MULTIPROCESSOR SCHEDULING THEORY. A SHORT SURVEY

MERCIER, Gilles
ICARE Research Team,
IUT Blagnac, France
gilles.mercier@iut-blagnac.fr

VÎLCU, Dana
Faculty of Mathematics-Informatics
Spiru Haret University
d.vilcu.mi@spiruharet.ro

Abstract

We survey the main problems induced by the real time multiprocessor scheduling, particularly emphasising the independent task model. Properties of scheduling independent tasks and implementation restrictions are presented, as well as their consequences. The relationship with the bin-packing problem is also considered.

Keywords: *scheduling, multiprocessing, on-line algorithm, complexity*

AMS classification: 90B35, 68W27, 68W10

Computing Reviews categories: D.4.1

1. Introduction

The articles on real time multiprocessor scheduling are generally based on the assumption that the processors are identical and share the same memory. Nevertheless, the scheduling theory makes clear the distinction between at least three classes of different multiprocessors machines [6]. The most general class is that of *parallel machines without relation*: for every pair (T_i, p_j), with T_i a task and p_j a processor, the associated rate r_{ij} shows that the task T_i, while run t time units on processor p_j, completes by $r_{ij}t$ times its execution time. A second class consists of *uniform parallel machines*: every processor is characterised by its own computing capacity s (a task requiring t CPU cycles will run in st time units). The most restricted class contains *identical parallel machines*: machines with several identical processors (having the same computing capacity).

The multiprocessor scheduling proves particularly difficult, due to the complexity of its problems, but also due to lack of practical experience in the field. The main results presented here show this. On the other hand, there are more and more multiprocessor real time systems, and the use of multiprocessor SoC platforms is particularly interesting for embedded systems from the viewpoint of (significant) power savings, see [16].

This survey presents different scheduling techniques for parallel identical machines with shared memory, the usual case treated by the scheduling theory. Similarly to the monoprocessor case presented in [17], the independent tasks are emphasised. The first two types of parallel machines are particularly useful for our studies treating the scheduling and the power consumption at processor(s) level [15], [16], but they are not considered in this survey.

A few papers were particularly helpful to gather the results in this article. First of all, Stankovic et al. [14] gives good ideas on the main topics in this field and indicates well-chosen references to present the corresponding results. The very concise paper of Dertouzos [4] introduces the basic results on the scheduling of independent tasks on multiprocessor machines and so furnishing the manner to study the other models of tasks. Other categories of useful papers are those referring to properties of different scheduling algorithms, as for example [10] or [1], and those treating the complexity of the problems, see [2], [5], [8].

In Section 2 we recall the basic notions about tasks and scheduling. Properties of scheduling independent tasks are presented in Section 3 and are followed, in Section 4, by restrictions imposed by the implementation and their consequences. Section 5 indicates another approach to solve the multiprocessor scheduling problem, the bin-packing problem. The conclusions, in Section 6, resume the topics covered by this survey.

2. **Basic notions about tasks and scheduling**

The *tasks* (sometimes also called *jobs*) are the basic entities of real time scheduling problems, and they are *periodic* or *a-periodic*, with *hard* or *soft* temporal constraints [3].

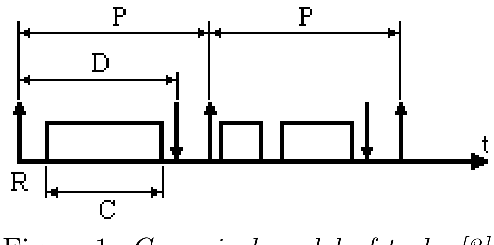

Figure 1. *Canonical model of tasks [3].*

The specific parameters of the canonical model of tasks are illustrated in Figure 1, where the symbols represent: R - the starting (activation) moment of the task execution, C - the maximal computation time to complete the task, also called worst-time execution, D - the deadline of the task execution, which can be *hard* or *soft* (depending on the importance of the task in the system, or on the type of the application), P - the period of periodical tasks. The deadline is *under request* if it coincides with the periodicity of the task. An index $_i$ will indicate that the parameters refer to the task T_i.

Some constraints complete these task parameters in order to define different task models: precedence relations between tasks, resource sharing, preemption, dependence, external priority, emergency, and maximal deadline.

A task is *preemptible* if, after starting its execution, it can be stopped and its state becomes ready, in order to give the processor to another task considered more important. If the execution of a task cannot be interrupted then the task is called *non-preemptible.*

For a set of n periodical tasks, the following two parameters are very important:
- the processor utilisation (usability) factor, $U = \sum_{i=1}^{n} \frac{C_i}{P_i}$, and
- the processor(s) charge, $CH = \sum_{i=1}^{n} \frac{C_i}{D_i}$.

We also define:
- the laxity $L(t)$ of a task at the moment t, as the maximum delay relative to the deadline, in order to (re)start its execution, when the task runs alone: $L(t) = D(t) - C(t)$, and
- the lateness of the task T, whose execution ends in C_{real} units of time, as: $L = C_{real} - D$.

The *scheduling strategies* are the rules to select the next task to be executed in a multitasking system. In other words, these rules decide the schedule of the tasks, attempting to optimally exploit the most important resource of the system, the processor. At the level of the operating system's kernel, these strategies form the scheduler. The scheduling algorithm determines, according to the implemented policy, the effective order of the task execution, called *the schedule*, or *the task scenario.*

A scheduling of a real time task set is *feasible with respect to a scheduling S* if the deadlines of all tasks are respected when ordered by S. A task set is *schedulable* if there exists a feasible scheduling for it.

We add the functioning principle of two scheduling algorithms.

Earliest Deadline First (EDF) allocates, at the moment t, the highest priority to the task with the closest deadline to t.

Least Laxity First (LLF) allocates, at the moment t, the highest priority to the task having the smallest laxity. The computation of laxities can be done either at the moment of returning to activity of the tasks, in which case the resulted sequence is the same as that produced by the EDF algorithm, or at any moment (in the case of many context switches).

3. **Properties of scheduling independent tasks**

We start our presentation by highlighting the necessary knowledge on the task set allowing to know the existence of a scheduling algorithm, optimal for the given task set. Since the notion of deadline is particularly important, the main results concern algorithms based on task deadlines.

Theorem 1. [4], [10] (The problem of insufficient knowledge) *For any multiprocessor machine, no deadline scheduling algorithm can be optimal with-*

out complete a priori knowledge of the activation moments, computation times, and deadlines of the tasks.

Having this said, the a priori knowledge of these three most important task parameters necessarily leads to off-line studies and has a direct connection between the task set and the employed algorithm. The classical scheduling algorithms, requiring the task activation moments, cannot be optimal if used on-line, and one cannot hope to find an optimal algorithm for the general case.

In the multiprocessor case, it would be very useful to have necessary and/or sufficient conditions for the feasibility of a task set, and to know properties of algorithms with respect to different task models. The analysis is still far from the concrete and concise results in the monoprocessor case. For example, there is no(t yet found a) condition meanwhile necessary and sufficient, not even for the periodical independent tasks, with deadline under request. Below we present the main results.

Theorem 2. [4] *For a set of periodical independent tasks, with deadlines under request, a necessary scheduling condition on a machine with m processors is $U \leq m$.*

Theorem 3. [4] *For a set of periodical independent tasks with deadlines under request and respecting the necessary feasibility condition $U \leq m$, there exists a feasible scheduling if $D\frac{C_i}{D_i} \in \mathbb{N}$, for all $i = 1, ..., n$, where $D = \gcd(D_1, ..., D_n)$.*

Here, the notation "gcd" stands for the *greatest common divisor* of the set of numbers following it between the parentheses.

Enlarging the study toward sets of arbitrary independent tasks, the results are even more complex, as shown by the next two theorems, giving a necessary condition (Theorem 4) and information on the necessary knowledge for the task set (Theorem 5) in order to simplify the scheduling problem and to allow a priori estimates on the feasibility.

It must be noted that, according to the way the scheduling theory was developed, these results are based on the assumption that the employed machine is either monoprocessor with constant speed, or multiprocessor with identical processors, their number being fixed at the beginning, and having all the same running speed, constant during the tasks execution.

Next we define a function F to measure the excess of computation power in the system for the next k time units. For every $k \in \mathbb{N}^*$, let $G_1(k) = \{T_i | D_i \leq k\}$, $G_2(k) = \{T_i | L_i \leq k \text{ and } D_i > k\}$, and

$$F(k) = km - \sum_{i \in G_1} C_i - \sum_{i \in G_2} (k - L_i) \, .$$

Theorem 4. [4] (Necessary condition for scheduling a set of independent tasks on m identical processors) *A necessary condition for a scheduling to meet the deadlines of a set of independent tasks whose activation moments are the same (at time $i = 0$) is that for all $k > 0$, $F(k) \geq 0$.*

Theorem 5. [4] *If a schedule exists which meets the deadlines of a set of independent tasks whose activation moments coincide, then the same set of tasks can be scheduled on-line with the Least Laxity algorithm even if their activation moments are different and not known a priori. Knowledge of only the deadlines and computation times suffices for scheduling.*

This last theorem provides two important conclusions.

On one hand, it allows to solve the feasibility problem for a set of independent tasks for which the activation moments are different and not known a priori, on a machine with m processors, by reducing it to the feasibility problem on the same machine for the same task set, but having the same activation moment. For this reason, without any loss of generality, whenever the feasibility of a set of independent tasks is searched for, the tasks will be considered with the same moment of activation.

On the other hand, Theorem 5 indicates sufficient knowledge allowing to answer the feasibility problem for a set of independent tasks on a machine with a given (fixed) number of processors. (LLF is indicated as the scheduling algorithm assuring the feasibility.)

The following theorems show the difficulty to find a non-preemptive scheduling for a set of independent tasks with the same deadline. For independent arbitrary tasks one can figure out that the situation has at least the same complexity.

Theorem 6. *Consider the non-preemptive multiprocessor scheduling problem with* 2 *processors, no resources, for independent tasks.*
a) [2] *For unit computation times this problem is polynomial.*
b) [5] *For computation times i) arbitrary, or ii) either* 1 *or* 2 *units of time, this problem is NP-complete.*

Theorem 7. [5] *The non-preemptive multiprocessor scheduling problem with at least* 3 *processors, one resource, for independent tasks with equal computation times, is NP-complete.*

Theorem 8 shows an example where, for a certain metric, the preemption brings no advantage. In the general case, finding an algorithm allowing preemption remains an NP-hard problem (Theorem 9), which means that for solving this problem one needs heuristics.

Theorem 8. [9] *For any instance of the multiprocessing scheduling problem with m processors, preemption allowed, and minimizing the weighted sum of completion times, there exists a schedule with no preemption for which the value of the sum of computation times is as small as for any schedule with a finite number of preemptions.*

Theorem 9. [8] *The multiprocessing problem of scheduling with m processors, preemption allowed, and minimizing the number of late tasks, is NP-hard.*

The results on dynamic scheduling are not many. One tried to transfer monoprocessor scheduling results to multiprocessors, concerning the optimality of algorithms and, as we have already seen with Theorem 1, it is necessary to have complete knowledge on tasks characteristics. That result, as well as Theorem 4, is completed by:

Theorem 10. [10] *Earliest Deadline algorithm is not optimal in the multiprocessor case.*

To illustrate this statement, Figure 2 shows an example (adapted from [14]) when the scheduling of the same task set on 2 processors fails with EDF, but is feasible with LLF. The task set is given by three periodic tasks with deadlines under request: $T_1(R_1 = 0, C_1 = 1, D_1 = 2)$, $T_2(R_2 = 0, C_2 = 1, D_2 = 2)$ and $T_3(R_3 = 0, C_3 = 3, D_3 = 3)$, and is the task T_3 which fails to meet its deadline. Notice that the necessary feasibility condition in Theorem 2 is respected, but (of course) this is not the case with the sufficient condition in Theorem 3.

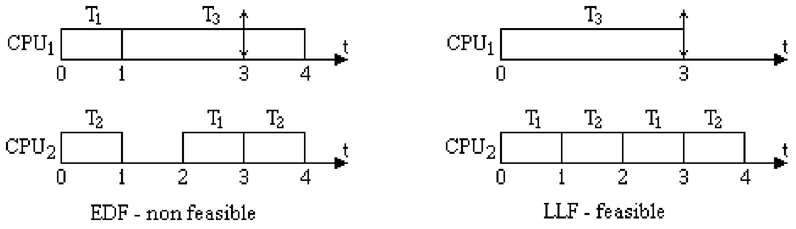

Figure 2. *EDF and LLF for multiprocessor scheduling.*

For all these theoretical reasons, but also because of the difficulty to realize precise worst-case analyses for different task models, the studies were directed toward probabilistic heuristics and stochastic analyses, see [12], [18]. However, those analyses are expensive if realized on-line, and often require the use of supplementary material, namely a scheduling chip. We will not present them here.

4. **Implementation restrictions and consequences**

When implementing an application on a multiprocessor machine, we encounter similar problems to the monoprocessor case, even for independent tasks, to mention only sharing the system resources or the overload.

According to [14], many researchers consider that the techniques to solve the sharing resource problem for a monoprocessor system cannot be applied to multiprocessor systems. For such systems, the sharing resources are usually addressed by on-line scheduling algorithms, see [12], [13], [18].

Concerning the system overload, for sporadic tasks with preemption allowed, on a machine with two processors, *the value of a task* is defined as its execution time if the deadline is met, and zero otherwise; we have:

Theorem 11. [1] *No on-line scheduling algorithm can guarantee a cumulative value larger than one-half for the dual processor case.*

Comparing to the monoprocessor case, extra complications may appear in the multiprocessor scheduling problem, the so called *Richard's anomalies.* The following theorem applies to task sets with precedence constraints, but clearly it also applies to the case of independent tasks with shared resources, where we present two examples.

Theorem 12. [7] *Consider a task set with precedence conditions and fixed computation times, optimally scheduled according to some priority criterion, on a multiprocessor machine. Changing the priority list, increasing the number of processors, reducing the execution times, or weakening the precedence constraints, can increase the schedule length.*

The first example is adapted from [14] and treats the reducing of the computation time of tasks. This seems to be the most frequent situation, because the scheduling analyses are based on worst-case estimates for the computation time of tasks. Our tasks are statically allocated, the first two on the first processor, and the last three on the second processor. One can see, on the right side of Figure 3, a decreased computation time for the task T_1. Therefore, the task T_2, in mutual exclusion with the task T_4, is launched sooner. This forces T_4 to start its execution later than planned, and the scheduling sequence becomes longer.

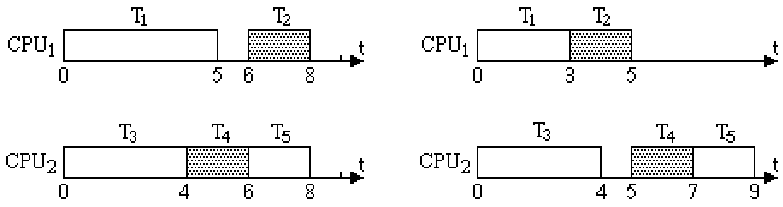

Figure 3. *Richard's anomaly caused by decreasing the execution time of a task.*

Our second example, illustrated by Figure 4, treats the increasing of the processor number. The anomaly is produced again by mutual exclusion. The tasks T_3 and T_4 have each two zones of reciprocal mutual exclusion. Passing from 2 to 3 processors, while keeping the task execution order, yields a longer scheduling time.

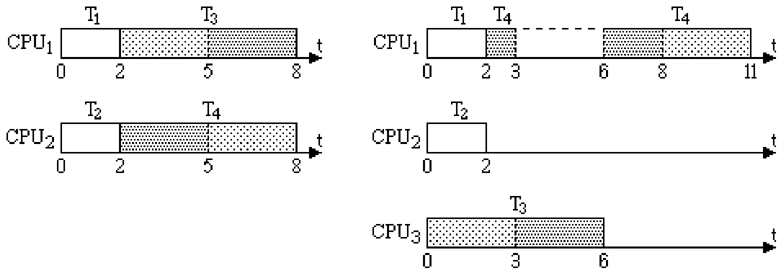

Figure 4. *Richard anomaly caused by increasing the number of processors.*

5. **The** *bin packing problem* **and multiprocessor scheduling**

The problem of running a task set with temporal constraints is generally treated to obtain the scheduling feasibility on a machine with fixed number of processors. Another approach, much less known and studied, is to find the minimal number of necessary processors to feasibly execute the task set. This approach leads, in a first stage, to the *bin packing* algorithmic problem. According to certain order (given by the tasks characteristics), the tasks are allocated one after the other and a processor (the bin) is allocated to them if it can meet the task constraints (deadline). This task allocation to processors can be done by several arranging algorithms: Next Fit - NF (if the current processor cannot meet the new task constraints, the next processor is considered), First Fit - FF (if the current processor cannot meet the new task constraints, the processors list is scanned, starting with the first processor, until a processor is found to assure the task deadline), Best Fit - BF (for the task to be scheduled, the processors list is completely scanned to find, according to a certain criterion, the best processor to guarantee the task deadline), or First Fit Decreasing - FFD or Best Fit Decreasing - BFD (versions of FF and BF where the tasks are ordered in a list according to their non-increasing execution times, before applying the arranging algorithm). These arranging algorithms can also be combined with the usual scheduling algorithms [11].

Let M be the minimal number of necessary processors to feasibly run a set of independent tasks with the same deadline.

Theorem 13. [7], [11] *For a large number of tasks, the worst-case bounds for the processor number is* $17M/10$ *for FF and BF,* $11M/9$ *for FFD, and the bound for BFD is not larger than that for FFD. The bound for RM-FD is at least* 1.66 *larger than that given by a clairvoyant optimal algorithm.*

The idea of increasing the processor number was also applied in [16], to provide a necessary and sufficient condition for scheduling a task set on a multiprocessor machine with variable speed processors, optimally from the power consumption viewpoint.

6. **Conclusions**

This paper is a short survey on the main results of multiprocessor real time scheduling for the models of independent tasks, and of independent and periodic tasks. The subject concerns the feasibility of the task models, the optimality of the scheduling algorithms and the complexity of the different problems. A special approach, aiming to determine the number of processors to be used in the schedule of a task set, known as the bin-packing problem, is also mentioned.

The goal was to present notions and results very useful –if not indispensable– for understanding the multiprocessor scheduling of other task models, and also for the multiprocessor power aware scheduling theory.

References

1. Baruah, S., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D., and Wang, F., *On the Competitiveness of On-Line Real-Time Task Scheduling*, "Proc. Real-Time Syst. Symp.", 106-115, 1991.
2. Coffman, E. G., and Graham, R., *Optimal Scheduling for Two-Processor Systems*, "ACTA Informatica" 1, 200-213, 1972.
3. Cottet, F., Delacroix, J., Kaiser, C., and Mammeri, Z., *Ordonnancement temps réel*, Éditions Hermes, 2000.
4. Dertouzos, M. L., and Mok, A. K., *Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks*, "IEEE Trans. Soft. Eng." 15, 1497-1506, 1989.
5. Garey, M. R., and Johnson, D. S., *Complexity Results for Multiprocessor Scheduling under Resource Constraints*, "SIAM Journal of Computing" 4, 397-411, 175.
6. Goossens, J., Baruah, S. and Funk, S., *Real-Time Scheduling on Multiprocessors*, Proceedings of the 10th international conference on real-time systems, "RTS Embedded Systems", Paris, France, 189-200, 2002.
7. Graham, R., *Bounds on the Performance of Scheduling algorithms*, in "Computer and Job Shop Scheduling Theory", John Willey and Sons, 165-227, 1976.
8. Lawler, E. L., *Recent Results in the Theory of Machine Scheduling*, in "Mathematical Programming: the State of the Art", A. Bachen and al. (eds.), Springer Verlag, New York, 202-234, 1983.
9. McNaughton, R., *Scheduling With Deadlines and Loss Functions*, "Management Science" 6, 1-12, 1959.
10. Mok, K., *Fundamental Design Problems for the Hard Real-Time Environments*, MIT Ph.D. Dissertation, 1983.
11. Oh, Y., and Son, S. H., *Allocating Fixed-priority Periodic Tasks on Multiprocessor Systems*, "Journal of Real-Time Systems" 9, 207-239, 1995.
12. Ramamritham, K., Stankovic, J., and Shiah, P., *Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems*, "IEEE Transactions on Parallel and Distributed Computing" 1, 184-194, 1990.
13. Shen, C., Ramamritham, K., and Stankovic, J., *Resource Reclaiming in Multiprocessor Real-Time Systems*, "IEEE Transactions on Parallel and Distributed Computing" 4, 382-397, 1993.
14. Stankovic, J., Spuri, M., Di Natale, M., and Buttazzo, G., *Implications of Classical Scheduling Results for Real-Time Systems*, "IEEE Computer" 28, 16-25, 1995.
15. Vîlcu, D., *Power Aware Scheduling on Variable Speed Processor: Limits of the Theoretical Model*, in Păltineanu, G. et al. (eds.), Proc. Intl. Conf., "Trends and challenges in applied mathematics", ICTCAM 2007, Bucharest, Romania, Ed. Matrix Rom., 361-365, 2007.
16. Vîlcu, D., *Real Time Scheduling and CPU Power Consumption in Embedded Systems*, in Miclea, L. et al. (eds.), Proc. 2008 IEEE Intl. Conf. on "Automation, Quality and Testing, Robotics", AQTR 2008, Cluj-Napoca, Romania, Vol. I, 261-266, 2008.

17. D. Vîlcu, *An Overview on Real Time Scheduling*, “Revista Română de Automatică” XXII, 56-64, 2009.
18. Zhao, W., Ramamritham, K. and Stankovic, J., *Preemptive Scheduling Under Time and Resource Constraints*, “Special Issue of IEEE Transactions on Computers on Real-Time Systems” 36, 946-960, 1987.