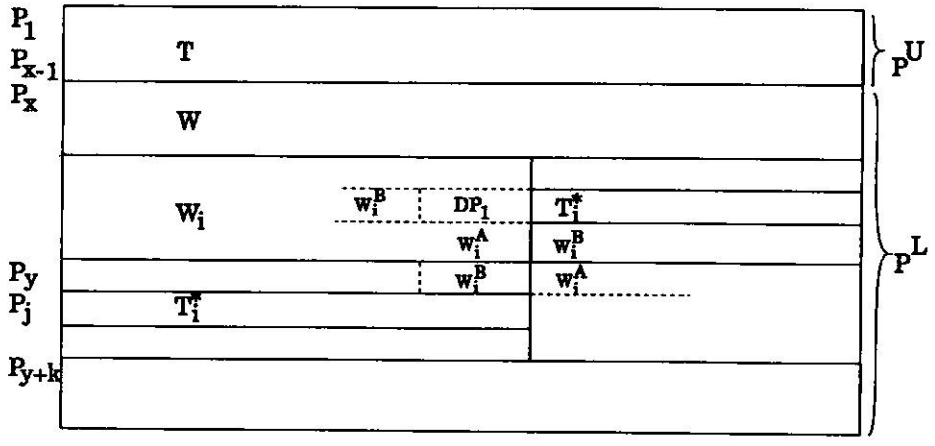

Fig. 2. A partial schedule after assignment of W-tasks in Case 1. Dotted lines show partial layout of W-task neighboring T, treated as k independent T-tasks as in C(2).

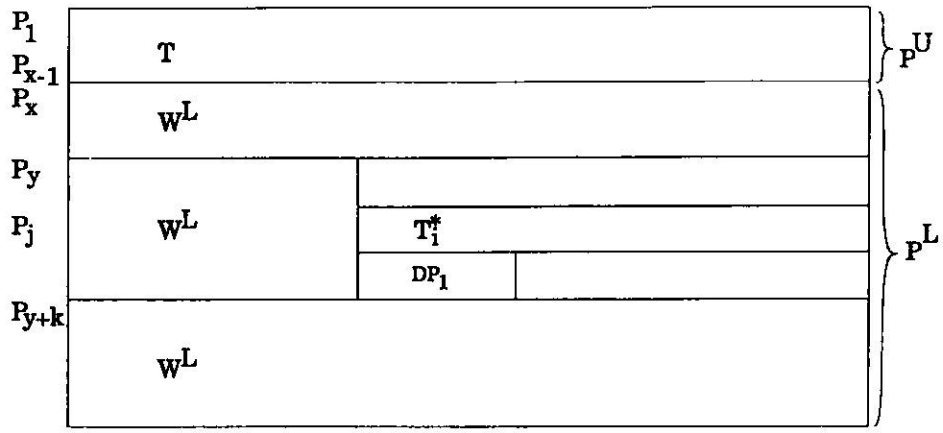

Fig. 3. Partial assignment in Case 2 of infeasibility.

the first task with DP, i.e. $T_1^* = T_j$ has appeared. In this case there is at least one processor below P_j completely free in the interval $[0, C]$. We denote by P_y the first processor from the k-processor to which P_j belongs. P_{x-1} is the slowest processor completely free in the interval $[0, C]$ after the assignment of W-tasks, and $x - 1 < j$. $P_x, \ldots, P_{y-1}$ are filled by W-tasks scheduled according to rules 1,2. The processor system may be now divided into two parts: an upper part P^U where processors P_1 through P_{x-1} are located and a lower part P^L which all other processors belong to.

Case 2 (Fig. 3). Dead processing capacity follows here from the 'nose' created by the non-splitted W-task assigned according to Rule 3. Let P_j, P_{x-1}, P_x, P_y be defined as above, but there is no completely free processor below P_j. Again, the processor system can be separated into two (possibly empty) parts of processors: P^U and P^L. The upper part P^U consists of processors $P_1, \ldots, P_{x-1}$ such that all W-tasks which are scheduled on P^U, now denoted by W^U, are assigned according to rules 1 or 2 and $P_{x-k}, \ldots, P_{x-1}$ do not process W-tasks at all. The lower part P^L consists of processors $P_x, \ldots, P_m$. W-tasks in the latter part of the schedule will now be denoted by W^L.

If no feasible schedule exists we must lengthen the current schedule. A lower bound for lengthening it is

$$\delta_1 = \max_{1 \le g \le l} \left\{ \sum_{i=1}^{g} DP_i \Big/ \sum_{i=1}^{g} s_i \right\}.$$

This follows from the fact that all processing capacities of processor system are shared fairly. On the other hand, we would have an upper bound by lengthening the schedule by

$$\delta_2 = \sum_{i=1}^{l} DP_i / s_1$$

where s_1 is the speed of the fastest processor. If we would lengthen the schedule each time by δ_1 we had to do it at most $O(\sum_{i=1}^{g} s_i)$ times to reach δ_2. Without loss of generality we can divide all processor speeds and all processing times of tasks by $\max_{1 \le i \le m}\{s_i\} = s_1$ and with this $O(\sum_{i=1}^{g} s_i)$ becomes $O(m)$. So we know that we have to lengthen our schedule at most $O(m)$ times. A more detailed analysis of the procedure to lengthen the schedule is given by Claim 2 and Theorem 1.

Claim 2. *If there is no feasible schedule for the set of T-tasks after scheduling W-tasks according to Rules* 1–3, *i.e.* $DP_i > 0$, *then one has to lengthen our schedule by at least*

$$\delta = \max_{1 \le g \le l} \left\{ \sum_{i=1}^{g} DP_i \Big/ \left(\sum_{i=1}^{x-1} s_i + \frac{j-y+g}{k} \sum_{i=x}^{a} s_i \right) \right\}$$

$$\text{where } a = \begin{cases} y+k-1 & \text{case 1} \\ m & \text{case 2} \end{cases}$$

Proof. Consider any schedule of W-tasks which gives a certain dead processing capacity DP. In such a case there exists a schedule of the type constructed by rules 1, 2, 3 for which the dead processing is not larger then DP according to Claim 1. Let ϵ be the minimum amount of time by which we have to lengthen the schedule to find a feasible assignment for our task set. Two cases of infeasibility are considered separately.

Case 1. Rescheduling T-tasks and the remaining W^U-tasks, if there are any, on P^U processors results in at most $\epsilon \sum_{i=1}^{x-1} s_i$ additional processing capacity. Assigning ϵ to P^L and rescheduling W^L results in additional processing capacity only from processors $P_x, \ldots, P_{y+k-1}$. This comes from the fact that there is some completely free processor below P_j, and that the k-processor $P_y, \ldots, P_{y+k-1}$ has been completely free before scheduling T_1^*. Following that fact processors $P_{y+k}, \ldots, P_m$ (original ordering) have not been combined with any processor from among $P_1, \ldots, P_{y+k-1}$ and do not appear in inequalities $\sum_{i=1}^{g} PC_i^w \ge \sum_{i=1}^{g} t_i$ before P_j and $T_j = T_1^*$. From [11] we know that a schedule exists iff (1) holds. To guarantee the existence of a feasible schedule we have to change directions of additional inequalities (from (2)):

$$\sum_{i=1}^{j+g-1} PC_i^w - \sum_{i=1}^{j-1} t_i - \sum_{i=1}^{g} t_i^* < 0 \quad \text{for } g = 1, \ldots, l \quad (T_1^* = T_j).$$

From that we have

$$\sum_{i=1}^{j+g-1} PC_i^w - \sum_{i=1}^{j-1} t_i - \sum_{i=1}^{g} t_i^* + \sum_{i=1}^{g} DP_i = 0 \quad \text{for } g = 1, \ldots, l. \tag{3}$$

After lengthening the schedule, the processing capacity of every original processors grows to $(PC_i^w)' = PC_i^w + \epsilon s_i$. Thus, in our inequalities we should have:

$$\sum_{i=1}^{j+g-1} (PC_i^w)' - \sum_{i=1}^{j-1} t_i - \sum_{i=1}^{g} t_i^* \ge 0 \quad \text{for } g = 1, \ldots, l. \tag{4}$$

If after lengthening the schedule, a division of the processor set to subsets P^U and P^L is not changed we have

$$\sum_{i=1}^{j+g-1} (PC_i^w)' = \sum_{i=1}^{j+g-1} PC_i^w + \epsilon\left(\sum_{i=1}^{x-1} s_i + \frac{j-y+g}{k} \sum_{i=x}^{y+k-1} s_i\right) \tag{5}$$

where $\sum_{i=1}^{x-1} s_i$ comes from P^U, and $((j-y+g)/k)\sum_{i=x}^{y+k-1} s_i$ from P^L. (For the case where the structure of the division to P^U and P^L has changed see Theorem 1 below). From (4) and (5) we have

$$\sum_{i=1}^{j+g-1} PC_i^w + \epsilon\left(\sum_{i=1}^{x-1} s_i + \frac{j-y+g}{k} \sum_{i=x}^{y+k-1} s_i\right) - \sum_{i=1}^{j-1} t_i - \sum_{i=1}^{j+g-1} t_i^* \geq 0 \tag{6}$$

and then from (3) and (6)

$$\epsilon\left(\sum_{i=1}^{x-1} s_i + \frac{j-y+g}{k} \sum_{i=x}^{y+k-1} s_i\right) \geq \sum_{i=1}^{g} DP_i.$$

Thus $\epsilon \geq \delta = \max_{1 \leq g \leq l}\{\sum_{i=1}^{g} DP_i / (\sum_{i=1}^{x-1} s_i + ((j-y+g)/k)\sum_{i=x}^{y+k-1} s_i)\}$ is the minimal amount of time by which we have to lengthen the schedule.

Case 2. Because there is no completely free k-processor in the interval $[0, C]$ assigning ϵ to $P_x, \ldots, P_m$ of P^L and rescheduling W^L results in involving these processors in inequalities (1) in the composite processor $P_x, \ldots, P_{x+k-1}$. This results in creating an additional processing capacity on $P_x, \ldots, P_{x+k-1}$ of $(\epsilon/k)\sum_{i=x}^{m} s_i$ after rescheduling all W^L-tasks. Thus, instead of (5) we have

$$\sum_{i=1}^{j+g-1} (PC_i^w)' = \sum_{i=1}^{j+g-1} PC_i^w + \epsilon\left(\sum_{i=1}^{x-1} s_i + \frac{j-y+g}{k} \sum_{i=x}^{m} s_i\right)$$

and the same argument applies (cf. (5,6)). □

Now, one of the following two cases will happen. We will find (a) a feasible schedule having length $C_{\max} = C + \delta$ and we are done, or (b) that there is no feasible schedule with length $C + \delta$ and we have to lengthen the schedule at least one more time. The necessary and sufficient conditions for the existence of a feasible schedule of length equal $C + \delta$ are given in the following Theorem 1.

Theorem 1. *After lengthening the schedule by δ (as defined in Claim 2) there exists a feasible assignment for our task set, if and only if P^U has at least the same number of processors as before.*

Proof. We will prove the theorem by cases. Before lengthening the schedule we had $\sum_{i=1}^{g} PC_i^w - \sum_{i=1}^{g} t_i \geq 0$ for $g = 1, \ldots, j-1$ and for some j $(y < j < y+k-1)$ $\sum_{i=1}^{g} PC_i^w - \sum_{i=1}^{g} t_i < 0$ for $g = j, \ldots, j+l-1$. After lengthening the schedule the processing capacity of every original processor grows to $(PC_i^w)' = PC_i + \delta s_i$ and $P^U = \{P_1, \ldots, P_z\}$. We will prove this theorem by contradiction.

First, consider the situation in which P^U has now less processors ($z < x$), P^L has more and the schedule is feasible. We will prove that it is impossible. For we have

$$\sum_{i=1}^{g}(PC_i^w)' - \sum_{i=1}^{g}PC_i^w = \delta\sum_{i=1}^{g}s_i \geq 0 \quad \text{for } g = 1,\ldots,z-1$$

and

$$\sum_{i=1}^{g}(PC_i^w)' - \sum_{i=1}^{g}PC_i^w = \delta\left(\sum_{i=1}^{z-1}s_i + \frac{g-z+1}{k}\sum_{i=z}^{a}s_i\right) \geq 0$$

$$\text{for } g = z,\ldots,z+k-1. \tag{7}$$

So, we gain processing capacity that is not big enough to satisfy our needs since what we need is at least (Claim 2)

$$\delta\left(\sum_{i=1}^{x-1}s_i + \frac{j-y+g}{k}\sum_{i=x}^{a}\right) \quad \text{for } g = 1,\ldots,l. \tag{8}$$

The deficit of processing capacity is (from (7) and (8))

$$\delta\left(\sum_{i=1}^{x-1}s_i + \frac{j-y+g}{k}\sum_{i=x}^{a}s_i - \sum_{i=1}^{z-1}s_i - \frac{j-y+g}{k}\sum_{i=z}^{a}s_i\right)$$

$$= \delta\left(\sum_{i=z}^{x-1}s_i - \frac{j-y+g}{k}\sum_{i=z}^{x-1}s_i\right) = \delta\left(1-\frac{j-y+g}{k}\right)\sum_{i=z}^{x-1}s_i \geq 0$$

$$\text{for } g = 1,\ldots,l.$$

From this fact we conclude, that for at least one task, the schedule is not feasible.

Now, consider the case where P^U and P^L are not changed ($z = x$) and there is no feasible schedule. The processing capacity we gain is

$$\sum_{i=1}^{g}(PC_i^w)' - \sum_{i=1}^{g}PC_i^w = \delta\sum_{i=1}^{g}s_i \geq 0 \quad \text{for } g = 1,\ldots,x-1$$

and

$$\sum_{i=1}^{g}(PC_i^w)' - \sum_{i=1}^{g}PC_i^w = \delta\left(\sum_{i=1}^{x-1}s_i + \frac{g-x+1}{k}\sum_{i=x}^{a}s_i\right) \geq 0$$

$$\text{for } g = x,\ldots,x+k-1. \tag{9}$$

Suppose there is no feasible schedule, then for some g ($x+y-j < g < x+k-1$) we have $\sum_{i=1}^{g}(PC_i^w)' - \sum_{i=1}^{g}t_i < 0$. We substitute $\sum_{i=1}^{g}(PC_i^w)'$ and the result is (from (9))

$$\sum_{i=1}^{g}PC_i^w + \delta\left(\sum_{i=1}^{x-1}s_i + \frac{g-x+1}{k}\sum_{i=x}^{a}s_i\right) - \sum_{i=1}^{g}t_i < 0. \tag{10}$$

From Claim 2 we have

$$\delta = \max_{1 \le h \le l} \left\{ \sum_{i=1}^{h} DP_i \Big/ \left(\sum_{i=1}^{x-1} s_i + \frac{j-y+h}{k} \sum_{i=x}^{a} s_i \right) \right\}. \tag{11}$$

Now, the schedule is lengthened and there is less dead processing capacity than before. Following that fact inequality (10) may not be satisfied because by substitution of δ we have (from (10) and (11))

$$\sum_{i=1}^{g} PC_i^w - \sum_{i=1}^{g} t_i + \sum_{i=1}^{g-x-y+j} DP_i < 0$$

and $\sum_{i=1}^{g} PC_i^w$ is less than before lengthening the schedule. This fact obviously contradicts (3). We may conclude that if there is no change in division of $\mathscr{P}$ into P^U and P^L then schedule with $C = C + \delta$ is feasible.

Finally, consider the case where P^U has more processors than before rescheduling ($z > x$) and schedule is not feasible.

Inequality $z > x$ means that W^L-tasks scheduled previously on processors $P_x, \ldots, P_{x+k-1}$ are now scheduled on slower processors ($P_{x+k}, \ldots, P_{y+k-1}$). This implies that:

$$\sum_{i=1}^{g} (PC_i^w)' \ge \sum_{i=1}^{g} PC_i^w + \delta \left(\sum_{i=1}^{x-1} s_i + \frac{g-x+1}{k} \sum_{i=x}^{a} s_i \right) \quad \text{for } g = x, \ldots, x+k-1$$

because we gain more from processors $P_x, \ldots, P_{z-1}$ than expected. Then the same arguments as to Eqs. (10) and (11) are applied. We see that theorem is proved. □

Thus, at each time there is a change in P^U and P^L after lengthening the schedule by δ, we have to increase the schedule length once again. Since there are at most $(x-1)/k < m$ W-tasks scheduled according to rules 1 and 2 after the first task assignment a new 'nose' on P^U does not occur more than m times.

Note that from inequalities (1) we cannot exclude simultaneous occurrence of Case 1 (even multiple) and Case 2 of infeasibility. This does not change proof of Claim 2 or Theorem 1. Every infeasibility occurrence is considered locally in these proofs, while calculating δ we have to keep in mind what kind of infeasibility corresponds to which T and k-processor (defining respectively P_x, P_y, **a**). Simultaneous handling of all groups of infeasibility (i.e. case 1 and 2) leads to an improvement of algorithm's complexity.

The algorithm to solve our scheduling problem can now be formulated as follows.

Algorithm 1.

Step 1. Calculate C, and schedule all W-tasks in $[0, C]$ using Rules 1–3;

Step 2. **While** no feasible schedule for T-tasks exists **Do**

Begin

Calculate δ; $C := C + \delta$;

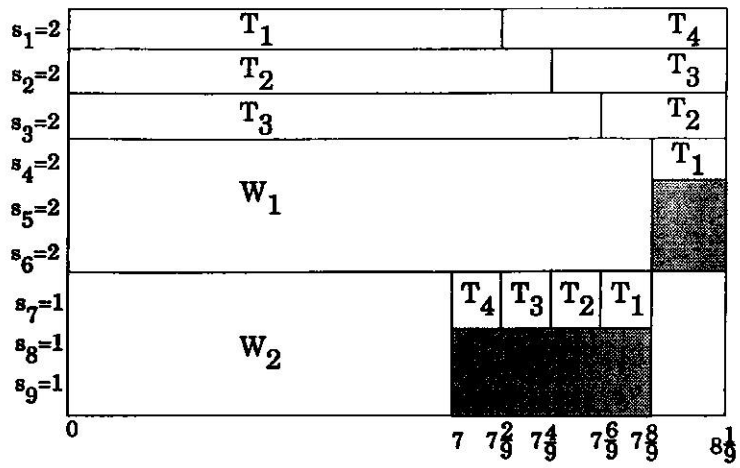

Fig. 4. Example 1 – an optimal schedule.

Schedule W-tasks in $[0, C]$ using Rules 1–3;
End;

Step 3. schedule T-tasks.

Calculating the lower bound needs $O(n \log n)$ time. The application of Rules 1–3 has time complexity $O(n)$ and the inner loop of the algorithm (Step 2) will be carried out less than $O(m)$ times because the situation described in Theorem 1 may not happen more than $O(m)$ times. The calculation of δ from inequalities (2) and Claim 2 lasts $O(m)$ since case 1 may happen $O(m)$ times and Case 2 only once for Claim 2. So, we have a total time complexity of $O(nm + n \log n)$ to solve our problem and of $O(nm)$ for constructing an optimal schedule.

Finally, let us go back to the example of Fig. 1. We have already gone through Step 1 and come to Step 2; no feasible schedule could be found. We go to the inner loop of Step 2. We have $x = 4$, $j = y = 7$, $l = 1$, $\mathbf{a} = \mathbf{m}$, and compute: $DP_1 = 1$ $\delta = 1/9$. Proceeding to Step 3 we get the schedule given in Fig. 4. It is feasible.

3. Conclusions

In the paper, a new model of deterministic scheduling, applicable in multimicroprocessor systems such as shared memory multiprocessors or hypercubes of processors, has been considered. It has been assumed that any task may require more than one processor at a time. The presented $O(nm + n \log n)$ time algorithms find a minimum length schedule on uniform processors under the assumption that tasks require one or k-processors simultaneously for their processing. This algorithm can be extended ([4]) to cover the case of tasks requiring certain numbers from one to k processors (e.g. 1, 2, 4, 8,..., m processors). A more general problem where

tasks may require any fixed number of processors from the set $\{1, \ldots, k\}$ may be solved via linear programming approach [4]. Further generalizations include, among others, deadline scheduling problems which are very important from the practical point of view. These problems are now being studied.

References

[1] A. Avizienis, Fault tolerance: the survival attribute of digital systems, *Proc. IEEE 66* (1978) 1109–1125.
[2] K. Baker, *Introduction to Sequenching and Scheduling* (Wiley, New York, 1974).
[3] J. Błażewicz, M. Drabowski and J. Węglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Trans. Comput.* C35 (1986) 389–393.
[4] J. Błażewicz, M. Drozdowski, G. Schmidt and D. de Werra, Scheduling independent multiprocessor tasks on a uniform k-processor system, Report R-92/030, Institute of Computing Science, Technical University of Poznań, Poland.
[5] J. Błażewicz, K. Ecker, G. Schmidt and J. Węglarz, *Scheduling in Computer and Manufacturing Systems* (Springer, New York, 1992).
[6] G.I. Chen and T.H. Lai, Preemptive scheduling of independent jobs on a hypercube, *IPL* 28 (1988) 201–206.
[7] E.G. Coffman Jr., *Computer and Job-Shop Scheduling Theory* (Wiley, New York, 1976).
[8] M. Dal Cin and E. Dilger, On diagnosibility of self-testing multimicroprocessor systems, *Microprocessing and Microprogramming* 7 (1981) 177–184.
[9] E. Gehringer, D. Siewiorek and Z. Segall, *Parallel Processing, The Cm* Experience* (Digital Press, 1987).
[10] J. Du and J.Y-T. Leung, Complexity of scheduling parallel task systems, *SIAM J. Discrete Math.* 12 (1989) 473–487.
[11] T. Gonzalez and S. Sahni, Preemptive scheduling of uniform processor systems, *J. ACM* 25 (1978) 92–101.
[12] E.L. Lawler, Recent results in theory of machine scheduling, in: Bachem et al. eds., *Mathematical Programming: The State of Art* (Springer, Berlin, 1983) 200–234.
[13] G. Schmidt, Scheduling on semi-identical processors, Z. Oper. Res. Theory 28 (1984) 153–162.
[14] C. Seitz, The cosmic cube, *Comm. ACM* (1) (1985).
[15] X. Shen and E.M. Reingold, Scheduling on a hypercube, *Informat. Processing Lett.* 40 (6) (1991) 323–328.
[16] J.A. Stankovic and K. Ramamritham, *Hard Real-Time Systems* (IEEE Computer Soc., Washington, 1988).