RAIRO Operations Research
RAIRO Oper. Res. **36** (2002) 37-51
DOI: 10.1051/ro:2002004

SCHEDULING MULTIPROCESSOR TASKS ON TWO PARALLEL PROCESSORS *

Jacek Błażewicz**,[2], Paolo Dell'Olmo[1] and Maciej Drozdowski[2]

Communicated by Bernard Lemaire

Abstract. In this work scheduling multiprocessor tasks on two parallel identical processors is considered. Multiprocessor tasks can be executed by more than one processor at the same moment of time. We analyze scheduling unit execution time and preemptable tasks to minimize schedule length and maximum lateness. Cases with ready times, due-dates and precedence constraints are discussed.

Keywords: Parallel processing, deterministic scheduling, multiprocessor tasks, coscheduling, gang scheduling, concurrent task systems.

1. Introduction

New parallel architecture and software systems are proposed nowadays. Efficiency is a crucial element of a parallel processing system. This is the reason for

Received November, 2000.

* *This paper stems from the summer school ODIP'2000 (Ordonnancement Déterministe pour l'Informatique et la Production), organized at the CNRS Center of Aussois, in September 2000, by Alix Munier, Stéphane Dauzère Pérès and Philip Chrétienne.*

** *The research has been partially supported by a KBN grant.*

[1] Dipartimento di Statistica, Probabilita' e Statistiche Applicate Universita' di Roma "La Sapienza", Piazzale Aldo Moro 5, 00185 Roma, Italy; and Istituto di Analisi dei Sistemi ed Informatica C.N.R., Viale Manzoni 30, 00185 Roma, Italy.

[2] Instytut Informatyki, Politechnika Poznańska, ul. Piotrowo 3a, 60-965 Poznań, Poland; e-mail: blazewic@sol.put.poznan.pl

a demand for efficient scheduling algorithms. In this work, we consider two processor parallel computer systems. Recently, two processor computer systems are becoming increasingly popular and are offered by many vendors even for low cost platforms (*e.g.* PCs).

In this work we assume that applications are multiprocessor tasks. This means that some of them may require two processors at the same moment of time. Examples of multiprocessor tasks include self-testing of processors by each-other [20], scheduling file transfers [9, 21] which simultaneously require the sender and the receiver. For efficiency reasons multiprocessor tasks are also favorable in parallel systems with time-sharing [15,27]. Threads of the same parallel program should be executed in the same time quantum on many processors rather than be scattered in various time quanta on different processors. In such systems multiprocessor tasks are called *coscheduled* or *gang-scheduled* tasks. An extensive description of the multiprocessor task concept can be found in [3, 4, 11, 26].

Now, we will formulate the considered deterministic scheduling problem. The computer system consists of two identical processors P_1 and P_2. Task set $\mathcal{T}$ is composed of two subsets: tasks requiring only one processor $\mathcal{T}^1 = \{T_1^1, \ldots, T_{n_1}^1\}$, and tasks requiring two processors at the same moment of time $\mathcal{T}^2 = \{T_1^2, \ldots, T_{n_2}^2\}$, where $n_1 + n_2 = n$. For the sake of conciseness we will denote uniprocessor tasks by 1-tasks and duoprocessor tasks by 2-tasks. In the situations where the number of required processors is meaningless we will denote tasks by T_j for $j = 1, \ldots, n$. This applies also to the rest of task parameters notation. Precedence constraints may exist among tasks, *e.g.* when the results of one task are the input for some other task. $T_i \prec T_j$ will denote that task T_i must be completed before the processing of task T_j starts. All such relations in the task system constitute precedence constraints graph (PCG). Tasks may arrive into the system at different moments of time or may have limited duration of availability for processing. Therefore, a task is characterized by a ready time and a due-date. We will denote ready times of task T_j^1 by r_j^1 for $j = 1, \ldots, n_1$, and of task T_j^2 by r_j^2 for $j = 1, \ldots, n_2$. Analogously, d_j^1 denotes due-date of a 1-task and d_j^2 due-date of a 2-task. In this work we distinguish two ways of processing tasks. Tasks have either unit execution times (UET) and are *nonpreemptable* or processing times of the tasks are different but tasks are *preemptable*. When preemptions are not allowed all tasks must be executed continuously on the same processor(s) from the beginning till the very end. Preemptability means that each task can be suspended, and restarted later (possibly on a different processor, in the case of 1-tasks) without incurring any overheads. In the former case processing time is $t_j = 1$ for $j = 1, \ldots, n$. In the latter case processing time is denoted t_j^1 for 1-tasks and t_j^2 for 2-tasks. We say that a task is *ready* when it arrived into the system and all its predecessors are finished. Let c_j denote the completion time of task T_j. Three optimality criteria will be analyzed: schedule length $C_{\max} = \max_j\{c_j\}$, maximum lateness $L_{\max} = \max_j\{c_j - d_j\}$, and the number of tasks completed after their due-dates $\sum U_j$. Mean tardiness $\sum \tau_j = \sum_{j=1}^{n} \max\{0, c_j - d_j\}$ is also mentioned in this work.

To denote considered problems the three-field notation introduced in [16, 26] will be used (*cf.* also [11]). In particular word size_j in the task field stands for the fact that tasks require several processors simultaneously.

A detailed survey of multiprocessor task scheduling can be found *e.g.* in [11]. Now, we briefly review previous results for classical and multiprocessor tasks scheduling on two parallel processors. For problem $P2 \mid p_j = 1, \text{prec} \mid C_{\max}$ of scheduling UET tasks with arbitrary precedences on two processors $O(n^2)$ algorithm was given in [8]. For the preemptive case ($P2 \mid pmtn, \text{prec} \mid C_{\max}$) an $O(n^2)$ algorithm was proposed in [25]. For problem $P2 \mid \text{size}_j, p_j = 1, \text{prec} \mid C_{\max}$ a solution based on a reduction to problem $P2 \mid p_j = 1, \text{prec} \mid C_{\max}$ was given in [23]. In [22] problem $P2 \mid \text{size}_j, p_j = 1 \mid L_{\max}$ is solved using the earliest due-date (EDD) rule. In [12] problem $P2 \mid \text{size}_j, pmtn, r_j \mid C_{\max}$ is solved by an $O(n^2)$ algorithm similar to the one proposed in [25]. On the other hand, problems $P2|\text{size}_j|C_{\max}$ [13, 19], $P3|\text{size}_j, p_j = 1, \text{chain}|C_{\max}$ [5], $P|\text{size}_j, p_j = 1|C_{\max}$ [3] are **NP**-hard. Yet, problems $Pm|\text{size}_j, p_j = 1|C_{\max}$ for any fixed m and problem $P|\text{size}_j, p_j = 1|C_{\max}$ when $\text{size}_j \in \{1, \ldots, \Delta\}$ can be solved in $O(n)$ time by use of integer linear programming with a fixed number of variables [3]. In [7] it was shown that the following problems are solvable in polynomial time: $P2|\text{size}_j, p_j = 1, r_j|\sum C_j$, $P2|\text{size}_j, p_j = 1|\sum w_j C_j$, $P2|\text{size}_j, p_j = 1|\sum U_j$, $P2|\text{size}_j, p_j = 1|\sum \tau_j$. On the other hand, problems $P2|\text{size}_j, p_j = 1, r_j, \text{chain}|C_{\max}$, $P2|\text{size}_j, p_j = 1, r_j, sp - \text{graph}|\sum C_j$, ($sp -$ graph is series-parallel graph) are **NP**-hard in the strong sense [7].

Another class of multiprocessor task scheduling problems is concerned with scheduling on dedicated processors. This kind of problems has been analyzed *e.g.* in [2, 6, 17]. We do not analyze dedicated processors in this work.

Further organization of the work is as follows. In Section 2 we consider scheduling UET tasks, and in Section 3 preemptive scheduling is analyzed. Conclusions are presented in the last section.

2. Scheduling UET tasks

In this section we study scheduling unit execution time multiprocessor tasks on two processors.

$P2 \mid \text{size}_j, p_j = 1, r_j \mid C_{\max}$

Here, we make a remark on applying the algorithm for problem $P2|\text{size}_j, p_j = 1, r_j|C_{\max}$. The algorithm with complexity $O(n \log n)$ was given in [22] by showing equivalence to problem $P2|\text{size}_j, p_j = 1|L_{\max}$. Yet, problem $P2|\text{size}_j, p_j = 1, r_j|C_{\max}$ can be solved on-line (*i.e.* without knowledge about future task arrivals) in $O(n)$ time because sorting the tasks according to the release times is done by the process issuing tasks to the computer system. Thus, 2-tasks should be executed as soon as possible, and 1-tasks in the remaining time intervals. This algorithm can be even expanded to work with the intervals of processor unavailability.

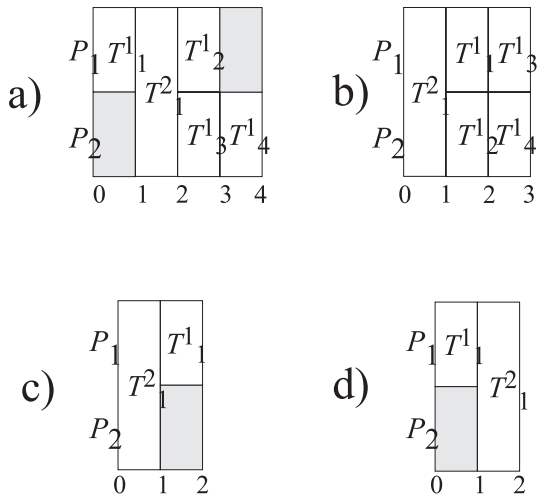

FIGURE 1. Lemma 1. Nonexistence of on-line optimization algorithm for $P2 \mid \text{size}_j, p_j = 1, r_j \mid L_{\max}$.

$P2 \mid \text{size}_j, p_j = 1, r_j \mid L_{\max}$

In this section we show that no optimization algorithm (*i.e.* delivering optimal schedules) can be run on-line.

Lemma 1. *There is no on-line optimization algorithm for problem* $P2 \mid \text{size}_j, p_j = 1, r_j \mid L_{\max}$.

Proof. Consider an example. $d^1_1 = 1, d^2_1 = 2, r^1_1 = r^2_1 = 0$. T^1_1 and T^2_1 cannot be executed in parallel. Let us consider two scenarios. Suppose T^1_1 is executed first and completes by 1. Then, at $r^1_2 = r^1_3 = r^1_4 = 1$ three new 1-tasks arrive which have $d^1_2 = d^1_3 = d^1_4 = 2$. T^2_1 and the 1-tasks cannot be processed simultaneously. Independently of the sequence for the four tasks $L_{\max} = 2$ (Fig. 1a). However, were T^2_1 executed first and completed by 1, T^1_1 and T^1_2 could be completed by 2 and T^1_3, T^1_4 by 3. Thus, the optimality criterion would be $L_{\max} = 1$ (Fig. 1b). Now analyze a different scenario. T^2_1 is executed first and completes by 1. No more tasks arrive at 1. T^1_1 is completed at 2. This gives $L_{\max} = 1$ (Fig. 1c). However, were T^1_1 processed first $L_{\max}$ could be reduced to $L_{\max} = 0$ (Fig. 1d). We conclude that whatever decision based on the available information is made, a scenario is possible which leads to unoptimality of the previous decision. □

$P2 \mid \text{size}_j, p_j = 1, \text{tree} \mid C_{\max}$

Now we examine UET tasks with tree-like PCG scheduled with the objective of minimizing schedule length. For a more general problem $P2 \mid \text{size}_j, p_j = 1,$

prec | $C_{\max}$ a solution based on a reduction to problem $P2 \mid p_j = 1, \text{prec} \mid C_{\max}$ was given in [23]. Here we improve the above results by reducing the computational complexity of the algorithm for the case of trees. To solve our problem we apply an algorithm for problem $P \mid p_j = 1, \text{tree} \mid C_{\max}$ [18]. The algorithm proposed in [18] builds level schedules, *i.e.* tasks are executed in the order of their nonincreasing levels. In our case, however, processing times of 2-tasks is ignored. The *level* of task T_j (denoted $l(T_j)$) is the length of the longest path of 1-tasks starting from the considered task plus one. Thus, a 1-task without successors has level one. Our algorithm can be summarized in the following steps:

Algorithm 1
1: calculate levels of 1-tasks;
2: **while** $\mathcal{T} \neq \emptyset$ **do**
begin
2.1: **while** there are ready 2-tasks execute them;
2.2: **if** two 1-tasks are ready **then** schedule two highest level ready 1-tasks
2.3: **else** schedule the single 1-task if there is any;
2.4: remove the scheduled tasks from $\mathcal{T}$
end;

Lemma 2. *Algorithm 1 solves problem* $P2 \mid \text{size}_j, p_j = 1, \text{tree} \mid C_{\max}$ *in* $O(n)$ *time.*

Proof. Note, that the schedule is feasible because in lines 2.1–2.3 only ready tasks are executed. A schedule for our problem can be non-optimal only when unnecessary idle time appears. Since 2-tasks introduce no idle time, the schedule is non-optimal only if a single 1-task is executed. Suppose that at time unit τ an idle time appears, and task T_j^1 is executed alone. Since a 1-task is executed alone only when the unfinished tasks are its successors this idle time is unavoidable.

Note, that each task is considered four times: 1) while calculating levels, 2) while notifying that one of its predecessors is finished - this requires $O(n)$ because PCG is a tree; 3) while putting it on the list of ready 1- or 2-tasks, 4) while executing it. Hence, the complexity of the algorithm is $O(n)$. □

Though 1-tasks are scheduled according to their levels the whole schedule built by Algorithm 1 is not strictly a level schedule. This is demonstrated in the following example.

Example 1. The PCG is presented in Figure 2a. The wide boxes are 2-tasks, the circles are 1-tasks. An example level schedule is depicted in Figure 2b, while the (optimal) schedule built by our algorithm is in Figure 2c.

$P2|\text{size}_j, p_j = 1, \text{prec}| \sum U_j$

In this section we show that the problem with precedence constraints, and the number of late tasks criterion is computationally hard.

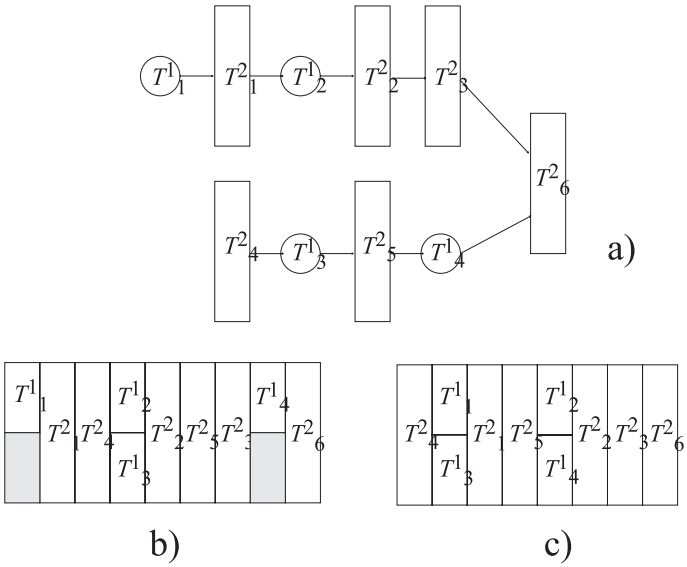

FIGURE 2. Example 1. a) PCG, b) level schedule, c) schedule built by our algorithm for $P2 \mid \text{size}_j, p_j = 1, \text{tree} \mid C_{\max}$.

Lemma 3. *Problem* $P2|\text{size}_j, p_j = 1, \text{prec}| \sum U_j$ *is* **NP***-hard.*

Proof. The proof has some similarity to the one presented in [14]. The problem is obviously in **NP**. We will show polynomial transformation from the CLIQUE problem defined as follows:

INSTANCE: A graph $G = (V, E)$ and positive integer $k \leq |V|$. Without loss of generality we assume that k is even.

QUESTION: Does G contain a clique of size k or more, that is, a subset $V' \subseteq V$ such that $|V'| \geq k$ and every two vertices in V' are connected by an edge in E?

The transformation of the CLIQUE to our problem is as follows. Set $\mathcal{T}$ has $n = |V|+|E|$ tasks in two subsets X and Y. Tasks in X correspond to vertices of G. All tasks in X are 1-tasks and have deadlines $d_j^1 = \lceil \frac{V}{2} \rceil + |E| + 1$ for $j = 1, \ldots, |V|$. Tasks in Y are 2-tasks corresponding to edges of G. Their deadlines are $d_j^2 = k^2/2$ for $j = 1, \ldots, |E|$. If vertex i in G is incident with edge j, then $T_i^1 \prec T_j^2$. We ask whether a schedule with $\sum U_j \leq |E| - k(k-1)/2$ exist.

Observe that only tasks in Y can be late. Suppose the answer to the CLIQUE problem is positive. Then a feasible schedule with the required $\sum U_j$ may have the following form: k 1-tasks corresponding to k vertices of clique V' are executed in interval $[0, k/2]$. $k(k-1)/2$ 2-tasks corresponding to the edges joining the k vertices of V' are scheduled in the interval $[k/2, k^2/2]$. The number of late tasks is $|E| - k(k-1)/2$ as required.

Assume that the answer to our scheduling problem is positive. Then at least $k(k-1)/2$ tasks from set Y were completed before their deadlines. This leaves space in interval $[0, k/2]$ for processing at most k tasks corresponding to the vertices. The 2-tasks which are not late (corresponding to edges in G) must have their predecessors completed (corresponding to the vertices in G). Thus, there is a subgraph of G which has $k(k-1)/2$ edges joining k vertices. Hence, a positive answer to the CLIQUE problem.

In the above proof we assumed k even. In the case of odd k the transformation should be augmented by task $T^1_{|V|+|E|+1}$ with $d^1_{|V|+|E|+1} = \lceil \frac{V}{2} \rceil + |E| + 1$ which precedes all tasks in set Y. The deadlines of tasks in set Y should be $d^2_j = (k^2+1)/2$ for $j = 1, \ldots, |E|$. □

3. Preemptive scheduling

In this section we assume that processing times are arbitrary but tasks are preemptable.

$P2 \mid \text{size}_j, pmtn, r_j \mid L_{\max}$

The problem of preemptive scheduling multiprocessor tasks with ready times for $L_{\max}$ criterion can be solved in polynomial time using linear programming. Observe that tasks appear in the system at ready times, disappear at respective due-dates increased by $L_{\max}$. The above events create bounds for executing tasks, yet they are not necessarily equivalent with starting or completing a task. Between ready times and due-dates increased by $L_{\max}$ the set of the tasks admissible for execution remains constant. Since the moments at which tasks should disappear from the system change with changing value of $L_{\max}$, also the set of tasks present between two consecutive events (ready times and due-dates plus $L_{\max}$) changes with $L_{\max}$. The set of tasks present between two events changes when $r_i = d_j + L_{\max}$ for some pair of tasks T_i, T_j. Thus, $\binom{n}{2}$ intervals of $L_{\max}$ value can be identified where the set of tasks present in the system between two consecutive events is the same. Assume, that the optimal schedule has $L_l \leq L_{\max} \leq L_{l+1}$ where $[L_l, L_{l+1}]$ is one of the above defined intervals of $L_{\max}$ values. If some event is a release of a task then let e_i $(i = 1, \ldots, 2n)$ denote the time moment at which the event takes place. When the event is related to a due-date then let e_i be equal to the due-date (without $L_{\max}$). With each event we associate function g_i $(i = 1, \ldots, 2n)$ equal 1 if the event is a due-date, and equal 0 otherwise. Let b^k_j $(j = 1, \ldots, n_k,\ k = 1, 2)$ denote the index of the event at which task T^k_j appears in the system, and f^k_j the index of the event when this task disappears from the system. Thus, task T^k_j can be executed in the interval $[e_{b^k_j} + g_{b^k_j} L_{\max}, e_{f^k_j} + g_{f^k_j} L_{\max}]$. Let $(i, i+1)$ denote the interval between a pair of consecutive events i and $i+1$. We will

denote by x_{ij} the amount of time 2-task T_j^2 is executed in $(i, i+1)$. Analogously, y_{ij} denotes the amount of processing 1-task T_j^1 receives in $(i, i+1)$. Our problem can be formulated as a linear program:

minimize $L_{\max}$

subject to

$$\sum_{i=b_j^2}^{f_j^2-1} x_{ij} \geq t_j^2 \quad \text{for} \quad j = 1, \ldots, n_2 \tag{1}$$

$$\sum_{i=b_j^1}^{f_j^1-1} y_{ij} \geq t_j^1 \quad \text{for} \quad j = 1, \ldots, n_1 \tag{2}$$

$$\sum_{j=1}^{n_2} x_{ij} + y_{ik} \leq e_{i+1} - e_i + L_{\max}(g_{i+1} - g_i) \quad \text{for} \quad k = 1, \ldots, n_1,\ i = 1, \ldots, 2n \tag{3}$$

$$\sum_{j=1}^{n_2} x_{ij} + \sum_{j=1}^{n_1} \frac{y_{ij}}{2} \leq e_{i+1} - e_i + L_{\max}(g_{i+1} - g_i) \text{ for } i = 1, \ldots, 2n \tag{4}$$

$$L_l \leq L_{\max} \leq L_{l+1} \tag{5}$$

$$x_{ij} \geq 0 \quad \text{for } i = b_j^2, \ldots, f_j^2 - 1, j = 1, \ldots, n_2 \tag{6}$$

$$x_{ij} = 0 \quad \text{for } i = 1, \ldots, b_j^2 - 1, f_j^2, \ldots, 2n, j = 1, \ldots, n_2 \tag{7}$$

$$y_{ij} \geq 0 \quad \text{for } i = b_j^1, \ldots, f_j^1 - 1, j = 1, \ldots, n_1 \tag{8}$$

$$y_{ij} = 0 \quad \text{for } i = 1, \ldots, b_j^1 - 1, f_j^1, \ldots, 2n, j = 1, \ldots, n_1. \tag{9}$$

In the above formulation equations (1, 2) guarantee that tasks are fully executed. Equations (3) guarantee that the longest 1-task in interval $(i, i+1)$ is not executed after event $i+1$. Equations (4) ensure that all tasks assigned to interval $(i, i+1)$ fit in it. Equation (5) guarantees that the set of tasks present in the system does not change between the events. Equations and inequalities (6–9) prevent tasks execution before their ready times and after their due-dates plus $L_{\max}$. The above linear program has $O(n^2)$ constraints and $O(n^2)$ variables, can be formulated and solved in polynomial time. When a feasible solution exists one may try an interval of smaller $L_{\max}$. And *vice versa*, when a feasible solution does not exist, one should try higher interval of $L_{\max}$ values. Thus, by binary search over intervals of $L_{\max}$ value the optimum can be found in $O(\log n)$ calls to linear programming procedure. We illustrate this method with an example.

Example 2. Consider an instance of $P2 \mid \text{size}_j, pmtn, r_j \mid L_{\max}$: $n = 4$, task data are as follows:

tasks	T_1^2	T_2^2	T_1^1	T_2^1	T_3^1
due dates	$d_1^2 = 1$	$d_2^2 = 3$	$d_1^1 = 2$	$d_2^1 = 3$	$d_3^1 = 4$
ready times	$r_1^2 = 0$	$r_2^2 = 1$	$r_1^1 = 0$	$r_2^1 = 1$	$r_3^1 = 2$
processing times	$t_1^2 = 1$	$t_2^2 = 2$	$t_1^1 = 2$	$t_2^1 = 1$	$t_3^1 = 1$

Equations $r_i = d_j + L_{\max}$ for some T_i, T_j yield six values of $L_{\max}$: -4, -3, -2, -1, 0, 1. Therefore, the optimal value of $L_{\max}$ is in one of seven intervals: $(-\infty, -4]$, $[-4, -3]$, $[-3, -2]$, $[-2, -1]$, $[-1, 0]$, $[0, 1]$, $[1, \infty)$. We should start with interval $[-2, -1]$ as the central one. However, from observation that $r_1^1 + t_1^1 \leq d_1^1 + L_{\max}$ we have that $L_{\max} \geq 0$. Therefore, we skip formulation of the linear program. We also skip the formulation of the linear program for interval $[0, 1]$ of $L_{\max}$, because even for $L_{\max} = 1$ tasks T_1^2, T_2^2, T_1^1 cannot be executed feasibly. Thus, we consider interval $[1, \infty]$ of $L_{\max}$. For these values of $L_{\max}$ six intervals between events can be distinguished: $[0, 1], [1, 2], [2, d_1^2 + L_{\max}], [d_1^2 + L_{\max}, d_1^1 + L_{\max}], [d_1^1 + L_{\max}, d_2^1 + L_{\max}], [d_2^1 + L_{\max}, d_3^1 + L_{\max}]$. For the simplicity of presentation we drop variables set to 0 according to equations (7, 9). The linear program is:
minimize $L_{\max}$
subject to

$$
\begin{array}{ll}
x_{11} + x_{21} + x_{31} \geq 1 & x_{22} + x_{32} + x_{42} + x_{52} \geq 2 \\
y_{11} + y_{21} + y_{31} + y_{41} \geq 2 & y_{22} + y_{32} + y_{42} + y_{52} \geq 1 \\
y_{33} + y_{43} + y_{53} + y_{63} \geq 1 & \\
x_{11} + y_{11} \leq 1 & \\
x_{21} + x_{22} + y_{21} \leq 1 & x_{21} + x_{22} + y_{22} \leq 1 \\
x_{21} + x_{22} + \frac{1}{2}(y_{21} + y_{21}) \leq 1 & \\
x_{31} + x_{32} + y_{31} \leq L_{\max} - 1 & x_{31} + x_{32} + y_{32} \leq L_{\max} - 1 \\
x_{31} + x_{32} + y_{33} \leq L_{\max} - 1 & x_{31} + x_{32} + \frac{1}{2}(y_{31} + y_{32} + y_{33}) \leq L_{\max} - 1 \\
x_{42} + y_{41} \leq 1 & x_{42} + y_{42} \leq 1 \\
x_{42} + y_{43} \leq 1 & x_{42} + \frac{1}{2}(y_{41} + y_{42} + y_{43}) \leq 1 \\
x_{52} + y_{52} \leq 1 & x_{52} + y_{53} \leq 1 \\
x_{52} + \frac{1}{2}(y_{52} + y_{53}) \leq 1 & y_{63} \leq 1 \\
L_{\max} \geq 1 & .
\end{array}
$$

Optimal solutions is $x_{11} = x_{42} = x_{52} = 1$, $y_{21} = y_{22} = y_{31} = y_{33} = 1, L_{\max} = 2$, other variables are 0. The corresponding schedule is shown in Figure 3.

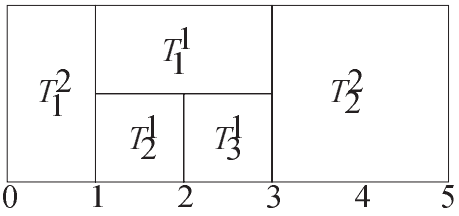

FIGURE 3. Optimal schedule for Example 2.

$P2 \mid \text{size}_j, pmtn, \text{prec} \mid C_{\max}$

The problem of preemptive scheduling multiprocessor tasks with precedence constraints on two processors for schedule length criterion can be solved analogously to the method given in [23] for $P2 \mid \text{size}_j, p_j = 1, \text{prec} \mid C_{\max}$. Namely, PCG G is replaced by its transitive closure G'. Next, 2-tasks are removed from G'. The remaining 1-tasks are scheduled according to their levels by the algorithm for $P2 \mid pmtn, \text{prec} \mid C_{\max}$ [25] in time $O(n^2)$. The *level* of a preemptable task is defined as the length of the longest path constructed of 1-tasks, starting at the considered task and including it. Finally, 2-tasks are inserted into the 1-tasks schedule as soon as their predecessors are completed. We will call this procedure **Algorithm 3**. Algorithm 3 has complexity $O(n^2 + \min\{nk, n^{2.376}\})$ because transitive closure can be built in $O(\min\{nk, n^{2.376}\})$ time, where k is the number of edges in PCG. This complexity is a result of applying the faster of the two methods: depth-first-search from each node of PCG requiring $O(nk)$ time, or a procedure proposed in [10] for finding transitive closure in $O(n^{2.376})$. In the further discussion we demonstrate that explicit calculation of the transitive closure is not necessary.

We propose an algorithm which combines Algorithm 1 and the algorithm for problem $P2 \mid pmtn, \text{prec} \mid C_{\max}$ [25]. The latter algorithm uses the concepts of processing capacities, and task level. *Processing capacity* is a number from range $[0, 1]$ which can be understood as a fraction of a processor obtained by a 1-task. Levels of 1-tasks are calculated as before. The level of a 2-task is equal to the highest level of its successor. If there is no 1-task successor then the level for the 2-task is zero. In the algorithm we denote by

Q – a set of ready 1-tasks;
$\overline{\beta} = [\beta_1, \ldots, \beta_{n_1}]$ – a vector of processing capacities;
τ – the length of the current processing capacities assignment.

Algorithm 4

1: calculate levels of 1-tasks; calculate set Q of ready 1-tasks;
order tasks in Q according to their nonincreasing level;
2: **while** $\mathcal{T} \neq \emptyset$ **do**
begin
2.1: **while** there are ready 2-tasks execute them;

2.2: remove the scheduled 2-tasks from $\mathcal{T}$;
2.3: CAPACITIES$(Q, \overline{\beta})$;
2.4: calculate times:
if $\exists_{T_j^1, T_{j+1}^1 \in Q} l(T_j^1) > l(T_{j+1}^1)$ **then**
$\tau' := \min_{T_j^1, T_{j+1}^1 \in Q} \{ \frac{l(T_j^1) - l(T_{j+1}^1)}{\beta_j - \beta_{j+1}} : \beta_j \neq \beta_{j+1}, l(T_j^1) > l(T_{j+1}^1) \}$
else $\tau' := \infty$
(* the shortest time required for two tasks T_j^1, T_{j+1}^1 with different levels to become equal *);
$\tau'' := \min_{T_j^1 \in Q} \{ \frac{t_j^1}{\beta_j} : \beta_j > 0 \}$; (*time to the earliest completion of any 1-task*)
2.5: $\tau := \min\{\tau', \tau''\}$;
2.6: **for** $T_j^1 \in Q$ **do** schedule $\tau\beta_j$ piece of task T_j^1 in interval of length τ according to the algorithm for $P \mid pmtn \mid C_{\max}$ [24];
2.7: $t_j^1 := t_j^1 - \tau\beta_j$, $l(T_j^1) := l(T_j^1) - \tau\beta_j$ for $T_j^1 \in Q$;
2.8: **for** $T_j^1 \in Q$ **do if** $t_j^1 = 0$ **then** $\mathcal{T} = \mathcal{T} - \{T_j^1\}$;
2.9: merge 1-tasks which became ready with Q in nonincreasing level order;
end; (* of the algorithm *)

procedure CAPACITIES(**in:**X;**out:**$\overline{\beta}$); (* X - a set of 1-tasks *)
begin
3: $\overline{\beta} := \overline{0}$; $avail := 2$; (* $avail$ is the number of free processors *)
4: **while** $avail > 0$ **and** $\mid X \mid > 0$ **do**
begin
4.1: construct set Y of tasks in X with the highest level;
4.2: **if** $\mid Y \mid > avail$ **then**
begin
4.2.1: $\beta_j := \frac{avail}{\mid Y \mid}$, for $T_j^1 \in Y$; $avail$:=0;
end
else (*at most $avail$ processors can be used by tasks in Y*)
begin
4.2.2: $\beta_j := 1$, for $T_j \in Y$; $avail := avail - \mid Y \mid$;
end;
4.3: $X := X - Y$;
end; (* of **while** loop *)
end; (* of procedure CAPACITIES *)

Lemma 4. *Algorithm 4 solves problem* $P2 \mid \text{size}_j, pmtn, \text{prec} \mid C_{\max}$ *in* $O(n^2)$ *time.*

Proof. In lines 2.1 and 2.6 only ready tasks are executed, therefore precedence constraints are observed. In line 2.1 2-tasks are fully processed. A 1-task stops receiving processor in line 2.6 only if its remaining processing time is 0 (line 2.8). The τ unit long schedule built in line 2.6 is feasible (*cf.* [24]) because $\tau\beta_j \leq \tau$, and $\sum_{T_j^1 \in A} \beta_j\tau + \sum_{T_j^1 \in Q-A} \beta_j\tau = \tau(\mid A \mid + \frac{m - \mid A \mid}{\mid Q - A \mid} \mid Q - A \mid) \leq m\tau$, where A is the set of tasks receiving processing capacity in line 4.2.2 of procedure CAPACITIES. Hence, schedules are feasible.

In the schedules for the problem two types of intervals can be distinguished: intervals with 2-tasks only (which will be called 2-blocks), and intervals with 1-tasks or idle times (called 1-blocks). The proof of optimality boils down to showing that the total length of 1-blocks is the same in Algorithm 4 and Algorithm 3.

Observe that in lines 2.3–2.8, 1-blocks are built by a rephrased algorithm for $P \mid pmtn, \text{prec} \mid C_{\max}$ [4, 25]. Thus, Algorithm 3 and Algorithm 4 use the same method based on levels to schedule 1-tasks. Now analyze levels. In Algorithm 3, 2-tasks are not present in PCG G'. Precedence constraints involving 2-tasks in the original PCG G are preserved in G' by additional transitive arcs. The level of a 1-task is equal to its processing time plus the highest level among its successors added by the transitive arcs and its original 1-task successors in G. Also in Algorithm 4 processing times of 2-tasks do not contribute to the level of 1-tasks. The level of a 1-task is its processing time plus the highest level of its original 1-task successors and 2-task successors. The level of a 2-task is equal to the highest level of its 1-task successor. Thus, levels of 1-tasks in Algorithm 3 and Algorithm 4 are the same and the schedules in 1-blocks have the same lengths.

Finally, we analyze complexity of Algorithm 4. Line 1 requires $O(n^2)$ to calculate levels, select ready tasks and sort them. Lines 2.1, 2.2 over all algorithm run need $O(n^2)$ time because one must verify completion of 2-task predecessors. Line 2.3 requires $O(n)$ time because loop 4 in procedure CAPACITIES can be executed at most twice, and line 4.2.1 requires $O(n_1)$ time. Observe that by calculation of $\overline{\beta}$ when two 1-tasks achieve the same level they remain equal until their completion. Therefore, line 2.4 is executed $O(n_1)$ times because two different levels may become equal at most $n_1 - 1$ times and $O(n_1)$ times some task can be finished. By the same token loop 2 is repeated $O(n_1)$ times. Lines 2.5–2.9 together require $O(n_1)$ time. Thus, total complexity of Algorithm 4 is $O(n^2)$. □

We finish this section with an example.

Example 3. Consider an instance of $P2 \mid \text{size}_j, pmtn, \text{prec} \mid C_{\max}$: $n_2 = 5$, $n_1 = 8$ task processing times are as follows: $t_1^2 = 2, t_2^2 = 5, t_3^2 = 2, t_4^2 = 4, t_5^2 = 3$, $t_1^1 = 3, t_2^1 = 4, t_3^1 = 1, t_4^1 = 6, t_5^1 = 3, t_6^1 = 5, t_7^1 = 4, t_8^1 = 3$. PCG is presented in Figure 4a. In Figure 4a 1-tasks are represented by circles, 2-tasks are represented by rectangles. Each task in Figure 4a is labeled with processing time and level calculated by Algorithm 4.

Algorithm 3 builds transitive closure G' of the PCG. Then, 2-tasks are removed from the graph. In Figure 4b G' is shown (for clarity of the presentation, only non-transitive arcs are depicted). Each task in Figure 4b is labeled with processing time and level calculated by Algorithm 3. Observe that levels of 1-tasks are identical in Algorithm 3 and Algorithm 4. Algorithm 3 builds optimal schedule for 1-tasks in PCG G' (Fig. 4c). Later, 2-tasks are inserted into this schedule to obtain the optimal schedule for all tasks (Fig. 4d). Algorithm 4 builds the same optimal schedule as Algorithm 3 because at each point in time 1-tasks have the same levels, *cf.* Figure 4d.

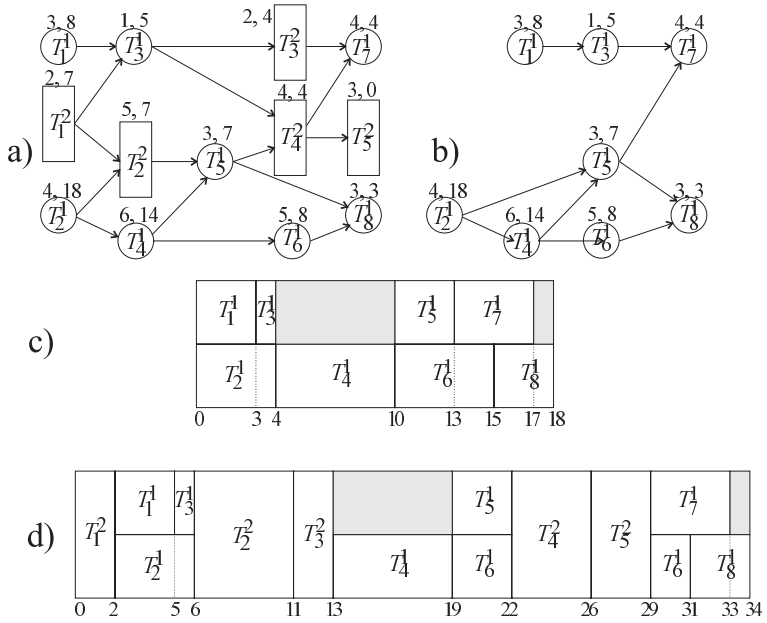

FIGURE 4. Example 3. a) PCG, b) PCG G' with 1-tasks only, c) optimal schedule for 1-tasks only, d) optimal schedule for all tasks.

4. Conclusions

In this work we analyzed scheduling multiprocessor tasks on two identical processors. The following results were obtained:

Problem	Result
Unit execution time	
$P2 \mid \text{size}_j, p_j = 1, r_j \mid C_{\max}$	$O(n)$ - on line
$P2 \mid \text{size}_j, p_j = 1, r_j \mid L_{\max}$	no on-line algorithm exist
$P2 \mid \text{size}_j, p_j = 1, \text{tree} \mid C_{\max}$	$O(n)$
$P2 \mid \text{size}_j, p_j = 1, r_j, \text{prec} \mid \sum U_j$	**NP**-hard
Preemptive scheduling	
$P2 \mid \text{size}_j, pmtn, r_j \mid L_{\max}$	linear programming
$P2 \mid \text{size}_j, pmtn, \text{prec} \mid C_{\max}$	$O(n^2)$

Further research may include, for example, analysis of problems where both release times and due-dates are present, *e.g.* reduction of problem $P2 \mid \text{size}_j, pmtn, r_j \mid L_{\max}$ complexity below the complexity of linear programming. Recently we

learnt that a solution for problem $P2 \mid \text{size}_j, p_j = 1, r_j \mid L_{\max}$, has been proposed [1].

References

[1] P. Baptiste and B. Schieber, Scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness, in *Proc. of the VIII Int. Workshop On Project Management and Scheduling* (2002) 55-58.

[2] J. Błażewicz, P. Dell'Olmo, M. Drozdowski and M.G. Speranza, Scheduling multiprocessor tasks on three dedicated processors. *Inform. Process. Lett.* **41** (1992) 275-280. Corrigendum: *Inform. Process. Lett.* **49** (1994) 269-270.

[3] J. Błażewicz, M. Drabowski and J. Węglarz, Scheduling multiprocessor tasks to minimize schedule length. *IEEE Trans. Comput.* **35** (1986) 389-393.

[4] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt and J. Węglarz, *Scheduling Computer and Manufacturing Processes*. Springer-Verlag, Heidelberg (1996).

[5] J. Błażewicz and Z. Liu, Scheduling multiprocessor tasks with chain constraints. *Eur. J. Oper. Res.* **94** (1996) 231-241.

[6] P. Brucker and A. Krämer, Shop scheduling problems with multiprocessor tasks and dedicated processors. *Ann. Oper. Res.* **57** (1995) 13-27.

[7] P. Brucker, S. Knust, D. Roper and Y. Zinder, Scheduling UET task systems with concurrency on two parallel identical processors. *Math. Meth. Oper. Res.* **52**, 369-387.

[8] E.G. Coffman Jr. and R.L. Graham, Optimal scheduling for two-processor systems. *Acta Informatica* **1** (1972) 200-213.

[9] E.G. Coffman Jr., M.R. Garey, D.S. Johnson and A.S. Lapaugh, Scheduling file transfers. *SIAM J. Comput.* **14** (1985) 744-780.

[10] D. Coppersmith and S. Winograd, Matrix multiplication *via* arithmetic progressions. *J. Symb. Comput.* **9** (1990) 251-280.

[11] M. Drozdowski, Scheduling multiprocessor tasks – An overview. *Eur. J. Oper. Res.* **94** (1996) 215-230.

[12] M. Drozdowski, *Selected problems of scheduling tasks in multiprocessor computer systems*. Poznań University of Technology Press, Series: Rozprawy, No. 321, Poznań (1997). See also: `http://www.cs.put.poznan.pl/~maciejd/txt/h.ps`

[13] J. Du and J.Y-T. Leung, Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.* **2** (1989) 473-487.

[14] M.R. Garey and D.S. Johnson, Scheduling tasks with nonuniform deadlies on two processors. *J. ACM* **23** (1976) 461-467.

[15] E.F. Gehringer, D.P. Siewiorek and Z. Segall, *Parallel Processing: The Cm* Experience*. Digital Press, Bedford (1987).

[16] R.L. Graham, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnoy Kan, Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math.* **5** (1979) 287-326.

[17] J.A. Hoogeveen, S.L. van de Velde and B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processors allocations. *Discrete Appl. Math.* **55** (1994) 259-272.

[18] T.C. Hu, Parallel sequencing and assembly line problems. *Oper. Res.* **9** (1961) 841-848.

[19] R.M. Karp, Reducibility among combinatorial problems, edited by R.E. Miller and J.W. Thatcher, *Complexity of Computer Computation*. Plenum Press, New York (1972) 85-104.

[20] H. Krawczyk and M. Kubale, An approximation algorithm for diagnostic test scheduling in multicomputer systems. *IEEE Trans. Comput.* **34** (1985) 869-872.

[21] M. Kubale, The complexity of scheduling independent two-processor tasks on dedicated processors. *Inform. Process. Lett.* **24** (1987) 141-147.

[22] C.Y. Lee and X. Cai, *Scheduling multiprocessor tasks without prespecified processor allocations*. Private communication (1998).
[23] E.L. Lloyd, Concurrent task systems. *Oper. Res.* **29** (1981) 189-201.
[24] R. McNaughton, Scheduling with deadlines and loss functions. *Management Sci.* **6** (1959) 1-12.
[25] R. Muntz and E.G. Coffman Jr., Preemptive scheduling of real time tasks on multiprocessor systems. *J. Assoc. Comput. Machinery* **17** (1970) 324-338.
[26] B. Veltman, B.J. Lageweg and J.K. Lenstra, Multiprocessor scheduling with communications delays. *Parallel Comput.* **16** (1990) 173-182.
[27] J. Zahorjan, E.D. Lazowska and D.L. Eager, The effect of scheduling discipline on spin overhead in shared memory parallel systems. *IEEE Trans. Parallel Distributed Systems* **2** (1991) 180-199.