BULLETIN OF THE POLISH
ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 43, No. 3, 1995

On the Complexity of Multiprocessor Task Scheduling

by

Maciej DROZDOWSKI

Presented by J. WĘGLARZ on January 13, 1995

Summary. The problem addressed here is the complexity of scheduling multiprocessor tasks, i.e. tasks which may require more than one processor simultaneously. This model seems natural for many parallel applications, however, is different than the standard approach in the scheduling theory. In this paper, we analyse both cases when the task requires a *number* of processors simultaneously, and when the task requires a *set* of processors simultaneously. In the former case the problem of preemptive scheduling is $\mathcal{NP}$-hard in general. Next, we show that the problem of non-preemptive scheduling on four processors, when tasks can be executed on any subset of processors, is solvable in pseudo-polynomial time in the absence of uni-processor tasks. Finally, the problem of non-preemptive scheduling multiprocessor tasks, requiring a set of processors simultaneously for $L_{\max}$ criterion, is shown to be strongly $\mathcal{NP}$-hard even for two processors.

1. Introduction. In classical scheduling models it is assumed that each task requires, for its processing, one processor at a time [3, 9, 12]. In systems of microprocessors, however, one often has tasks requiring several processors simultaneously (e.g. [5, 23] and many others). It is, for instance, the case of self-testing multi-microprocessor systems in which one processor is used to test other processors, or in fault-detection systems in which test signals stimulate the elements to be tested; then the outputs are analysed simultaneously [2, 14].

Novel parallel algorithms and corresponding future task systems create another branch for application of this kind of scheduling [1, 16, 17, 21]. Transferring files between computers engages two computers simultaneously [13]. We will be calling such tasks the *multiprocessor* tasks (a problem of scheduling such tasks is also called the *coscheduling* problem [1] or *gang*

Key words: complexity, scheduling, multiprocessor tasks, scheduling with prespecified processor allocations.

scheduling problem [16]). More details about practical uses of multiprocessor tasks can be found in [8].

When modelling task sets for the above applications, it is often assumed that either the *number* of simultaneously required processors is important, or that the *set* of required processors is crucial for the execution of task.

In this work we will denote scheduling problems according to the three-field scheme proposed in [18] and [10, 24]. Multiprocessor requirements of tasks will be represented by using one of the words: $size_j$, *any*, fix_j. The word $size_j$ is used when tasks require for their execution a certain fixed number of processors. The word *any* means that each task can be executed on any subset of processors, but the execution time depends on the number of assigned processors. fix_j denotes tasks requiring a fixed *set* of processors simultaneously.

Before going into further details, let us formulate the problem in a more formal way. When the *number* of simultaneously required processors is important, we can divide the set of tasks $\mathcal{T}$ 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_i^j requires exactly j arbitrary processors simultaneously during a prespecified period t_i^j of its processing. If the model *any* of requiring processors is considered, t_i^j will denote the processing time of task i executed on j processors simultaneously. We assume that the number of processors assigned to a task does not change during its processing. This model is useful both from the theoretical and practical points of view. For it is often the case that changing the number of processors used by a task during its execution, if not impossible, is very time-consuming. This is, for example, the level of a scheduler in the operating system. What is more, it enables the derivation of complexity results valid also for more sophisticated models. In the case of tasks requiring a set of processors simultaneously, we will denote by T^D the set of tasks using the set D of processors, and by T_i^D task i which uses the set D of processors. We assume that the tasks are independent, and each processor can be assigned to only one task at a time. There are two objectives considered in this work. The first one is to find the shortest feasible schedule, i.e. minimize $C_{\max} = \max_{1 \leqslant j \leqslant n}\{c_j\}$, where c_j is the completion time of the jth task, and the second one is the maximum lateness defined as follows: $L_{\max} = \min_{1 \leqslant j \leqslant n}\{c_j - d_j\}$, where d_j is the due-date of task j.

The computational complexity of the multiprocessor task scheduling has already been analysed in a number of works. In the case of tasks requiring a certain *number* of processors simultaneously, it has been shown in [15] that the problem of non-preemptive scheduling with precedence constraints consisting of chains is strongly $\boldsymbol{NP}$-hard when tasks can be executed on two processors (i.e. problem $P2 \mid size_j, chain \mid C_{\max}$). When the tasks have unit execution times, the above problem can be solved in $O(n^2)$ time for

arbitrary precedence constraints ($P2 \mid size_j, \prec, p_j = 1 \mid C_{\max}$) [20]. The independent tasks scheduling problem is strongly $\boldsymbol{NP}$-hard for five processors [15] (problem $P5 \mid size_j \mid C_{\max}$). When tasks require processors according to the model *any*, the problem of scheduling on four processors (i.e. $P4 \mid any \mid C_{\max}$) is open in that sense that it is not known whether the problem is strongly $\boldsymbol{NP}$-hard or solvable in a pseudo-polynomial time. The cases of two and three processors ($P2 \mid any \mid C_{\max}$, and $P3 \mid any \mid C_{\max}$, respectively) can be solved in pseudo-polynomial time [15]. A polynomial-time algorithm based on the integer linear programming with a limited number of variables is known for the special case of the unit execution time tasks, namely, for problem $P \mid size_j, p_j = 1 \mid C_{\max}$, where $size_j \in \{1, \ldots, k\}$, and k is bounded [5].

For the preemptive multiprocessor task scheduling and identical processors, some results have been obtained. For independent multiprocessor task systems with m fixed (m is the number of processors), the problem can be solved in the polynomial time using a linear programming formulation [5]. If there are only tasks requiring either one or k processors simultaneously, the optimal schedule can be constructed in $O(n)$ time [5]. A special case of the preemptive version of the problem is scheduling on a hypercube of processors [7, 11, 22]. An $O(nm^2 + n \log n)$ algorithm was proposed in [7] to schedule preemptively tasks requiring a number of processors which is a power of two.

When tasks require a set of processors simultaneously, nonpreemptive scheduling on two processors ($P2 \mid fix_j \mid C_{\max}$) is trivially solvable in $O(n)$ time. The case of three processors ($P3 \mid fix_j \mid C_{\max}$) is already $\boldsymbol{NP}$-hard in the strong sense [4].

In this paper, we prove that the problem of scheduling independent preemptable multiprocessor tasks on identical processors, for the maximum length criterion, is $\boldsymbol{NP}$-hard when the number of processors is not bounded. This result can be surprising when considering the above-mentioned low-order polynomial-time algorithms (e.g. [5, 6, 7, 11, 22]) for very closely related problems. Next, we show that problem $P4 \mid any \mid C_{\max}$ can be solved by a pseudopolynomial algorithm when there are no uni-processor tasks (i.e. $T^1 = \emptyset$). Finally, for tasks requiring a set of processors simultaneously, we show that the non-preemptive scheduling with due-dates is strongly $\boldsymbol{NP}$-hard even for two processors.

2. Complexity of the problem.

2.1. $P \mid size_j, pmtn, p_j = 1 \mid C_{\max}$. In this section the computational complexity for preemptive scheduling multiprocessor tasks, requiring some number of identical processors, will be established. As it has been mentioned

in the previous section, for some sub-cases of the problem, the low-order polynomial time algorithms exist. In the following theorem and proof we restrict ourselves to considering tasks of equal length. Hence, $p_j = 1$ in the notation of the problem.

THEOREM 1. *The problem of preemptive scheduling of independent unit-execution-time multiprocessor tasks on identical processors for* $C_{\max}$ *criterion (i.e. problem* $P \mid size_j, pmtn, p_j = 1 \mid C_{\max}$*) is* $\boldsymbol{NP}$*-hard.*

P r o o f. In order to show $\boldsymbol{NP}$-hardness of the above problem, we will present the polynomial-time transformation of a known $\boldsymbol{NP}$-complete problem to a decision version of our problem. As a known $\boldsymbol{NP}$-complete problem, we will use a partition problem.

PARTITION PROBLEM.

D a t a. A finite set $A = \{a_1, \ldots, a_q\}$, for each $a_i \in A$, there exists a finite size $s(a_i) \in Z^+$.

Q u e s t i o n. Does a set $A' \subset A$ satisfying $\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A - A'} s(a_i) = B$ exist?

Now, we will describe a reduction from any instance of the partition to the instance of our problem. We set the parameters of the scheduling problem as follows:

$n := q$,
$T_j^{s(a_j)}$, $j = 1, \ldots, n$ tasks are created,
$t_j^{s(a_j)} := 1$, $j = 1, \ldots, n$,
$C_{\max} := 2$,
$m := B$.

If for the partition problem the answer is positive, then tasks corresponding to elements $a_i \in A'$ are scheduled feasibly in the interval of time [0, 1], and the rest of tasks in the interval [1, 2] (cf. Fig. 1).

On the other hand, if a feasible schedule exists, then also, for the partition problem, the answer must be positive. This can be explained in the following way. In the feasible schedule there can be no idle time on any processor. This means that, in any moment of time $0 \leqslant \tau \leqslant 2$, the sum of processor requirements of tasks is equal to B. That, however, is equivalent to the existence of A' consisting of the elements a_i corresponding to the tasks executed in the instant τ. The rest of the tasks correspond to the elements of set $A - A'$. Note, that the set of simultaneously executed tasks corresponding to $A - A'$ may be absent from the schedule. This finishes our reduction and the whole proof. □

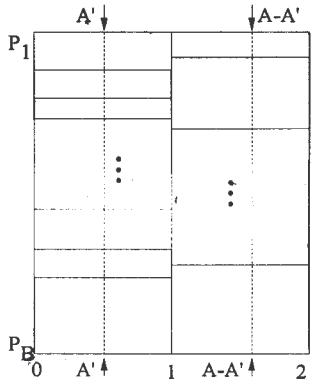

Fig. 1. A feasible schedule with $C_{\max} = 2$ for the proof of Theorem 1

From the above theorem, we conclude that our problem is $\boldsymbol{NP}$-hard when the number of processors is not fixed.

2.2. $P4 \mid any \mid C_{\max}$. In this section we show that the problem of non-preemptive scheduling multiprocessor tasks requiring processors according to the model *any* on four processors for the schedule-length optimality criterion (i.e. $P4 \mid any \mid C_{\max}$) can be solved by a pseudo-polynomial algorithm when there are no uni-processor tasks (i.e. $T^1 = \emptyset$). The above result may also shed some light on the complexity of problem $P4 \mid any \mid C_{\max}$, when uni-processor tasks are allowed. The complexity of the latter problem is considered open [15]. The cases of two and three processors ($P2 \mid any \mid C_{\max}$, and $P3 \mid any \mid C_{\max}$, respectively) are $\boldsymbol{NP}$-hard but solvable in pseudo-polynomial time [15]. For five processors ($P5 \mid size_j \mid C_{\max}$), the problem is strongly $\boldsymbol{NP}$-hard.

When there are no uni-processor tasks, all the schedules on four processors can be transformed into the schedule presented in Fig. 2a. Four-processor tasks and triple-processor tasks cannot be executed in parallel with any other tasks. Thus, without changing the length of the schedule, we can group them according to the set of used processors. Duo-processor tasks can be executed in parallel only with the duo-processor tasks requiring the two complementary processors. Thus, we can shift complementary duo-processor tasks executed in parallel so that duo-processor tasks using the same set of processors are executed together. Finally, if there are any duo-processor tasks executed alone, we can shift them, without increasing the schedule length, such that they are executed together with tasks using the same set of processors. Hence, there is no advantage in moving tasks in Fig. 2a anywhere else in the schedule. What is more, executing duo-processor tasks in another way (but without changing assignment to the processors) may only incur increasing the length of the schedule.

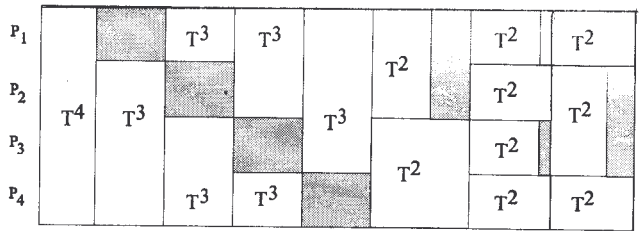

Fig. 2a. A schedule for problem $P4 \mid any \mid C_{\max}$ with all the possible ways of using processors in the absence of the uni-processor tasks

Since in the model *any* the *number* of simultaneously required processors is important for scheduling a task, we can force triple-processor tasks to use processors P_2, P_3, P_4. Next, we can force duo-processor tasks to use either processors P_1, P_2 or processors P_3, P_4. Without loss of generality, we can assume that the duo-processor tasks, which are executed longer than these duo-processor tasks requiring complementary two processors, are scheduled on processors P_3, P_4 (cf. Fig. 2b). Thus, we conclude that each schedule on four processors, in the absence of uni-processor tasks, can be transformed to the schedule of the same length with a normalized structure presented in Fig. 2b.

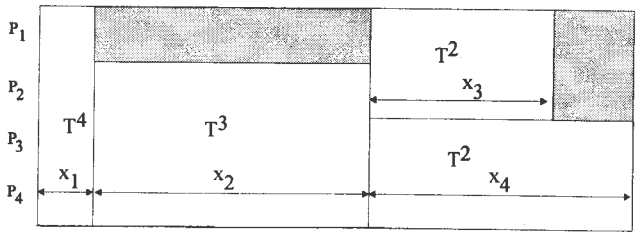

Fig. 2b. A schedule with the normalized structure for problem $P4 \mid any \mid C_{\max}$

For a schedule with the fixed structure as the one in Fig. 2b, we can devise a dynamic programming algorithm. Let us denote, in Fig. 2b, by x_1 and x_2 the sum of processing times of the tasks scheduled as the four-processor and triple-processor tasks, respectively, by x_3 and x_4 the sum of processing times of the tasks scheduled as the duo-processor tasks on processors P_1, P_2, and P_3, P_4, respectively. The dynamic programming algorithm can be based on calculating the following function

$f(j, x_1, x_2, x_3, x_4) = 1$ if and only if tasks $T_1, \ldots, T_j$ can be feasibly executed using the units of time x_1 and x_2 for the tasks scheduled as the four-processor and triple-processor tasks, respectively, units of time x_3 and x_4 for the tasks scheduled as the duo-processor tasks on processors P_1, P_2, and P_3, P_4, respectively. In the opposite case

$$f(j, x_1, x_2, x_3, x_4) = 0,\ j = 1, \ldots, n,$$

$$x_1, x_2, x_3, x_4 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}.$$

The above formulation can be rewritten to the recursive form.

$$f(1, t_1^4, 0, 0, 0) = f(1, 0, t_1^3, 0, 0) = f(1, 0, 0, t_1^2, 0) = f(1, 0, 0, 0, t_1^2) = 1,$$

otherwise

$$f(1, x_1, x_2, x_3, x_4) = 0 \quad \text{for } x_1, x_2, x_3, x_4 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}.$$

The above formulation states that the partial schedule consisting of the first task only must finish after executing the first task in one of the allowed processing modes, i.e. as a four-processor task, as a triple-processor task or as a duo-processor task.

For $j > 1$, $f(j, x_1, x_2, x_3, x_4) = 1$ if and only if one of the following cases happens

Case 1.

$$f(j-1, x_1 - t_j^4, x_2, x_3, x_4) = 1 \qquad \text{for } j = 2, \ldots, n,$$

$$x_1 = t_j^4, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}, \qquad x_2, x_3, x_4 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\},$$

which means that task T_j is scheduled in a four-processor mode (cf. Fig. 3a).

Case 2.

$$f(j-1, x_1, x_2 - t_j^3, x_3, x_4) = 1 \qquad \text{for } j = 2, \ldots, n,$$

$$x_2 = t_j^3, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}, \qquad x_1, x_3, x_4 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\},$$

which means that task T_j is scheduled in a triple-processor mode (cf. Fig. 3b).

Case 3.

$$f(j-1, x_1, x_2, x_3 - t_j^2, x_4) = 1 \qquad \text{for } j = 2, \ldots, n,$$

$$x_3 = t_j^2, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}, \qquad x_1, x_2, x_4 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\},$$

which means that task T_j is scheduled in a duo-processor mode on processors P_1, P_2 (cf. Fig. 3c).

Case 4.

$$f(j-1, x_1, x_2, x_3, x_4 - t_j^2) = 1 \qquad \text{for } j = 2, \ldots, n,$$

$$x_4 = t_j^2, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\}, \qquad x_1, x_2, x_3 = 0, \ldots, \sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\},$$

analogously to Case 3, but T_j is scheduled on processors P_3, P_4.

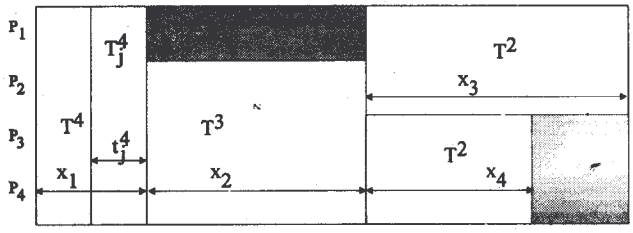

Fig. 3a

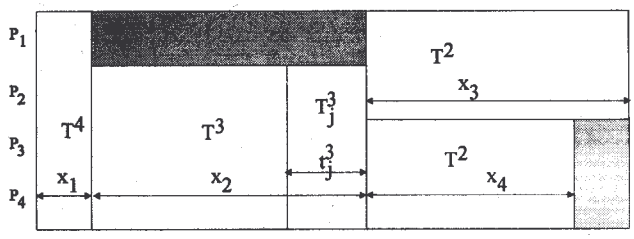

Fig. 3b

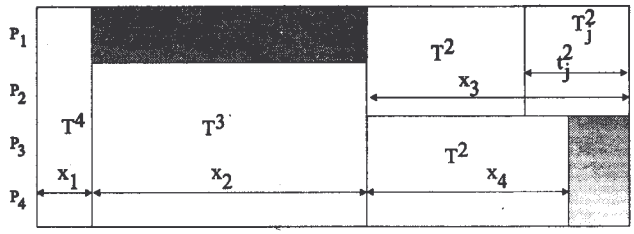

Fig. 3c

Fig. 3. A partial schedule built by a dynamic programming algorithm for problem $P4 \mid any \mid C_{\max}$ in the absence of uni-processor tasks, a — case 1, b — case 2, c — case 3

It takes time $O(n(\sum_{i=1}^{n} \min\{t_i^2, t_i^3, t_i^4\})^4)$ to calculate the values of the above function. The length of the optimal schedule is

$$C^*_{\max} = \min\{x_1 + x_2 + \max\{x_3, x_4\} \mid f(n, x_1, x_2, x_3, x_4) = 1\}.$$

The optimal schedule can be found by backtracking from the function entry for which $C^*_{\max}$ has been found, i.e. equal to one for $j = n$, and with minimal $x_1 + x_2 + \max\{x_3, x_4\}$. The correctness of the algorithm follows

from two facts. Each feasible schedule for the considered problem can be transformed, without increasing its length, to the one presented in Fig. 2b. The second fact is that the length of such a schedule does not depend on the order of tasks. However, it does depend on the values of x_1, x_2, x_3, x_4. While calculating function $f(j, x_1, x_2, x_3, x_4)$, we have analysed all the possible values of x_1, x_2, x_3, x_4. Hence, problem $P4 \mid any \mid C_{\max}$ can be solved in the pseudo-polynomial time provided $T^1 = \emptyset$,

2.3. $P2 \mid fix_j \mid L_{\max}$. The problem of scheduling non-preemptable multiprocessor tasks requiring sets of processors simultaneously is easily solvable for two processors. For three processors ($P3 \mid fix_j \mid C_{\max}$) is $\boldsymbol{NP}$-hard in the strong sense [4]. In this section we show that also problem $P2 \mid fix_j \mid L_{\max}$ is strongly $\boldsymbol{NP}$-hard. Let us remind that here T_j^D denotes task j which is the requiring set D of processors simultaneously. For example, T_j^{12} is task j, and requires processors $\{P_1, P_2\}$ simultaneously.

THEOREM 2. *The problem of non-preemptive scheduling of multiprocessor tasks, requiring a certain set of processors simultaneously for $L_{\max}$ optimality criterion even for two processors (i.e. $P2 \mid fix_j \mid L_{\max}$), is strongly $\boldsymbol{NP}$-hard.*

P r o o f . To show strong $\boldsymbol{NP}$-hardness of our scheduling problem, we will present the polynomial-time transformation from 3-partition.

3-PARTITION.

D a t a. A finite set $A = \{a_1, \ldots, a_{3q}\}$, for each $a_i \in A$, there exists a size $s(a_i) \in Z^+$ such that $\sum_{i=1}^{3q} s(a_i) = qB$, and $B/4 < s(a_i) < B/2$ for $i = 1, \ldots, 3q$.

Q u e s t i o n. Can A be partitioned into q disjoined sets $A_1, A_2, \ldots, A_q$, such that $\sum_{a_i \in A_j} s(a_i) = B$ for $j = 1, \ldots, q$?

Now, we will present the polynomial-time transformation of the 3-partition instance to the instance of our problem. The following tasks are created

$$
\begin{aligned}
& n := 5q, \\
& T_j^2, && j = 1, \ldots, q, \\
& T_j^{12}, && j = q+1, \ldots, 2q, \\
& T_j^1, && j = 2q+1, \ldots, 5q. \\
& t_j^2 := B, && j = 1, \ldots, q, \\
& t_j^{12} := B, && j = q+1, \ldots, 2q, \\
& t_j^2 := s(a_{j-2q}), && j = 2q+1, \ldots, 5q,
\end{aligned}
$$

$$d_j := \begin{cases} (2j-1)B & \text{for } j = 1,\ldots,q \\ 2(j-q)B & \text{for } j = q+1,\ldots,2q \\ 2qB & \text{for } j = 2q+1,\ldots,5q. \end{cases}$$

$L_{max} := 0,$

$m := 2.$

When the 3-partition has a positive answer, then we can build a feasible schedule with $L_{\max} = 0$ as the one presented in Fig. 4.

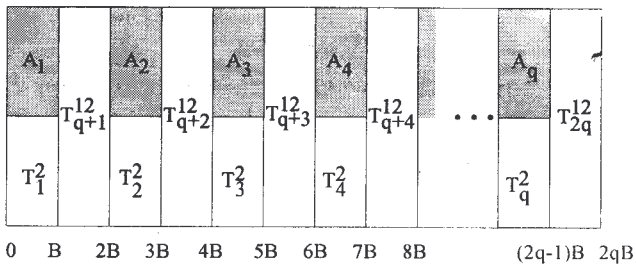

Fig. 4. A feasible schedule with $L_{\max} = 0$ for the proof of Theorem 2

On the other hand, when there exists a feasible schedule with $L_{\max} = 0$, for instance, built in the above-defined way, then this must be isomorphic with the one in Fig. 4. Note that there can be no idle time on processor P_2 in the schedule with $L_{\max} = 0$. Task T_1^2 must be executed first on P_2, and must be immediately followed by T_{q+1}^{12} in order to have $L_{\max} = 0$. After T_{q+1}^{12} task T_2^2 must follow, etc. This reasoning can be applied to the rest of tasks T_j^{12}, $j = q+2,\ldots,2q$, and T_j^2, $j = 2,\ldots,q$. This results in creating q slots of size B in which tasks T_j^1, $j = 2q+1,\ldots,5q$ must fit. Note that also on P_1 we may have no idle time to schedule T_j^1, $j = 2q+1,\ldots,5q$ before due-date $2qB$. Hence, also for 3-partition, the answer must be positive and the theorem is proved. □

3. Conclusions and future work. We have proved that problem $P \mid size_j, pmtn, p_j = 1 \mid C_{\max}$ is $\boldsymbol{NP}$-hard in the ordinary sense, problem $P2 \mid fix_j \mid L_{\max}$ is strongly $\boldsymbol{NP}$-hard, and that $P4 \mid any \mid C_{\max}$ can be solved in the pseudo-polynomial time in the absence of uni-processor tasks. There are, however, still existing multiprocessor task-scheduling problems with unknown complexity. It is, for example, the case of problem $P4 \mid any \mid C_{\max}$, i.e. the problem of non-preemptive scheduling on four processors of tasks whose execution time depends on the number of assigned processors when executing uni-processor tasks are allowed. Yet another perplexing problem is $P \mid fix_j, pmtn \mid C_{\max}$, i.e. the problem of preemptive

scheduling tasks requiring a set of processors simultaneously when the number of processors is not fixed. $\boldsymbol{NP}$-hardness of a very similar but slightly different problem has been shown in [19] (in fact, a "discretized" version of the problem has been considered in this work, because it has been assumed that the starting times and finishing times of processing the task, or its part, have discrete values).

This research has been partially supported by grant No. 3P40600106 of the State Committee of Scientific Research.

INSTITUTE OF COMPUTING SCIENCE, TECHNICAL UNIVERSITY OF POZNAŃ, PIOTROWO 3A, PL-60-965 POZNAŃ
(INSTYTUT INFORMATYKI POLITECHNIKI POZNAŃSKIEJ)

REFERENCES

[1] M. J. Atallah, C. Lock Black, D. C. Marinescu, *Models and algorithms for coscheduling compute-intensive tasks on a network of workstations,* J. Parallel and Distrib. Comput., **16** (1992) 319–327.

[2] A. Avizienis, *Fault tolerance: the survival attribute of digital systems,* Proc. IEEE, **66/10** (1978) 1109–1125.

[3] K. Baker, *Introduction to sequencing and scheduling,* John Wiley & Sons, New York, 1974.

[4] J. Błażewicz, P. Dell'Olmo, M. Drozdowski, M. G. Speranza, *Scheduling multiprocessor tasks on three dedicated processors,* Inf. Process. Lett., **41/5** (1992) 275–280; *Corrigendum,* ibid., **49** (1994) 269–270.

[5] J. Błażewicz, M. Drabowski, J. Węglarz, *Scheduling multiprocessor tasks to minimize schedule length,* IEEE Trans. Comput., **C35/5** (1986) 389–393.

[6] J. Błażewicz, M. Drozdowski, G. Schmidt, D. de Werra, *Scheduling independent two processor tasks on a uniform duo-processor system,* Discrete Appl. Math., **28** (1990) 11–20.

[7] J. Błażewicz, M. Drozdowski, G. Schmidt, D. de Werra, *Scheduling independent multiprocessor tasks on a uniform k-processor system,* Poznań Univ. Technol., Inst. Comput. Sci., Tech. Rept. R-92/030.

[8] J. Błażewicz, M. Drozdowski, J. Węglarz, *Scheduling multiprocessor tasks — a survey,* Int. J. Microcomput. Appl., **13/2** (1994) 89–97.

[9] J. Błażewicz, K. Ecker, G. Schmidt, J. Węglarz, *Scheduling in Computer and Manufacturing Systems,* Springer, New York, 1993.

[10] J. Błażewicz, J. K. Lenstra, A. H. G. Rinnoy Kan, *Scheduling subject to resource constraints: classification and complexity,* Discrete Appl. Math., **5** (1983) 11–24.

[11] G. I. Chen, T. H. Lai, *Preemptive scheduling of independent jobs on a hypercube,* Inf. Process. Lett., **28** (1988) 201–206.

[12] E. G. Coffman Jr., *Computer and job-shop scheduling theory,* John Wiley & Sons, New York, 1976.

[13] E. G. Coffman Jr., M. R. Garey, D. S. Johnson, A. S. Lapaugh, *Scheduling file transfers,* SIAM J. Comput., **14/3** (1985) 744–780.

[14] M. Dal Cin, E. Dilger, *On diagnosibility of self-testing multimicroprocessor systems,* Microprocess. and Microprogramm., **7/3** (1981) 177–184.

[15] J. Du, J. Y-T. Leung, *Complexity of scheduling parallel task systems,* SIAM J. Discrete Math., **2/4** (1989) 473–487.

[16] D. G. Feitelson, L. Rudolph, *Gang scheduling performance benefits for fine-grain synchronization,* J. Parallel and Distrib. Comput., **16** (1992) 306–318.

[17] E. Gehringer, D. Siewiorek, Z. Segall, *Parallel processing. The Cm* experience,* Digital Press, 1987.

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

[19] M. Kubale, *Preemptive scheduling of two-processor tasks on dedicated processors,* [in Polish], Zesz. Nauk. Politech. Śląskiej, Ser.: Autom., **100** (1990) 145–153.

[20] E. L. Lloyd, *Concurrent task systems,* Oper. Res., **29/1** (1981) 182–201.

[21] C. Seitz, *The Cosmic Cube,* Commun. ACM, **28/1** (1985) 22–33.

[22] X. Shen, E. M. Reingold, *Scheduling on a hypercube,* Inf. Process. Lett., **40/6** (1991) 323–328.

[23] J. A. Stankovic, K. Ramamritham, *Hard real-time systems,* Comput. Soc. IEEE, Washington 1988.

[24] . B. Veltman, B. J. Lageweg, J. K. Lenstra, *Multiprocessor scheduling with communication delays,* Parallel Comput., **16** (1990) 173–182.