Scheduling multiprocessor tasks - an overview

Maciej Drozdowski*

Abstract

Multiprocessor tasks require more than one processor at the same moment of time. This relatively new concept in scheduling theory emerged with the advent of parallel computing systems. In this work we present the state of the art for multiprocessor task scheduling. We show the rationale behind the concept of multiprocessor tasks. The standard three-field notation is extended to accommodate multiprocessor tasks. The main part of the work is presentation of the results in multiprocessor tasks scheduling both for parallel and for dedicated processors.

Keywords: parallel processing, scheduling, multiprocessor tasks, coscheduling, gang-scheduling.

1 Introduction

Parallel processing is expected to bring a breakthrough in the increase of computing speed. The new parallel computer systems are often expensive assets that must be shared by many users. Parallel applications can take the advantage of parallelism only when their parts do not wait for data longer than necessary. This calls for appropriate scheduling strategies controlling access to such resources as well as scheduling strategies controlling execution of the parallel application modules. There are some new concepts on the grounds of deterministic scheduling theory proposed to deal with scheduling parallel applications. One of them is scheduling with communication delays [36, 39, 72]. Another approach is divisible job theory [4, 15, 34]. In this paper we deal with yet another approach, namely, with multiprocessor tasks. Multiprocessor tasks require more than one processor at the same moment of time. This contradicts the classical scheduling theory assumption that a task is executed on one processor at a time only.

*Instytut Informatyki, Politechnika Poznańska, Poznań, Poland. The research has been partially supported by KBN grant 3P40600106.

The idea of multiprocessor tasks receives a growing attention in the 'scheduling community'. Since the field is rather new there is an inevitable risk of repeating some research. Hoping to lessen this problem we present recent results in the multiprocessor task scheduling. The organization of the paper is the following. Section 2 presents the justification for considering multiprocessor tasks. In Section 3 we define the problem and introduce the notation used. Section 4 presents the results for parallel processors and Section 5 for dedicated processors.

2 Motivation

A good scheduling strategy is an indispensable element of an efficient parallel computer system. Many distributed and multiprocessor computer systems offer some kinds of parallelism. It is not so evident, however, that the concurrency of the application execution is guaranteed, especially in the extreme load conditions. In this section we are going to explain that simultaneous execution on multiple processors should be provided.

Consider a general purpose shared-memory multiprocessor system with time sharing. A parallel application which consists of many concurrent threads (tasks within a program) is run on a number of processors. The access to a critical section is guarded by a lock which must be acquired by threads using the section. A thread which captured the lock must compete for the processors with an uncertain number of other threads. Soon it can lose its processor. Then, other threads must busy-wait for the release of the lock. But this, will not take place as long as the thread which holds the lock is not running. Thus, a bad decision about scheduling such a crucial thread results in a significant performance degradation. Let us consider a different situation. The threads of the same application communicate with each other but are run in different time quanta. Suppose that one thread tries to communicate with some other thread. It sends the data, but since the other thread is suspended, the message must wait at least until the end of the time quantum. In one of the following time quanta the receiving thread receives the data, processes it and sends back the results. The results in order to be received must still wait for the first thread to start running . In both of the above examples the progress in computation depends on the speed of context switching rather than on the raw speed of the processors or the communication system. *Coscheduling* (called also *gang scheduling*) is a scheduling policy proposed to avoid these difficulties [50]. Coscheduling consists in granting simultaneously (in the same time quantum) the processors to the threads of the same application. It has been demonstrated in [49, 84] that coscheduling performs well in a wide range of conditions and for various models of parallel applications. Thus, coscheduling is postulated in parallel systems. Note, that coscheduled applications are multiprocessor tasks because more than one processor is used simultaneously.

Parallel applications are often represented by DAGs. Yet, this representation has only a limited applicability for the operating system. Contemporary parallel applications have DAGs with thousands of nodes. At the current state of technology, it is hard to imagine a scheduler able to handle, analyze and optimize so big structures. Furthermore, threads of the application are indistinguishable for the scheduler. Hence, without additional information from the application it is not able to give a priority to important threads (e.g. holding a lock) [84]. Note, that the DAG is highly data-dependent and can be precisely known after the execution rather than before. From the above we conclude that it is reasonable for the operating system to control only the number of processors granted to a parallel application and leave the control of threads to the application (i.e. to the compiler and the programmer).

Following these observations a number of massively parallel computer systems divide their processors into *partitions* [26]. The main idea of a partition is to give an exclusive access to a number of processors to one application only. The operating system is responsible for managing the partitions, granting the access to them etc. Note, that from the viewpoint of the partition manager the applications are multiprocessor tasks because they occupy all the processors in the partition at the same moment of time.

In computer control systems a high level of reliability is often achieved by executing redundant copies of the program on different processors and voting on the final control decision [3, 50, 56]. Such applications are multiprocessor tasks because more than one processor is simultaneously occupied.

In the preceding discussion we concentrated on parallel processors. Now, we are going to demonstrate that multiprocessor tasks scheduling is also applicable in the case of dedicated (i.e. specialized) processors. In massively parallel computers there are processing nodes equipped with communication and other I/O hardware (e.g. disks), while other processing nodes are not.

There can be nodes with arithmetic, vector, graphic, signal processing facilities, while there can be other nodes without them. It is not inconceivable to present parallel applications requiring a little bit of all these facilities. Thus, multiprocessor tasks can be considered also in the case of dedicated processors. Not only specialization of the processing elements can justify considering processors as dedicated. A multiprocessor task can be regarded as executed by a dedicated processor also for preallocation reasons. For a certain communication pattern among the parts of the parallel application and for a given communication network it can be advantageous to map tasks to processors in some fixed way. Changing the preallocation may increase the communication overhead (due to dilatation, congestion etc.). In some cases even parallel processors may behave as dedicated devices. Since the costs of filling a pipeline, vector registers or a cache are high it is disadvantageous to frequently transfer threads to new sites. Hence, there is a kind of affinity between tasks and processors [69].

Testing of processors by one another requires at least two processors simultaneously [60]. Due to the fact that the graph of mutual tests cannot be arbitrary to guarantee testability of the system [53, 71] one may regard a test as a dedicated task. A similar situation takes place in testing VLSI chips where some functional units are simultaneously required to test other units [40]. Another application for multiprocessor task scheduling can be scheduling of file transfers [38]. A file transfer requires at least two 'processing' elements simultaneously: the sender and the receiver. Also simultaneous transfers on multiple buses can be considered as multiprocessor tasks [57]. Finally, simultaneous execution of multiple instructions in a superscalar processor requires matching instructions in such a way that the sets of simultaneously required processor units do not intersect. Scheduling in this case is performed by a compiler or by the hardware of the processor.

Though we introduced multiprocessor tasks in the computer context it is not difficult to find application for this kind of scheduling in production systems. In fact, the first papers considering an idea of simultaneous execution of a task by many processors dealt with scheduling operations in chemical plants [25] and project scheduling [77].

3 Definitions

In this section we formulate the multiprocessor task scheduling problem. For deterministic scheduling three elements define the problem: the processor set, the task system and the optimality criterion. We describe them in the following paragraphs.

We assume that processor set $\mathcal{P}=\{P_1,\dots,P_m\}$ consists of m elements. There are two types of processors: dedicated and parallel. Dedicated processors are usually specialized to perform different functions. In some situations, mentioned in the previous section, even identical parallel processors may behave as dedicated. In a dedicated environment a multiprocessor task requires certain processors not just some number of them. Hence, in the case of dedicated processors a multiprocessor task can be executed by a *set* of processors required simultaneously. Moreover, a task may be executed by some family of alternative processor sets. As it is in the classical scheduling theory [21, 37] multiprocessor tasks may consist of operations. In such a case we distinguish three types of dedicated processor systems: flow-shop, open-shop and job-shop. In the flow-shop all tasks have the same number of operations which are performed sequentially and require the same sets of processors. In the open-shop the order among the operations is immaterial. For the job-shop, the sequence of operations and the sets of required processors are defined for each task separately.

In the case of parallel processors each processor is capable of processing any task. Hence, a task requires some *number* of arbitrary processors. As in the classical scheduling theory parallel processors are divided into three classes: identical processors, uniform processors and unrelated processors.

The second element of the scheduling problem is the task system. We assume that the set of tasks $\mathcal{T}$ consists of n tasks $T_1,\dots,T_n$. For the whole task system it is possible to determine such features as preemptability (or nonpreemptability) and existence (or unexistence) of precedence constraints. These characteristics are defined as in the classical scheduling theory [21, 37]. A new feature is *fixed* or *variable profile* of a task. A profile of the multiprocessor task is fixed when the number (for parallel processors) or the set (for dedicated processors) of used processors does not change during the execution of the task. A profile is variable if it is possible to change the above characteristics within the schedule.

Each task T_j $(j=1,\dots,n)$ is defined by a number of parameters. We describe them in the following.

1. *Number of operations* n_j. This parameter is given for tasks scheduled on dedicated processors. $n_j > 1$ implies that task T_j consists of operations $\{o_{j1}, \ldots, o_{jn_j}\}$.

2. The *set of simultaneously required processors* fix_j or the *family of alternative processors* set_j. These parameters are defined only in the case of dedicated processors. The multiprocessor task requires for its processing a set fix_j of dedicated processors simultaneously. It is also possible that more than one set of processors can execute a task. Such a set of alternative processor ensembles will be called a family of alternative processors set_j. We will use the concept of a family of alternative processors only when $\mid set_j \mid > 1$. Analogously, for flow-shop, open-shop, and job-shop set fix_{ji} or family set_{ji} is defined for every operation o_{ji}.

3. The *number of simultaneously required processors* $size_j$ or the *maximum number of usable processors* δ_j. These parameters are defined in the case of parallel processors. When the first parameter is given the task can be executed only on $size_j$ processors required simultaneously. The multiprocessor task can be executed by some number of processors from the range $[1, \delta_j]$ if the second parameter is given. (Note, that for uniprocessor tasks $size_j = \delta_j = 1$.) It is possible that none of the above parameters is given when the number of processors executing a task is neither fixed nor bounded. We will denote $\Delta = \max_{T_j \in \mathcal{T}}\{\delta_j\}$. For tasks that can be executed only by one number of processors $\Delta = \max_{T_j \in \mathcal{T}}\{size_j\}$.

4. *Execution time.* For task T_j to be processed on set fix_j of dedicated processors the execution time will be denoted by $t_j^{fix_j}$. When task T_j can be executed by a family of alternative processors set_j then for each $fix_{ji} \in set_j$ the processing time is defined and denoted by $t_j^{fix_{ji}}$, where $i = 1, \ldots, |\, set_j \,|$. In the case of preemptable tasks it is necessary to determine how long a task must be processed (possibly by different processor sets) to consider it as finished. Analogously to the classical scheduling on uniform and unrelated processors we assume that task T_j executed in l different time intervals of lengths τ_i, by processors in sets fix_{ji} $(i = 1, \ldots, l)$, is finished when $\sum_{i=1}^{l} \frac{\tau_i}{t_j^{fix_{ji}}} \geq 1$. For a task consisting of n_j operations the execution time $t_{ji}^{fix_{ji}}$ is defined for each operation o_{ji} requiring processors in set fix_{ji}. Analogously, for operation o_{ji} with family set_{ji} of alternative processors execution time $t_{ji}^{fix_{jil}}$ is

defined for each $fix_{jil} \in set_{ji}$.

The situation is different in the case of parallel processors. Let us consider identical processors first. When task T_j can be executed only by $size_j$ processors, its execution time is $t_j^{size_j}$. If it can be executed by various numbers of processors, then for each feasible number k of processors execution time t_j^k is defined. Let us note that there is a number of models describing the relation between execution time and the number of allowed processors. In the scheduling literature it is:

- an arbitrary discrete function [47],
- inversely proportional function (i.e. $t_j^k = \frac{t_j^1}{k}$) [78],
- inversely proportional function up to δ_j processors [81],
- a function inversely proportional to k^α (i.e. $t^k = \frac{t_j^1}{k^\alpha}$) where $0 < \alpha < 1$ [75],
- an arbitrary continuous function [83].

Analogously, to the case of dedicated processors we consider preemptable task T_j executed in l intervals of length τ_i on k_i processors $(i = 1, \dots, l)$ as being finished when $\sum_{i=1}^{l} \frac{\tau_i}{t_j^{k_i}} \geq 1$. For uniform and unrelated processors the execution time of the task can be calculated as in the classical case.

Parameters such as ready time r_j, due-date d_j, weight w_j, and additional requirements R_{ji} of resource i are defined in the same way as in the classical scheduling theory.

In a feasible schedule of multiprocessor tasks it is decreed beyond the standard requirements that:

- In the case of dedicated processors, a multiprocessor task requiring processors in set fix_j is granted all these processors throughout all its execution time. Multiprocessor task with family set_j of alternative processors is executed by exactly those processors which are specified in some set(s) $fix_{ji} \in set_j$. Operation o_{ji} receives all processors required in fix_{ji} or when $| set_{ji} | > 1$ all processors included in some set(s) $fix_{jil} \in set_{ji}$, where $l = 1, \dots, |set_{ji}|$.

- In the case of parallel processors, a multiprocessor task which can be executed by only one number $size_j$ of processors is granted that number of processors simultaneously throughout all its execution time. When maximum number δ_j of usable processors is specified, in no moment of time is the task executed by more than δ_j processors simultaneously.

The criteria of the optimality are defined in the standard way [21, 37].

To denote the analyzed scheduling problems we will use the standard three-field notation $\alpha \mid \beta \mid \gamma$ proposed in [52] with extensions introduced in [22, 76]. Yet, the modifications proposed in [76] are not satisfactory to describe the variety of the considered multiprocessor scheduling problems. Hence, we propose further expansion of the notation. The scheme $\alpha \mid \beta \mid \gamma$ in its three fields describes the processor system (α), the task system (β), and the optimality criterion (γ).

The first field contains up to three symbols $\alpha = \alpha_1, \alpha_2, \alpha_3$. The first two symbols are the standard ones:

- $\alpha_1 \in \{1, P, Q, R, O, F, J\}$ - describes the type of processors,

- $\alpha_2 \in \{k, \circ\}$ - denotes the number of processors fixed to k or not fixed in the definition of the problem.

The third symbol $\alpha_3 \in \{win, \circ\}$ - denotes, respectively, that processors are available in time windows or always available.

The second field $\beta = \beta_1, \ldots, \beta_6$ defines the task system.

$\beta_1 \in \{spdp\text{–}lin, spdp\text{–}lin\text{–}\delta_j, spdp\text{–}any, size_j, cube_j, fix_j, fix_{ij}, set_j, set_{ij}, \circ\}$ - describes the type of the multiprocessor task.
- $\beta_1 = spdp\text{–}lin$ - denotes that t_j^k is inversely proportional to k, in other words, speedup is linear.
- $\beta_1 = spdp\text{–}lin\text{–}\delta_j$ - describes a situation similar to the previous one, but the task cannot use more than δ_j processors simultaneously.
- $\beta_1 = spdp\text{–}any$ - execution time t_j^k is an arbitrary function of k.
- $\beta_1 = size_j$ - a task can be executed only by one number $size_j$ of processors.
- $\beta_1 = cube_j$ - is a special case of $size_j$ demanding that the tasks be executed by numbers of processors being powers of 2 (1, 2, 4, 8,... etc. processors). This situation refers to scheduling on hypercubes.
- $\beta_1 = fix_j$ - denotes that the task can be executed by only one set fix_j of simultaneously required dedicated processors. In the case of multiprocessor task comprising a number of operations we use $\beta_1 = fix_{ij}$.
- $\beta_1 = set_j$ - means that tasks have families of alternative dedicated processors which can execute the tasks. In the case of tasks with operations we will use $\beta_1 = set_{ij}$.
- $\beta_1 = \circ$ - stands for standard uniprocessor tasks.

According to the current value of β_1 we will say that tasks require processors according to model $spdp\text{–}lin$, $spdp\text{–}lin\text{–}\delta_j$ etc.

$\beta_2 \in \{var, pmtn, \circ\}$ - denotes preemptability and variability of the profile or their absence.

- $\beta_2 = var$ - denotes variable profile. Note, that $\beta_2 = var$ implies $\beta_1 \in \{spdp\text{–}lin, spdp\text{–}lin\text{–}\delta_j, spdp\text{–}any, set_j, set_{ij}\}$.
- $\beta_2 = pmtn$ - tasks are preemptable, but the profile is fixed.
- $\beta_2 = \circ$ - denotes that tasks are nonpreemptable and their profiles are fixed.

The rest of the notation for the task system is classical:

$\beta_3 \in \{prec, tree, chain, \circ\}$ - describes the type of precedence constraints.

$\beta_4 \in \{p_j = 1, p_{ij} = 1, \circ\}$ - means, respectively, that processing times of tasks are equal, processing times of operations are equal, processing times are arbitrary.

$\beta_5 \in \{r_j, \circ\}$ - denotes that tasks have different ($\beta_5 = r_j$) or identical ($\beta_5 = \circ$) ready times .

$\beta_6 \in \{res\lambda\sigma\rho, \circ\}$ where $\lambda\sigma\rho \in \{k, \cdot\}$ - denotes the type of additional resource requirements ($res\lambda\sigma\rho$) or absence of such requirements ($\circ$). $\lambda, \sigma, \rho = k$ denote, respectively, the number of resource types is k, each resource has k units, k is the maximal number of any resource units required by any task. $\lambda, \sigma, \rho = \cdot$ denote that the above values are arbitrary.

The third field $\gamma = \gamma_1$ where $\gamma_1 \in \{C_{max}, L_{max}, U, \sum C_j, \sum w_j C_j, \sum w_j U_j, \sum T_j, X\}$ denotes the optimality criterion. When a non-standard optimality criterion was considered we denoted such a case by X.

Many of the multiprocessor task scheduling problems are computationally hard. In such cases heuristics are often proposed. To evaluate the worst-case performance of some heuristic H we will use the worst-case performance ratio (performance ratio in short): $S_H = \inf\{r \geq 1 : \forall_{I \in D} \frac{f_H(I)}{OPT(I)} \leq r\}$, where I - is the instance, D - the domain of the scheduling problem, $f_H(I)$ - the value of the optimality criterion for I and the schedule generated by H, $OPT(I)$ - the optimal value of the criterion for I. For the sake of conciseness we will write **NP**h to denote that some problem is **NP**-hard, and s**NP**h to denote that the problem is **NP**-hard in the strong sense.

Multiprocessor task scheduling is tightly linked with other types of scheduling and with combinatorial optimization in general. Scheduling multiprocessor tasks on dedicated processors is similar to scheduling with resource constraints. The latter problem, however, distinguishes between processors and resources while multiprocessor task scheduling does not. Multiprocessor task scheduling generalizes the classical scheduling on dedicated processors. The multipurpose machine scheduling [27] and scheduling with restricted processor allocation [30, 33, 58, 59] are special cases of scheduling multiprocessor tasks with families of alternative processors where tasks are uniprocessor. In this paper we restrict ourselves to the works explicitly considering tasks requiring more than one processor at the time. Observe, that problems of scheduling on dedicated processors are closely related to various types of graph coloring. Fixed-profile nonpreemptive scheduling on parallel processors can be represented as 2-dimensional packing: the number of required processors constitute one dimension and time the second dimension.

4 Parallel processors

In this section we consider scheduling on parallel processors. In the absence of a better one, the order of presentation will be (in general) chronological according to appearance in the printed form referred to in the literature section. Obviously, it is approximate chronology and many papers are known in a preliminary form years before the final publication. Due to space limitations only some papers can be commented on. Table 1 presents the results in multiprocessor scheduling on parallel processors.

One of the first papers considering multiprocessor task scheduling was [68]. It was shown that problems $P \mid size_j, p_j = 1 \mid C_{max}$ and $P3 \mid size_j, p_j = 1, prec \mid C_{max}$ with $size_j \in \{1, 2\}$ are **NP**h. The performance of any list scheduling heuristic (LS in short) has been proved to be bounded from above by $(2m - \Delta)/(m - \Delta + 1)$ and the ratio of $\lfloor (2m - \Delta)/(m - \Delta + 1) \rfloor$ has been achieved. For problem $P2 \mid size_j, p_j = 1, prec \mid C_{max}$ an algorithm with complexity $O(n^2)$ based on the Coffman-Graham algorithm was proposed.

Problem $R \mid spdp\text{–}lin, var, r_j \mid L_{max} = 0$ (i.e. testing the existence of a feasible schedule with $L_{max} = 0$) was reduced in [77] to solving a linear program with $O(n^3)$ variables and $O(n^2)$ constraints.

In [80] problem $P \mid spdp\text{–}lin, var, r_j \mid X$ is analyzed. For each task a deadline is given. All tasks must be completed in time. To achieve this goal each processor is capable of increasing its speed. Firstly, the optimality criterion is minimizing the maximum speed necessary for processing the tasks. Secondly, when processing at some speed is unavoidable then the period of processing at such speed is minimized. In [78] a similar problem is considered, but the optimality criterion is the total intensity cost, where the cost function is convex.

Problem $P \mid spdp\text{–}lin\text{–}\delta_j, var, r_j \mid L_{max}$ was considered in [79]. This problem can be solved by a reduction to a sequence of equivalent maximum network flow problems which check the existence of a feasible schedule for some given value of L_{max}. $O(\sum_{i=1}^{n} \delta_j) \leq O(nm)$ calls to an $O(n^3)$ network flow algorithm are required. The total complexity is $O(n^4 m)$.

In [14] preemptive and nonpreemptive scheduling of multiprocessor tasks is considered. This paper extends preliminary results of [24]. For problem $P \mid size_j, p_j = 1 \mid C_{max}$ and $size_j \in \{1, \Delta\}$ an $O(n)$ algorithm is proposed. When the numbers of simultaneously required processors are in the set $\{1, \ldots, \Delta\}$ and Δ is fixed the above problem can be solved in $O(n)$ time by an integer linear program with fixed number of variables. Problem $P \mid size_j, p_j = 1 \mid C_{max}$ was shown to be s**NP**h in general. For problem $P \mid size_j, pmtn \mid C_{max}$ where $size_j \in \{1, \Delta\}$ it was proved that among the optimal schedules there must be a so-called *A-schedule*. In the A-schedule tasks with $size_j = \Delta$ are assigned in the interval $[0, C_{max}]$ using McNaughton's wrap-around-rule, and tasks with $size_j = 1$ are assigned in the same way in the remaining part of the schedule. An algorithm building *A-schedules* in $O(n)$ time was proposed for this problem. For the general case of $size_j$ values an algorithm based on linear programming (LP) and the concept of a *processor feasible set* has been proposed. The processor feasible set is a set of tasks that can be executed in parallel on the given number of processors. Note, that for n tasks and m processors there are $O(n^m)$ processor feasible sets. Thus, the LP can be formulated and solved in polynomial time, provided m is fixed.

In [20] problem $P \mid size_j, pmtn, res1 \cdot 1 \mid C_{max}$ was considered. It was assumed that $size_j \in \{1, 2\}$, all tasks with $size_j = 1$ required a unit of the resource, while only some tasks with $size_j = 2$ required the resource. It was shown that among the optimal schedules there must exist an A-schedule. An $O(n \log n)$ algorithm has been proposed. This problem was further analyzed in [21] for $\Delta > 2$ and tasks with $size_j = 1$ requiring a unit or no resource. An $O(nm)$ algorithm was given.

Problem $P \mid cube_j, pmtn \mid C_{max}$ of scheduling on hypercube has been tackled in [31]. An $O(n^2)$ algorithm has been proposed to test whether a feasible schedule of length T exists. The algorithm builds *stair-like* schedules. A schedule is stair-like when (i) each processor P_j is busy before time $f(P_j)$ and idle after $f(P_j)$, (ii) f is nonincreasing function of processor number. In other words, the stair-like schedule consists of a number of 'steps'. Tasks are scheduled in the order of decreasing $size_j$. A task is scheduled such that it ends at the common deadline T, 'steps' of the stair-like schedule are consecutively filled one by one from left to right and no sooner is the lower (less loaded) step used than the higher one is completely full. This results in $O(n^2)$ preemptions. The testing algorithm can be applied in time $O(n^2(\log n + \log \max_j\{t_j^{size_j}\}))$ to find the optimal schedule (C_{max} is calculated with unit granularity).

In [47] the authors proved that problems $P2 \mid size_j, chain \mid C_{max}$, and $P5 \mid size_j \mid C_{max}$ are s**NP**h. Each schedule for $P2 \mid size_j \mid C_{max}$, and $P3 \mid size_j \mid C_{max}$ can be transformed into a *canonical schedule* for which dynamic programming can be used to obtain the optimal schedule. The complexity of problem $P4 \mid size_j \mid C_{max}$ remains open. The fixed-profile preemptive scheduling has been shown to be **NP**h for $P2 \mid \textit{spdp–any}, pmtn \mid C_{max}$ and s**NP**h for $P \mid \textit{spdp–any}, pmtn \mid C_{max}$. When the number of processors is fixed, a dynamic program has been given.

In [54] an $O(n \log n)$ algorithm testing the existence of a schedule for problem $P \mid cube_j, pmtn \mid C_{max}$ has been proposed. The algorithm differs from the one from [31] in the use of *pseudo-stairlike* schedules. In the pseudo-stairlike schedule a task is not filling 'steps' one by one, but fills at most two 'steps' (subcubes): one from the moment it becomes available till the end of the schedule and possibly one more but only partially. This results in lower number of preemptions $O(n)$. The algorithm can be applied in time $O(n \log n(\log n + \log \max_j\{t_j^{size_j}\}))$ to find the optimal schedule by a binary search. A similar approach has been proposed independently in [1].

In [83] problem $P \mid \textit{spdp–any}, var, r_j \mid L_{max}$ is considered. The processors are assumed to be numerous enough to deal with them as with a continuous medium. Each task is described by a continuous function binding the processing speed and the number (amount) of assigned processors. The problem is reduced to a set of nonlinear equations.

In [16] problem $Q \mid size_j, pmtn \mid C_{max}$ of scheduling on uniform processors was considered. It was also assumed that $size_j \in \{1, 2\}$ and that processors form pairs of equal speed. An $O(n \log n + nm)$ algorithm was proposed. First, a lower bound of the schedule length is calculated. Tasks with $size_j = 2$ are scheduled in the order of decreasing processing times. Then, in the remaining free intervals tasks with $size_j = 1$ are scheduled. When the schedule is to short to accommodate all the tasks it is extended by some calculated amount of time. This algorithm was extended to solve problem $Q \mid size_j, pmtn \mid C_{max}$ with $size_j \in \{1, \Delta\}$ in [17], and to solve problem $Q \mid cube_j, pmtn \mid C_{max}$ in

[18]. In [43] results of computational experiments on the above algorithm for $Q \mid cube_j, pmtn \mid C_{max}$ are reported. For problem $Qm \mid size_j, pmtn \mid C_{max}$ with arbitrary $size_j$ a solution based on linear programming and feasible sets has been proposed in [18].

Heuristics for scheduling $P \mid spdp-lin-\delta_j, prec \mid C_{max}$ were proposed in [81]. It was proved that any LS algorithm has tight performance ratio $\Delta + \frac{m-\Delta}{m}$. The standard LS algorithm assigns a task to the first available processor. However, it can be advantageous to delay the starting of the task until a moment when more processors are available. The Earliest Completion Time (ECT) heuristic is an LS heuristic which assigns tasks to ready processors in a manner minimizing the completion time of a task. The worst case performance ratio of ECT was shown to be less than $\ln \Delta + 2$. Further analysis of ECT in [82] proved that the performance is bounded from above by $3 - \frac{2}{m}$ and an instance with 2.5 performance ratio was demonstrated. The same idea as of ECT algorithm was independently proposed in [2, 48] to find the set of processors executing a compute intensive task on a distributed workstation system (which is a variation of problem $Q, win \mid spdp\text{–}any \mid C_{max}$ with $n = 1$). An algorithm with complexity $O(m^2)$ was proposed in [2] and with complexity $O(m \log m)$ in [48].

Preemptive scheduling on a hypercube was considered again in [74]. A feasibility testing algorithm of [1] was modified to obtain complexity $O(nm)$. By the observation that in the optimal schedule at least one task must use all the time remaining up to the end of the schedule, an $O(n^2m^2)$ algorithm finding optimal schedule for $P \mid cube_j, pmtn \mid C_{max}$ was given. The above algorithm uses for each subcube a parametric representation of the remaining processing time as a linear function of some (hypothetical) common deadline T. The parameters are modified as a result of building partial schedules and consuming the free processing time by the scheduled tasks. Using such a function the optimum schedule length can be calculated. A similar approach and an $O(n^2 \log^2 n)$ algorithm finding the optimal schedule was proposed in [85].

A variation of $P \mid spdp\text{–}any \mid C_{max}$ was considered in [61]. It was assumed that $n \leq m$, all tasks (at least initially) are executed in parallel, and minimization of C_{max} was achieved by changing the number of processors used by the tasks. The proposed approximation algorithm successively increases the number of processors used by the longest tasks until δ_j is achieved or all m processors are occupied. We will call this algorithm Varying Size (VS).

The tight performance ratio of VS is $\min\{n, R/(1-n/m)\}$, where R is the maximum of the ratio of two successive acceptable sizes of any task.

In [19] problem $Pm \mid size_j, pmtn \mid L_{max}$ for $size_j \in \{1, \Delta\}$ was considered. An LP based on the processor feasible sets was proposed. Since this method requires an LP with big number of variables an approximation method based on tabu search and linear programming has been proposed. The reported good experimental results for the second method have been explained by a particular topology of the criterial function.

For $P \mid cube_j \mid C_{max}$ Largest Dimension First (LDF) heuristic with tight performance ratio $2-1/m$ is proposed in [85].

In [62] an experimental study is reported for dynamic scheduling (i.e. the set of tasks is not known in advance) for problem $P \mid cube_j \mid \sum C_j$. It is observed that even sophisticated processor allocation strategies alone cannot guarantee good performance. A set of *Scan* strategies is proposed which combine the simple buddy allocation scheme with clustering tasks according to their $size_j$. Tasks with the same $size_j$ are appended to one queue. Queues with different size tasks are scanned in the direction of increasing (or decreasing) size. These scheduling strategies effectively overcome the shortcomings (weak ability to recognize idle subcubes) of the buddy allocator.

Problem $P \mid spdp-any, prec \mid C_{max}$ was considered in [75]. It is assumed that $\forall_j t_j^k = \frac{t_j^1}{k^\alpha}$. An algorithm for determining $size_j$ and sequencing tasks in an arbitrary DAG was proposed. Processors are considered here as a continuous medium which behaves like electrical charge passing from task to task in the DAG. The optimality conditions impose a set of nonlinear equations on the flow of processing power (processors) and on the completion times of independent paths of execution. These equations are analogous to Kirchhoff's laws of electrical circuit theory. An algorithm based on conjugate gradient method has been proposed. The complexity is $O(e^2+ne+I(n+e))$, where e - is the number of edges in the precedence graph and I - is the number of iterations in the algorithm.

A similar problem motivated by computer vision application is considered in [35] (a variation of $P \mid spdp-any, prec \mid C_{max}$). The difference here is that the computations are pipelined and the tasks constantly coexist on the processor set. Long sequences of data sets undergo processing by collection of tasks forming a series-parallel DAG. The throughput defined as the longest execution of a single task in the DAG is the interval between obtaining results for two consecutive data sets. Two problems are posed: for the given throughput find minimal response time, and for the given response time find maximal throughput. Heuristic algorithms are proposed.

In [44] problem $P \mid size_j, pmtn \mid C_{max}$ is proved to be **NP**h, but it is an open problem whether it is s**NP**h.

In [46] a special case of problem $P \mid spdp-lin-\delta_j, var, chain \mid C_{max}$ is considered. It is assumed that tasks form chains of three elements (denoted $\mid chain \mid = 3$): a sequential head ($size_j = \delta_j = 1$), parallel central part with an unboundedly linear speedup ($spdp$–$lin, \delta_j > m$) and a tail which is sequential again. This model is motivated by a master-slave model of computations. It was shown that the above problem is s**NP**h. Yet, the optimal schedule for the $m-1$ longest tasks can be extended to an optimal schedule for all the tasks. Furthermore, when m is fixed such a schedule can be obtained in polynomial time. When the chain of tasks consists of two elements of one type only, e.g. a head and a central part (denoted $\mid chain \mid = 2$), the optimal solution can be found in $O(n \log n)$ time. Three approximation algorithms have been proposed with tight performance bounds $3, 2, 2$, respectively.

In [23] problem $P3 \mid size_j, p_j = 1, chain \mid C_{max}$ is proved to be s**NP**h. When the chains consist of multiprocessor tasks which are preceding uniprocessor task and $size_j = \Delta > m/2$ for all multiprocessor tasks, an $O(n \log n)$ algorithm can be applied. For chains consisting either of multiprocessor tasks with $size_j = \Delta$ or of uniprocessor tasks the optimal schedule can be found in $O(n \log n)$ time (cf. [14]).

Table 1 presents results in multiprocessor task scheduling. The entries of the table are organized chronologically according to the model of requiring processors. The following abbreviations are used in Table 1: ? - an open problem with unknown complexity, pseudopoly. - a pseudopolynomial algorithm was proposed, LP - an algorithm based on linear programming, ILP - an algorithm based on integer linear programming.

Insert Table 1 here

5 Dedicated processors

The first paper considering multiprocessor scheduling seems to be [25] in which branch and bound (B&B) algorithm is proposed for scheduling in chemical plants. A concept of *compatibility* and *incompatibility* of tasks has been introduced. Two tasks T_i and T_j are compatible if $fix_i \cap fix_j = \emptyset$.

This can be easily extended to *incompatibility graph*. In bounding the search a Maximum Degree of Incompatibility (MDI) was an incentive to prefer executing some tasks over the others.

In [60] scheduling of diagnostic tests is analyzed. The tests to be performed are represented by a *diagnostic graph* in which nodes represent processors and edges - tasks. The edge has weight - processing time of a task. Two processors connected by the edge are simultaneously required to test each other. We will call such a representation *scheduling graph* (after [63]). The considered problem $P \mid fix_j \mid C_{max}$ with $\forall_j \mid fix_j \mid = 2$ is proved in [60] to be **NP**h. An LPT heuristic is analyzed, and the worst case performance bound $4(d-1)/d$ is demonstrated, where d is the maximum degree of any vertex. For graphs with $d \leq 5$ this bound is tightened to 3, and for binomial graphs with integral ratio of the weights it is 2.

In [38] the problem of scheduling file transfers is considered. Each computer may be able to use p ports to execute simultaneous file transfers. The transfers to be performed are described by a scheduling graph in which vertices are communicating nodes and edges are files to transfer. The problem is analyzed for the case with central controller as well as for the distributed case. It is proved that $4/3 < S_{LS} \leq 3$ and this bound can be tightened for special forms of the scheduling graph. LPT heuristic has performance ratio $5/2 - 1/p$ when $p \geq 2$. Two distributed protocols are proposed to schedule file transfers. For the first (called Demand Protocol 1) it is proved that $C_{max}^{DP1} \leq 3C_{max}^{*} + e\varepsilon$, where C_{max}^{DP1} is the length of the schedule, C_{max}^{*} the length of the optimal schedule, e is the number of edges in the scheduling graph, ε is the maximum time to initiate some file transfer. For the second protocol similar bounds have been obtained.

In [40] the analysis of problems $P \mid fix_j \mid C_{max}$ and $P \mid fix_j, p_j = 1 \mid C_{max}$ is motivated by scheduling of built-in tests for VLSI circuits. Heuristics based on Maximum Degree of Incompatibility are proposed.

In [64] preemptive scheduling is considered. By reduction of edge multicoloring problem $P \mid fix_j, pmtn \mid C_{max}$ with $\mid fix_j \mid = 2$ is proved to be s**NP**h (via equivalence with $P \mid fix_j, p_j = 1 \mid C_{max}$). For problem $Pm \mid fix_j, pmtn \mid C_{max}$, i.e. when the number of processors is fixed an algorithm based on linear programming and processor feasible sets is given.

In [13] the case of three processors is analyzed. The problem is s**NP**h. A normal schedule (NS) is the one in which tasks requiring two processors simultaneously are executed in parallel with tasks requiring the third processor. Three special cases are identified when normal schedules are optimal. In general case performance ratio of normal schedules is shown to be less than 4/3. For the same problem it is shown in [41] that normal schedules guarantee performance 5/4 and this bound is tight. A better approximation algorithm with tight performance ratio $S_{18} = 7/6$ has been proposed in [51] (we call it *18* for it chooses the best out of 18 schedules).

In [29] open-, flow- and job-shop with multiprocessor tasks are considered. For the open-shop it is assumed that the same number operations of different tasks require the same set of processors. In some of the considered problems the number of *stages* is fixed, i.e. for each task number of operations can be fixed. These problems are further pursued in [28]. In some cases number of task types was fixed to R.

Paper [6] considers Earliest Due-Date algorithm applied to solve $P2, 3, 4 \mid fix_j, pmtn \mid L_{max}$. EDD is linear time algorithm provided that the order of tasks according to the due-dates is given initially. For $P2 \mid fix_j, pmtn \mid L_{max}$ is shown to be optimal, for $m \in \{3, 4\}$ conditions are given under which EDD is optimal. Computational results are reported.

Scheduling according to model set_j is tackled in [10]. Dynamic programming formulations are given for $P2 \mid set_j \mid C_{max}$ and $P3 \mid set_j \mid C_{max}$ in the absence of one of the three duo-processor task types. For $P \mid set_j \mid C_{max}$ a heuristic scheduling tasks in the Shortest Processing Time Mode (SPTM) is proposed. Its tight performance ratio is m. For $Pm \mid set_j, pmtn \mid C_{max}$ a polynomial time algorithm based on processor feasible sets and linear programming is proposed.

In Table 2 results for scheduling multiprocessor tasks on dedicated processors are given. Abbreviation s.g. stands for scheduling graph.

Insert Table 2 here

6 Conclusions

Scheduling multiprocessor tasks is considered in this paper. Multiprocessor tasks require multiple processors simultaneously. Recent results in scheduling on parallel and dedicated processors are reported in Tables 1 and 2, respectively.

References

[1] M.Ahuja, Y.Zhu, "An O($n \log n$) feasibility algorithm for preemptive scheduling of n independent jobs on a hypercube", *Information Processing Letters* 35 (1990) 7-11.

[2] M.J.Atallah, C.L.Black, D.C.Marinescu, H.J.Siegel, T.L.Casavant, "Models and algorithms for coscheduling compute-intensive tasks on a network of workstations", *Journal of Parallel and Distributed Computing* 16 (1992) 319-327.

[3] A.Avizienis, G.C.Gilley, F.P.Mathur, D.A.Rennels, J.A.Rohr, D.K.Rubin, "The STAR (Self-Testing And Repairing) computer: An investigation of the theory and practice of fault-tolerant computer design", *IEEE Transactions on Computers* C-20/11 (1971) 1312-1321.

[4] V.Bharadwaj, D.Ghose, V.Mani, "Optimal sequencing and arrangement in distributed single-level tree networks with communication delays", *IEEE Transactions on Parallel and Distributed Systems* 5/9 (1994) 968-976.

[5] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Preemptive scheduling of multiprocessor tasks on the dedicated processors system subject to minimal lateness", *Information Processing Letters* 46 (1993) 109-113.

[6] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Linear algorithms for preemptive scheduling of multiprocessor tasks subject to minimal lateness", 1993, to appear in *Discrete Applied Mathematics.*

[7] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Scheduling preemptive multiprocessor tasks on dedicated processors", *Performance Evaluation* 20 (1994) 361-371.

[8] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Scheduling UET multiprocessor tasks", *Foundations of Computing and Decision Sciences* 19/4 (1994) 273-283.

[9] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Preemptive multiprocessor task scheduling with release times and time windows", 1994, to appear in *Annals of Operations Research.*

[10] L.Bianco, J.Błażewicz, P.Dell'Olmo, M.Drozdowski, "Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors", *Annals of Operations Research* 58 (1995) 493-517.

[11] L.Bianco, P.Dell'Olmo, M.G.Speranza, "Nonpreemptive scheduling of independent tasks with prespecified processor allocations", *Naval Research Logistics Quarterly* 41 (1994) 959-971.

[12] L.Bianco, P.Dell'Olmo, M.G.Speranza, "Scheduling independent tasks with multiple modes", *Discrete Applied Mathematics* 62 (1995) 35-50.

[13] J.Błażewicz, P.Dell'Olmo, M.Drozdowski, M.G.Speranza, "Scheduling multiprocessor tasks on three dedicated processors", *Information Processing Letters* 41 (1992) 275-280. Corrigendum, *IPL* 49 (1994) 269-270.

[14] J.Błażewicz, M.Drabowski, J.Węglarz, "Scheduling multiprocessor tasks to minimize schedule length", *IEEE Transactions on Computers* C-35/5 (1986) 389-393.

[15] J.Błażewicz, M.Drozdowski, "Scheduling divisible jobs on hypercubes", *Parallel Computing* 21 (1995) 1945-1956.

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

[17] J.Błażewicz, M.Drozdowski, G.Schmidt, D.de Werra, "Scheduling independent multiprocessor tasks on a uniform k-processor system", *Parallel Computing* 20 (1994) 15-28.

[18] J.Błażewicz, M.Drozdowski, G.Schmidt, D.de Werra, "Scheduling independent multiprocessor tasks on a uniform k-processor system", Technical Report R92/030, Institute of Computing Science, Poznań University of Technology, 1992.

[19] J.Błażewicz, M.Drozdowski, D.de Werra, J.Węglarz, "Deadline scheduling of multiprocessor tasks", *Discrete Applied Mathematics* 65 (1996) 81-96.

[20] J.Błażewicz, K.Ecker, "Scheduling multiprocessor tasks under unit resource constraints", Proceedings of International Conference on Optimization Techniques and Applications, Singapore, Apr. 1987, 161-169.

[21] J.Błażewicz, K.Ecker, G.Schmidt, J.Węglarz, *Scheduling in Computer and Manufacturing Systems*, Springer Verlag, Berlin, 1993.

[22] J.Błażewicz, J.K.Lenstra, A.H.G.Rinnoy Kan, "Scheduling subject to resource constraints: classification and complexity", *Discrete Applied Mathematics* 5 (1983) 11-24.

[23] J.Błażewicz, Z.Liu, "Scheduling multiprocessor tasks with chain constraints", 1995, private communication.

[24] J.Błażewicz, J. Węglarz, M. Drabowski, "Scheduling independent 2-processor tasks to minimize schedule length", *Information Processing Letters* 18 (1984) 267-273.

[25] G.Bozoki, J.-P.Richard, "A branch-and-bound algorithm for the continuos-process job-shop scheduling problem", *AIIE Transactions* 2/3 (1970) 246-252.

[26] T.Bönniger, R.Esser, D.Krekel, "CM-5, KSR2, Paragon XP/S: A comparative description of massively parallel computers", *Parallel Computing* 21 (1995) 199-232.

[27] P.Brucker, *Scheduling Algorithms*, Springer Verlag, Berlin, 1995.

[28] P.Brucker, A.Krämer, "Polynomial algorithms for resource constrained and multiprocessor task scheduling problems with a fixed number of task types", Osnabrücker Schriften zur Mathematik, Reihe P Preprints, Heft 165, May 1994, Fachbereich Mathematik/Informatik, Universität Osnabrück.

[29] P.Brucker, A.Krämer, "Shop scheduling problems with multiprocessor tasks and dedicated processors", *Annals of Operations Research: Mathematics of Industrial Systems I* 57 (1995) 13-27.

[30] R.S.Chang, R.C.T.Lee, "On a scheduling problem where a job can be executed only by a limited number of processors", *Computers and Operations Research* 15/5 (1988) 471-478.

[31] G.-I.Chen, T.-H.Lai, "Preemptive scheduling of independent jobs on a hypercube", *Information Processing Letters* 28 (1988) 201-206.

[32] G.-I.Chen, T.-H.Lai, "Scheduling Independent Jobs on Partitionable Hypercubes", *Journal of Parallel and Distributed Computing* 12 (1991) 74-78.

[33] Y.L.Chen, Y.H.Chin, "Scheduling unit-time jobs on processors with different capabilities", *Computers and Operations Research* 16/5 (1989) 409-417.

[34] Y.C.Cheng, T.G.Robertazzi, "Distributed Computation with Communication Delay", *IEEE Transactions on Aerospace and Electronic Systems* 24/6 (1988) 700-712.

[35] A.N.Choundhary, B.Narahari, D.M.Nicol, R.Simha, "Optimal processor assignment for a class of pipelined computations", *IEEE Transactions on Parallel and Distributed Systems* 5/4 (1994) 439-445.

[36] P.Chretienne, "Tree scheduling with communication delays", *Discrete Applied Mathematics* 49 (1994) 129-141.

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

[38] E.G.Coffman Jr., M.R.Garey, D.S.Johnson, A.S.Lapaugh, "Scheduling file transfers", *SIAM Journal on Computing* 14/3 (1985) 744-780.

[39] J.Y.Colin, P.Chrétienne, "C.P.M. scheduling with small communciation delays and task duplication", *Operations Research* 39/4 (1991) 680-684.

[40] G.L.Craig, C.R.Kime, K.K.Saluja, "Test scheduling and control for VLSI built-in self-test", *IEEE Transactions on Computers* 37/9 (1988) 1099-1109.

[41] P.Dell'Olmo, M.G.Speranza, Z.Tuza, "Easy and hard cases of a scheduling problem on three dedicated processors", 1993, private communication.

[42] G.Dobson, U.S.Karmarkar, "Simultaneous resource scheduling to minimize weighted flow times", *Operations Research* 37/4 (1989) 592-600.

[43] M.Drozdowski, "Scheduling multiprocessor tasks on hypercubes", *Bulletin of the Polish Academy of Sciences, Technical Sciences* 42/3 (1994) 437-445.

[44] M.Drozdowski, "On complexity of multiprocessor tasks scheduling", *Bulletin of the Polish Academy of Sciences, Technical Sciences* 43/3 (1995) 381-392.

[45] M.Drozdowski, "Real-time scheduling of linear speedup parallel tasks", *Information Processing Letters* 57 (1996) 35-40.

[46] M.Drozdowski, W.Kubiak, "Scheduling parallel tasks with sequential heads and tails", working paper of Faculty of Business Administration, Memorial University of Newfoundland, 1995.

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

[48] K.Efe, V.Krishnamoorthy, "Optimal scheduling of compute-intensive tasks on a network of workstations", *IEEE Transactions on Parallel and Distributed Systems* 6/6 (1995) 668-673.

[49] D.G.Feitelson, L.Rudolph, "Gang scheduling performance benefits for fine-grain synchronization", *Journal of Parallel and Distributed Computing* 16 (1992) 306-318.

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

[51] M.X.Goemans, "An approximation algorithm for scheduling on three dedicated processors", *Discrete Applied Mathematics* 61 (1995) 49-60.

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

[53] S.L.Hakimi, A.T.Amin, "Characterization of the connection assignment of diagnosable systems", *IEEE Transactions on Computers* 23/1 (1974) 86-88.

[54] C.P.M.van Hoesel, "Preemptive scheduling on a hypercube", Technical Report 8963/A, Erasmus University, Rotterdam, 1989.

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

[56] A.L.Hopkins, J.M.Lala, T.B.Smith, "FTMP - A highly reliable fault-tolerant multiprocessor for aircraft", *Proceedings of the IEEE* 66/10 (1978).

[57] R.Jain, K.Somalwar, J.Werth, J.C.Browne, "Scheduling parallel I/O operations in multiple bus systems", *Journal of Parallel and Distributed Computing* 16 (1992) 352-362.

[58] K.Jansen, "Scheduling with constrained processor allocation for interval orders", *Computers and Operations Research* 20/6 (1993) 587-595.

[59] H.Kellerer, G.Woeginger, "UET-scheduling with constrained processor allocations", *Computers and Operations Research* 19/1 (1992) 1-8.

[60] H.Krawczyk, M.Kubale, "An approximation algorithm for diagnostic test scheduling in multicomputer systems", *IEEE Transactions on Computers* C-34/9 (1985) 869-872.

[61] R.Krishnamurti, E.Ma, "An approximation algorithm for scheduling tasks on varying partition sizes in partitionable multiprocessor systems", *IEEE Transactions on Computers* 41/12 (1992) 1572-1579.

[62] P.Krueger, T.-H.Lai, V.A.Dixit-Radiya, "Job scheduling is more important than processor allocation for hypercube computers", *IEEE Transactions on Parallel and Distributed Systems* 5/5 (1994) 488-497.

[63] M.Kubale, "The complexity of scheduling independent two-processor tasks on dedicated processors", *Information Processing Letters* 24 (1987) 141-147.

[64] M.Kubale, "Preemptive scheduling of two-processor tasks on dedicated processors (in Polish)", *Zeszyty Naukowe Politechniki Śląskiej, Seria: Automatyka* z.100, No.1082 (1990) 145-153.

[65] M.Kubale, "File transfer scheduling within time windows (in Polish)", *Zeszyty Naukowe Politechniki Śląskiej, Seria:Automatyka* z.110, No.1176 (1992) 69-76.

[66] M.Kubale, "Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors", private communication, 1995.

[67] J.F.Lin, S.J.Chen, "Scheduling algorithm for nonpreemptive multiprocessor tasks", *Computers and Mathematics with Applications* 28/4 (1994) 85-92.

[68] E.L.Lloyd, "Concurrent task systems", *Operations Research* 29/1 (1981) 189-201.

[69] E.P.Markatos, T.J.LeBlanc, "Using processor affinity in loop scheduling on shared-memory multiprocessors", *IEEE Transactions on Parallel and Distributed Systems* 5/4 (1994) 379-400.

[70] J.Plehn, "Preemptive scheduling of independent jobs with release times and deadlines on a hypercube", *Information Processing Letters* 34 (1990) 161-166.

[71] F.P.Preparata, G.Metze, R.Chien, "On the connection assignment problem of diagnosable systems", *IEEE Transactions on Electronic Computers* 16/6 (1967) 848-854.

[72] V.J.Rayward-Smith, "UET scheduling with interprocessor communication delays", *Discrete Applied Mathematics* 18 (1987) 55-71.

[73] K.C.Sevcik, "Application scheduling and processor allocation in multiprogrammed parallel processing systems", *Performance Evaluation* 19 (1994) 107-140.

[74] X.Shen, E.M.Reingold, "Scheduling on a hypercube", *Information Processing Letters* 40 (1991) 323-328.

[75] G.N.Srinivasa Prasanna, B.R.Musicus, "Generalized multiprocessor scheduling for directed acyclic graphs", Proceedings of Supercomputing 1994, IEEE Press, 1994, 237-246.

[76] B.Veltman, B.J.Lageweg, J.K.Lenstra, "Multiprocessor scheduling with communication delays", *Parallel Computing* 16 (1990) 173-182.

[77] V.G.Vizing, "About schedules observing deadlines (in Russian)", *Kibernetika* No. 1 (1981) 128-135.

[78] V.G.Vizing, "Optimal choice of task execution intensities with a convex penalty function for intensity (in Russian)", *Kibernetika* No. 3 (1982) 125-127.

[79] V.G.Vizing, "Minimization of the maximum delay in servicing systems with interruption", *U.S.S.R. Computational Mathematics and Mathematical Physics* 22/3 (1982) 227-233.

[80] V.G.Vizing, L.N.Komzakowa, A.V.Tarchenko, "An algorithm for selecting the processing intensity (in Russian)", *Kibernetika* No. 5 (1981) 71-74.

[81] Q.Wang, K.H.Cheng, "List scheduling of parallel tasks", *Information Processing Letters* 37 (1991) 291-297.

[82] Q.Wang, K.H.Cheng, "A heuristic of scheduling parallel tasks and its analysis", *SIAM Journal on Computing* 21/2 (1992) 281-294.

[83] J.Węglarz, "Scheduling under continuous performing speed vs. resource amunt activity models", in R.Słowiński, J.Węglarz, *Advances in Project Scheduling*, Elsevier Science Publisher B.V., Amsterdam, 1989, 273-295.

[84] J.Zahorjan, E.D.Lazowska, D.L.Eager, "The effect of scheduling discipline on spin overhead in shared memory parallel systems", *IEEE Transactions on Parallel and Distributed Systems* 2/2 (1991) 180-198.

[85] Y.Zhu, M.Ahuja, "On job scheduling on a hypercube", *IEEE Transactions on Parallel and Distributed Systems* 4/1 (1993) 62-69.

Table 1. Scheduling multiprocessor tasks on parallel processors

Problem	Result	Reference
	Nonpreemptive scheduling	
$P \mid size_j, p_j = 1 \mid C_{max}$	**NP**h	[68]
$P3 \mid size_j, p_j = 1, prec \mid C_{max}$ and $size_j \in \{1, 2\}$	**NP**h	[68]
$P \mid size_j, p_j = 1, prec \mid C_{max}$	$S_{LS} = \frac{2m-\Delta}{m-\Delta+1}$	[68]
$P2 \mid size_j, p_j = 1, prec \mid C_{max}$	$O(n^2)$	[68]
$P \mid size_j, p_j = 1 \mid C_{max}$ and $size_j \in \{1, \Delta\}$	$O(n)$	[14]
$P \mid size_j, p_j = 1 \mid C_{max}$ and $size_j \in \{1, \dots, \Delta\}$	$O(n)$ ILP	[14]
$P \mid size_j, p_j = 1 \mid C_{max}$	s**NP**h	[14]
$P2 \mid size_j \mid C_{max}$, $P3 \mid size_j \mid C_{max}$	**NP**h,pseudopoly.	[47]
$P4 \mid size_j \mid C_{max}$	**?**	[47]
$P5 \mid size_j \mid C_{max}$	s**NP**h	[47]
$P2 \mid size_j, chain \mid C_{max}$	s**NP**h	[47]
$P \mid size_j \mid C_{max}$	$S_{LPT} \leq \frac{4\Delta}{3} - \frac{\Delta(\Delta+1)}{6m}$	[67]
$P4 \mid size_j \mid C_{max}$ and $size_j \neq 1$	pseudopoly.	[44]
$P \mid cube_j \mid C_{max}$	$S_{LDLPT} = 2 - \frac{2}{m}$	[32]
$P \mid cube_j \mid C_{max}$	$S_{LDF} = 2 - \frac{1}{m}$	[85]
$P \mid cube_j \mid \sum C_j$	experimental study	[62]
$P \mid cube_j \mid C_{max}$	$S_{LPT} \leq 2 - \frac{1}{m}$	[67]
$P \mid spdp\text{–}lin\text{–}\delta_j, prec \mid C_{max}$	$S_{LS} = \Delta + \frac{m-\Delta}{m}$	[81]
$P \mid spdp\text{–}lin\text{–}\delta_j, prec \mid C_{max}$	$S_{ECT} < \ln \Delta + 2$	[81]
$P \mid spdp\text{–}lin\text{–}\delta_j, prec \mid C_{max}$	$S_{ECT} < 3 - \frac{2}{m}$	[82]
$P \mid spdp-any, var, r_j \mid L_{max}$	continuous processor medium	[83]
$P \mid spdp\text{–}any \mid C_{max}$ and $n \leq m$	$S_{VS} = \min\{n, \frac{R}{1-\frac{n}{m}}\}$	[61]
$Q, win \mid spdp\text{–}any \mid C_{max}$ and $n = 1$	$O(m^2)$	[2]
$P \mid spdp\text{–}lin \mid \sum C_j$	SPT is optimal	[73]
$P \mid spdp\text{–}any \mid \sum C_j$	special cases analyzed	[73]
$P \mid spdp\text{–}any, prec \mid C_{max}$	heuristic	[35]
$P \mid spdp\text{–}any, prec \mid C_{max}$	$O(e^2 + en + I(e + n))$	[75]
$Q, win \mid spdp\text{–}any \mid C_{max}$ and $n = 1$	$O(m \log m)$	[48]

Table 1. continued

Problem	Result	Reference
Preemptive scheduling		
$P \mid size_j, pmtn \mid C_{max}$ and $size_j \in \{1, \Delta\}$	$O(n)$	[14]
$Pm \mid size_j, pmtn \mid C_{max}$	LP	[14]
$P \mid size_j, pmtn, res1 \cdot 1 \mid C_{max}$ and $size_j \in \{1, 2\}$	$O(n \log n)$	[20]
$Q \mid size_j, pmtn \mid C_{max}$ and $size_j \in \{1, 2\}$	$O(n \log n + nm)$	[16]
$Q \mid size_j, pmtn \mid C_{max}$ and $size_j \in \{1, \Delta\}$	$O(n \log n + nm)$	[17]
$Qm \mid size_j, pmtn \mid C_{max}$	LP	[18]
$Pm \mid size_j, pmtn \mid L_{max}$ and $size_j \in \{1, \Delta\}$	LP or tabu search+LP	[19]
$P \mid size_j, pmtn, res1 \cdot 1 \mid C_{max}$ and $size_j \in \{1, \Delta\}$	$O(nm)$	[21]
$P \mid size_j, pmtn \mid C_{max}$	**NP**h, ?	[44]
$P \mid cube_j, pmtn \mid C_{max}$	$O(n^2(\log n + + \log \max_j\{t_j^{size_j}\}))$	[31]
$P \mid cube_j, pmtn \mid C_{max}$	$O(n \log n(\log n + + \max_j\{t_j^{size_j}\}))$	[54, 1]
$P \mid cube_j, pmtn, r_j \mid L_{max} = 0$	LP	[70]
$Q \mid cube_j, pmtn \mid C_{max}$	$O(n \log n + nm)$	[18, 43]
$P \mid cube_j, pmtn \mid C_{max}$	$O(n^2 m^2)$	[74]
$P \mid cube_j, pmtn \mid C_{max}$	$O(n^2 \log^2 n)$	[85]
$R \mid spdp\text{–}lin, var, r_j \mid L_{max} = 0$	LP	[77]
$P \mid spdp\text{–}lin, var, r_j \mid X$	$O(n^2)$	[80]
$P \mid spdp\text{–}lin, var, r_j \mid X$	$O(n^2)$	[78]
$P \mid spdp\text{–}lin\text{–}\delta_j, var, r_j \mid L_{max}$	$O(n^4 m)$	[79]
$Pm \mid spdp\text{–}any, pmtn \mid C_{max}$	**NP**h,pseudopoly.	[47]
$P \mid spdp\text{–}any, pmtn \mid C_{max}$	s**NP**h	[47]
$P \mid spdp\text{–}lin\text{–}\delta_j, var, chain \mid C_{max}$ and $\mid chain \mid = 3$	**sNP**h, $S_{H1} = 3$ $S_{H2} = S_{H3} = 2$	[46]
$Pm \mid spdp\text{–}lin\text{–}\delta_j, var, chain \mid C_{max}$ and $\mid chain \mid = 3$	polynomially solvable	[46]
$P \mid spdp\text{–}lin\text{–}\delta_j, var, chain \mid C_{max}$ and $\mid chain \mid = 2$	$O(n \log n)$	[46]
$P \mid spdp\text{–}lin\text{–}\delta_j, var, r_j \mid C_{max}$	$O(n^2)$	[45]

Table 2. Scheduling multiprocessor tasks on dedicated processors

Problem	Result	Reference
	Nonpreemptive scheduling	
$P \mid fix_j \mid C_{max}$	B&B	[25]
$P \mid fix_j \mid C_{max}$ and $\mid fix_j \mid = 2$	**NP**h, $S_{LPT} = \frac{4(d-1)}{d}$	
	$S_{LPT} \leq 3$ when $d \leq 5$	
	$S_{LPT} < 2$ binomial s.g.	[60]
$P \mid fix_j \mid C_{max}$ and $\mid fix_j \mid = 2$	20 cases **NP**h	
	23 cases polynomial	
	$\frac{4}{3} < S_{LS} \leq 3$	
	$S_{LPT} = \frac{5}{2} - \frac{1}{p}$	
	$C_{max}^{DP1} \leq 3C_{max}^{*} + e\varepsilon$	[38]
$P \mid fix_j \mid C_{max}$ and $\mid fix_j \mid \in \{1, 2\}$	9 cases **NP**h	
	9 polynomial cases	[63]
$P \mid fix_j \mid C_{max}$		
$P \mid fix_j, p_j = 1 \mid C_{max}$	experimental study	[40]
$P2 \mid fix_j \mid \sum w_j C_j$	$O(n \log n)$	[42]
$P \mid fix_j \mid \sum w_j C_j$	ILP+experiment	[42]
$P3 \mid fix_j \mid C_{max}$	s**NP**h,	[13, 55]
	$S_{NS} < \frac{4}{3}$	[13]
	$S_{NS} = \frac{5}{4}$	[41]
	$S_{LPT} = S_{SPT} = 3$	[41]
$O \mid fix_{ij} \mid C_{max}$ and $stages = 2$	$O(n)$	[29]
$O \mid fix_{ij} \mid C_{max}$ and $stages = 3$	**NP**h	[29]
$O \mid fix_{ij}, p_{ij} = 1 \mid C_{max}$		
and $stages = r$	polynomial	[29]
$F \mid fix_{ij} \mid C_{max}$ and $stages = 2$	$O(n \log n)$	[29]
$F2 \mid fix_{ij} \mid C_{max}$ and $stages = 3$	s**NP**h	[29]
$J2 \mid fix_{ij}, p_{ij} = 1 \mid C_{max}$	s**NP**h	[29]
$J2 \mid fix_{ij} \mid C_{max}$ and $n_j \leq 2$	$O(n \log n)$	[29]
$J \mid fix_{ij} \mid C_{max}$ and $n = 2$	$O(n^2 \log n)$	[29]
$J2 \mid fix_{ij} \mid C_{max}$ and $n = k$	$O(n^{3k})$	[29]

Table 2. continued

Problem	Result	Reference
$Pm \mid fix_j, p_j = 1 \mid f$ and $f \in \{\sum w_jU_j, \sum T_j, \sum w_jC_j\}$ and R number of task types	$O(R2^Rn^{R+1} + 2^R(R+m))$	[28]
$Pm \mid fix_j, p_j = 1, r_j \mid f$ and $f \in \{C_{max}, \sum C_j\}$ and R number of task types	$O(R2^Rn^{R+1} + 2^R(R+m))$	[28]
$J \mid fix_{ij}, p_{ij} = 1, prec, r_j \mid f$ and $n = k, f \in \{max f_j, \sum f_j\}$ and f_j nondecreasing function of C_j	$O(k2^km \sum_{j=1}^{k} n_j \prod_{j=1}^{k} n_j)$	[28]
$F \mid fix_{ij}, p_{ij} = 1 \mid f$ and $stages = r$ and $f \in \{\sum w_jC_j, \sum T_j, \sum w_jU_j\}$	$O(r^2 2^r n^{r+2} + 2^r(r+m))$	[28]
$F \mid fix_{ij}, p_{ij} = 1, r_j \mid f$ and $stages = r, f \in \{\sum C_j, C_{max}\}$	$O(r^2 2^r n^{r+2} + 2^r(r+m))$	[28]
$O \mid fix_{ij}, p_{ij} = 1 \mid f$ and $stages = r$ and $f \in \{\sum w_jC_j, \sum T_j, \sum w_jU_j\}$	$O(r^3(r!)^2 2^r n^{r!(r+1)+1} +$ $+2^r(r+m))$	[28]
$O \mid fix_{ij}, p_{ij} = 1, r_j \mid f$ and $stages = r$ and $f \in \{C_{max}, \sum C_j\}$	$O(r^3(r!)^2 2^r n^{r!(r+1)+1} +$ $+2^r(r+m))$	[28]
$O \mid fix_{ij}, p_{ij}, prec \mid f$ and $n = 2, stages = r$ and $f \in \{max f_j, \sum f_j\}$ and f_j nondecreasing function of C_j	$O(r^{2.5})$	[28]
$P2,3,4 \mid fix_j, p_j = 1 \mid C_{max}$	$O(n)$	[11]
$P5 \mid fix_j, p_j = 1 \mid C_{max}$	$O(n^{2.5})$	[11]
$P \mid fix_j \mid C_{max}$	B&B	[11]
$Pm \mid fix_j, p_j = 1 \mid C_{max}$	ILP	[55]
$P \mid fix_j, p_j = 1 \mid C_{max}{=}3$	s**NP**h	[55]
$P2 \mid fix_j, p_j = 1, chain \mid C_{max}$	s**NP**h	[55]
$P2 \mid fix_j, p_j = 1, r_j \mid C_{max}$	s**NP**h	[55]
$Pm \mid fix_j, p_j = 1, r_j \mid C_{max}$	ILP	[55]
$P2 \mid fix_j \mid \sum C_j$	**NP**h	[55]
$P3 \mid fix_j \mid \sum C_j$	s**NP**h	[55]
$P2 \mid fix_j \mid \sum w_jC_j$	s**NP**h	[55]

Table 2. continued

Problem	Result	Reference
$P \mid fix_j, p_j = 1 \mid \sum C_j$	s**NP**h	[55]
$P2 \mid fix_j, p_j = 1, chain \mid \sum C_j$	s**NP**h	[55]
$P, win \mid fix_j \mid C_{max}$	**NP**h, 3 cases polynomial	[65]
$P2 \mid fix_j, p_j = 1 \mid L_{max}$	$O(n)$	[8]
$P3 \mid fix_j \mid C_{max}$	$S_{18} = \frac{7}{6}$	[51]
$P2 \mid fix_j \mid L_{max}$	s**NP**h	[44]
$P3 \mid fix_j \mid C_{max}$ and $\mid fix_j \mid = 2$	$O(n)$	[66]
$P3 \mid fix_j, chain \mid C_{max}$		
and $\mid fix_j \mid = 2$	$O(n)$	[66]
$P4 \mid fix_j, p_j = 1, chain \mid f$		
and $\mid fix_j \mid = 2$ and $f \in \{C_{max}, \sum C_j\}$	s**NP**h	[66]
$P \mid fix_j, p_j = 1 \mid \sum C_j$ and $\mid fix_j \mid = 2$	s**NP**h	[66]
$P3 \mid fix_j, chain \mid \sum C_j$ and $\mid fix_j \mid = 2$	$O(n \log n)$	[66]
$P4 \mid fix_j \mid \sum C_j$ and $\mid fix_j \mid = 2$	**NP**h	[66]
$P \mid fix_j \mid \sum C_j$		
and $\mid fix_j \mid = 2$ and s.g. is 2-star	**NP**h	[66]
$P2 \mid set_j \mid C_{max}$	pseudopoly.	[10]
$P3 \mid set_j \mid C_{max}$		
and $\forall_{T_j} fix_j \neq \{P_1, P_3\}$	pseudopoly.	[10]
$P \mid set_j \mid C_{max}$	$S_{SPTM} = m$	[10]
$P \mid set_j \mid C_{max}$	s**NP**h, heurisctic	[12]
Preemptive scheduling		
$P \mid fix_j, pmtn \mid C_{max}$		
and $\mid fix_j \mid = 2$	s**NP**h	[64]
$Pm \mid fix_j, pmtn \mid C_{max}$		
and $\mid fix_j \mid = 2$	LP	[64]
$P2,3,4 \mid fix_j, pmtn \mid C_{max}$	$O(n)$	[7]
$P4 \mid fix_j, pmtn, res1 \cdot 1 \mid C_{max}$	$O(n)$	[7]
$P2 \mid fix_j, pmtn \mid L_{max}$	$O(n)$	[6]
$P3,4 \mid fix_j, pmtn \mid L_{max}$	experimental study	[6]
$P2, win \mid fix_j, pmtn \mid C_{max}$		
and p number of time windows	$O(n+p)$	[9]
$P2, win \mid fix_j, pmtn, r_j \mid C_{max}$	$O(n+p)$	[9]
$P3, win \mid fix_j, pmtn \mid C_{max}$	$O(n+p)$	[9]

Table 2. continued

Problem	Result	Reference
$P \mid fix_j, pmtn \mid C_{max}$ and $\mid fix_j \mid = 2$ s.g. bipartite, unicyclic	$O(n^2)$	[66]
$P \mid fix_j, pmtn \mid C_{max}$ and $\mid fix_j \mid = 2$ s.g. candy,caterpillar	$O(n)$	[66]
$P4 \mid fix_j, pmtn, chain \mid f$ and $\mid fix_j \mid = 2$ and $f \in \{C_{max}, \sum C_j\}$	s**NP**h	[66]
$P \mid fix_j, pmtn \mid \sum C_j$ and $\mid fix_j \mid = 2$	s**NP**h	[66]
$P3 \mid fix_j, pmtn, chain \mid \sum C_j$ and $\mid fix_j \mid = 2$	$O(n \log n)$	[66]
$P \mid fix_j, pmtn \mid C_{max}$ and $\mid fix_j \mid = 2$ and s.g. 2-star,superstar	$O(n \log n)$	[66]
$Pm \mid set_j, var \mid C_{max}$	LP	[10]
$Pm \mid set_j, var, r_j \mid L_{max}$	LP	[5]
$Pm, win \mid set_j, var, r_j \mid C_{max}$	LP	[9]
$Pm, win \mid set_j, var, r_j \mid L_{max}$	LP	[9]