Non-preemptive multiprocessor scheduling of strict periodic systems with precedence constraints

Liliana Cucu, Yves Sorel

INRIA Rocquencourt,
BP 105 - 78153 Le Chesnay Cedex, France
liliana.cucu@inria.fr, yves.sorel@inria.fr

November 18, 2004

Abstract

We evoke our model of strict periodic systems with precedence constraints and our model of architecture. We prove that the problem of multiprocessor scheduling of these systems is NP-hard. We give some theoretical results which allows us to propose a heuristic to solve this problem and we compare its performances to those of an exact algorithm of type "branch & bound".

1 Introduction

Real-time systems, which are reactive systems [1], interact permanently with the physical environment through sensors and actuators. This behavior is, usually, described using periodic systems of operations. We study here strictly periodic systems because our real-time applications are based on signal, image and control processing algorithms [2]. In this case, operations are always available and the notion of release time has no meaning. We study here non-preemptive case but it is only a starting point to study later the preemptive case taking into account the preemption cost. Indeed, that is necessary because neglecting it leads to an overload of the processors [3].

This paper is composed of six sections. The following section evokes briefly existing results on non-preemptive and preemptive multiprocessor scheduling of periodic systems. The section 3 introduces the model and the problem to solve. The section 4 proves that our problem is NP-hard. It is followed by the section 5 presenting the heuristic and the exact algorithm of type branch & bound, and their compared performances. The paper ends with section 6 which discusses these results.

2 Existing results

Scheduling a task in the multiprocessor case means to assign to the task not only a start time but also a processor on which the task will be executed. The problem of scheduling n tasks on m processors is generally proved NP-hard [4]. Consequently the existing results for these problems are heuristics or pseudo-polynomial algorithms.

Most of these results [5, 6, 7, 8] concerns the problem of multiprocessor scheduling for periodic tasks using the model of Liu and Layland [9]. For the same model enriched with precedence constraints, Ramamritham proposes a heuristic [10]. Korst considers the problem of multiprocessor scheduling of strictly periodic independent tasks [2] and proposes a heuristic. Our paper generalizes this latter problem by adding precedence constraints and proposes a heuristic which has good performance.

3 Model

In order to clearly distinguish the specification level and its associated model from the implementation level, we use the term *operation* instead of the commonly used "task" too closely related to the implementation level, and in turn we use the term *operator* instead of "processor" or "machine".

We use a graph-based model to specify on the one hand the precedences between operations through an algorithm graph [11], and on the other hand to specify the multiprocessor through an architecture graph [12].

The algorithm graph is a directed graph $G_{al} = (V_{al}, E_{al})$ where V_{al} is the set of operations and $E_{al} \subseteq V_{al} \times V_{al}$ the set of edges which represents the precedence constraints possibly due to data transfer between operations (see figure 1). Therefore, the directed pair of operations $(A, B) \in E_{al}$ means that B must be scheduled, only if A was already scheduled. Moreover if there is a data transfer, then A produces it while B consumes it. This is noted by adding a d located on the arrow representing the oriented edge (see the data transfer between A and C in figure 1). We say that an operation is available when all its predecessors are already scheduled.

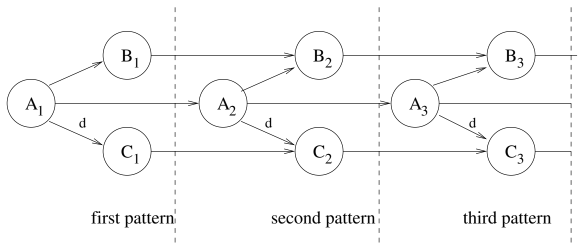

Figure 1: Algorithm graph

Each operation may belong to a precedence constraint, i.e. it belongs to a pair defining a partial order on the execution of operations. The operations which do not belong to a precedence constraint define a "potential parallelism" [13]. Moreover, because the algorithm graph describes a real-time system, which is reactive, this latter graph has a *pattern* infinitely repeated [12] which induces an infinite repetition of all operations.

The infinite repetition of an operation allows to define its strict periodicity constraint as follows:

Definition 1 *[14] For two consecutive repetitions A_i and A_{i+1} of the same operation A, we say that A has a periodicity constraint (is periodic with period) T_A if $s_{A_{i+1}} - s_{A_i} = T_A, \forall i \in \mathbb{N}$, where s_A is the start time of operation A. We denote by A_1 the first repetition of A.*

For the sake of simplicity, we use later on the term "periodicity constraint" instead of "strict periodicity constraint".

The first consequence of the latter definition is that a precedence constraint between two operations imposes a relation between the periods of these operations, i.e. for two periodic operations A, B with $(A, B) \in E_{al}$, as shown in [11], we have:

$$T_A \leq T_B \tag{1}$$

Moreover, if there is a data transfer between two operations then the theorem 1 states that their periods must be equal.

Theorem 1 *If a system of two periodic operations A and B with $(A, B) \in E_{al}$ having a data transfer between them is schedulable then we have $T_A = T_B$.*

Proof We prove the theorem by contradiction. We suppose that $T_A < T_B$ is satisfied. This latter equation is, because of equation 1, the contrary of the equation we want to prove. Starting from this supposition, we obtain a contradiction and the theorem is proved as follows.

We denote by s_A^k, respectively, by s_B^k the start time of operation A_k and of operation B_k. We have $s_B^k - s_A^k + C_B = s_B^1 + kT_B + s_A^1 + kT_A + C_B$. This latter equation, because $T_A < T_B$, implies that:

$$lim_{k \to +\infty} s_B^i - s_A^i + C_B = +\infty$$

and we should have an infinite memory to store the data produced by A. Therefore, we can not store the data produced by A, and B can not be scheduled. So, the system is not schedulable and we have our contradiction. The supposition made at the beginning of the proof is wrong and we have : $T_A = T_B$. The theorem is proved.

The architecture graph is a complete graph $G_{ar} = (V_{ar}, E_{ar})$ where V_{al} is the set of operators and $E_{ar} = V_{ar} \times V_{ar}$ the set of edges which represents the communication media connecting the operators. For example in figure 2, we have the communication media $m_i, \forall i \in \{1, 2, 3, 4, 5, 6\}$. This graph provides a "physical parallelism" which may be different from the potential parallelism, generally smaller than the latter.

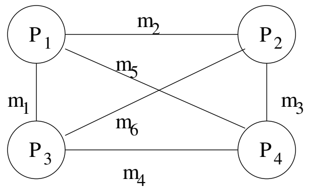

Figure 2: Architecture graph

Because the multiprocessor architecture is assumed heterogeneous, each operation A may have different computation times C_{Ap} (exactly known) for each operator p. Similarly each data transfer between operations scheduled on different operators may have different communication times for each communication medium.

Without any loss of generality, we assume that all characteristics, i.e. the computation times, the communication times, the periods, are defined as multiples of a clock tick τ (time is discrete). Afterwards the integer values of the given characteristics are implicitly multiplied by τ.

For a given algorithm graph and a given architecture graph, a solution to our multiprocessor scheduling problem is described by the start time of all operations and for each operation the

	A	B	C
p_1	2	2	9
p_2	2	4	2
p_3	1	2	1
p_4	2	4	5

Table 1: Computation times

operator where it is executed. For example, the figure 3 gives one of the possible solutions for the algorithm graph of figure 1 and for the architecture graph of figure 2, where all execution times are given by the table 1, the communication times are $m_1 = 2$, $m_2 = 1$, $m_3 = 1$, $m_4 = 4m_5 = 2$ and $m_6 = 1$ and the periods are $T_A = 4$, $T_B = 8$ and $T_C = 4$. The multiprocesor scheduling is obtained by using only operators p_1 and p_2.

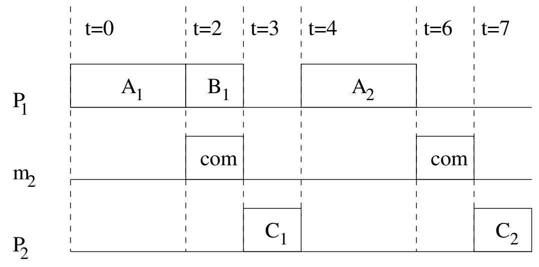

Figure 3: Multiprocessor scheduling

Definition 2 *For a system of operations, a schedule S is the set of pairs (s_A, p) where s_A is the start time and p the operator on which an operation A is scheduled such that the precedence and the periodicity constraints are satisfied. If there is a schedule for a system, the system is called schedulable; if not, then it is unschedulable.*

4 Complexity of our problem

In order to study the complexity of our multiprocessor scheduling problem, we define the decision problem associated, and we compare it to a decision problem already proved NP-complete. This way, we may conclude that our multiprocessor scheduling problem is NP-hard.

We denote by PP the decision problem associated to our multiprocessor scheduling problem defined as follows: is a system, given by an algorithm graph with periodic constraints and an architecture graph, schedulable?

We choose the decision problem PSP given by Korst [2] to prove the NP-completeness of problem PP. The decision problem PSP is defined as follows: is a system given by a set of independent operations with periodic constraints and an architecture, schedulable?

The lemma 1 proves the equivalence between PP and PSP and the theorem 2 proves the NP-completeness of our problem. The detailed version of these proofs are given in [15].

Lemma 1 *The answer is "yes" for PP if and only if the answer is "yes" for PSP.*

Proof We build a polynomial transformation between the two problems which allows us to obtain from a schedule for PP, a schedule for PSP.

Theorem 2 *PP is NP-complete.*

Proof The proof has two parts:

- we prove that PP belongs to NP;
- the lemma 1 proves that PP is polynomially reduced to PSP.

Once we proved that PP, which is the decision problem associated to our multiprocessor scheduling problem, is NP-complete, we can conclude that our multiprocessor scheduling problem is NP-hard. So, no polynomial algorithm may solve this problem unless P = NP and we have to propose heuristics.

5 Heuristic and exact algorithm

We present a heuristic and an exact algorithm applied on the same example. Then, we compare their performances.

Our heuristic is based on the fact that, due to strict periodicity, two operations with periods T_i and T_j which do not form a sequence such that $T_i/T_j, \forall i < j$ can not be scheduled on the same operator.

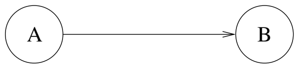

Figure 4: Algorithm graph

Example 1 *For the pattern of algorithm graph given in figure 4 with $T_A = 2$ and $T_B = 3$ and all computation times equal to 1, the system is not schedulable using one operator. The figure 5 shows that the third repetition of A and the second repetition of B must have the same start time equal to 4 and this is not possible.*

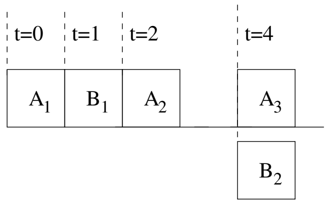

Figure 5: Monoprocessor scheduling

In order to schedule operations with periods coprime on different operators, we define classes of operations where each class $\mathcal{C}_T$ contains only operations with their periods not coprime, where T is the smallest period within operations belonging to the class. These classes are not mutually exclusive. For example, an operation with a period $T = 10$ could belong to the class $\mathcal{C}_2$ and to the class $\mathcal{C}_5$.

Once an operation is scheduled on an operator, only its class may use the operator. We denote by s_{lp} the start time and by C_{lp} the execution time of the last operation scheduled on operator p. We denote by $Prec(A)$ the predecessors set of an operations A and by $com(p, p')$ the communication time between operators p and p'. The heuristic 1 uses a working-set W containing the available operations which is updated each time an operation is scheduled. It proceeds as follows:

Heuristic 1

Step 1: we choose for each class within W the operation with the smallest period and the smallest computation time on all operators. We schedule these operations.

Step 2: for each operator p, we search for the operation $A \in W$ with A_1 already scheduled such that the following expression has the smallest value: $s_{B_i} + T_B - s_{lp} - C_{lp}$. If $W = \emptyset$ then we stop.

Step 3: for each A and p chosen at Step 2, if there is $B \in W$ with B_1 not scheduled yet and B and A belonging to the same class such that:

$$s_{lp} + C_{lp} + C_B \leq max_{C \in Prec(B)}\{s_{Ap'} + C_{Ap'} + com(p', p)\} \leq s_{A_i} + T_A \qquad (2)$$

then we schedule B and go to Step 5. If not, go to Step 4.

Step 4: we schedule A chosen at Step 2.

Step 5: if there are operations of W belonging to a class without scheduled operations, we chose the operation with the smallest period.

Step 6: if there are operations (with the first repetition not scheduled) belonging to a class which can not be scheduled anymore on its operator, then we choose a new operator for this class. If we can not choose a new operator which is not already used by another class, then the system is not schedulable with this heuristic. If not, go to Step 2.

Remark 1 *The equation 2 takes into account the communication time between two operators when the predecessors of an operation are scheduled on different operators as this operation. This communication time is computed according to the cost function given in [12]*

Example 2 *For the pattern of the algorithm graph given by figure 6 with $T_A = T_B = T_D = T_E = 4$, $T_G = T_H = 8$ and $T_C = T_F = T_I = 3$ and the architecture graph given by figure 7. We have two classes of operations $\mathcal{C}_4$ and $\mathcal{C}_3$. All computation and communication times are equal to 1. The figure 8 gives the multiprocessor scheduling obtained using the heuristic. Because there is no communication using the media, these latter are not shown.*

The exact algorithm is an algorithm of type branch & bound which searches for a solution within the set of all possible solutions. In order to bound its complexity and without any loss of generality, we suppose that the operations have the same execution time for all operators, and

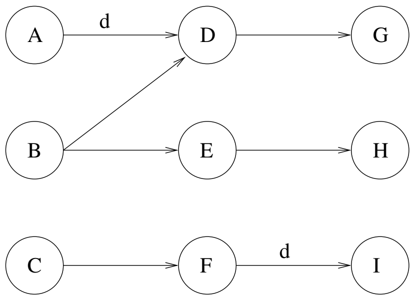

Figure 6: Algorithm graph

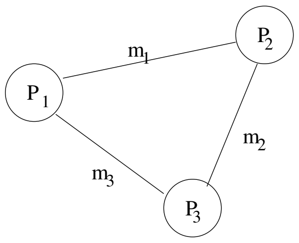

Figure 7: Architecture graph

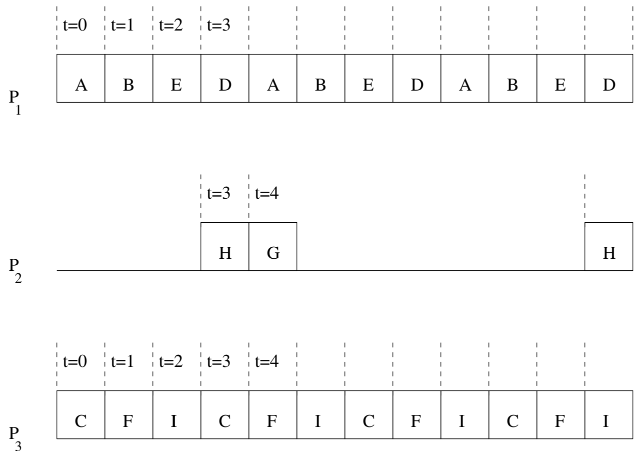

Figure 8: Multiprocessor scheduling obtained using heuristic

the data transfers have the same communication times for all media (homogeneous architecture). Consequently, we may consider a list of m operators in any order.

We consider an ordered list of n operations such that the predecessors of an operation of index i have the index $k < i$. Also all operations, which become available at the same time as an operation of index i and have a bigger period, have the index $k > i$. The following example shows such an order:

Example 3 *For the pattern of the algorithm graph given by figure 9 with* $T_A = T_B = T_D = T_E = 4$, $T_G = T_H = 8$ *and* $T_C = T_F = T_I = 3$ *we have the list* $\{C, F, I, A, B, D, E, G, H\}$.

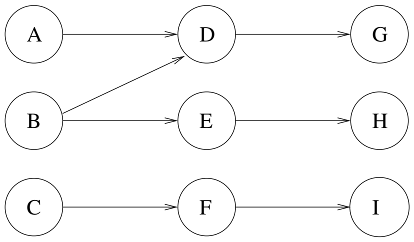

Figure 9: Algorithm graph

The exact algorithm uses the following notations: i for the operation of index i in the list of operations, j for the operator of index j in the list of operators, $s(j)$ the start time of the last operation scheduled on j and $C(j)$ the computation time of the last operation scheduled on j. It proceeds as follows:

Step 1: we initialize $i = 1$, $j = 1$, $s(k) = 0$ and $C(k) = 0, \forall k$.

Step 2: if $i \leq n$ then we schedule all repetitions of operation i on operator j, $s(j) = s(j) + C(j)$ and $C(j) = C_i$, else we stop. If there is at least one constraint which is not satisfied then $j = j + 1$ and go to Step 3, else go to Step 4.

Step 3: if $j > m$ then the system is not schedulable, else go to Step 2.

Step 4: i = i + 1 and go to Step 2.

Example 4 *For the example given in section 3, the figure 3 gives the multiprocessor scheduling obtained by using the exact algorithm.*

We end this section by presenting the performances of the heuristic compared to those of the exact algorithm. The set of operations, their periods and computation times and the precedence constraints are randomly generated and the time needed by this generation is not taken into account in the final time. The precedence constraints are generated such that the obtained graphs do not contain cycles. The obtained random problems are similar is size, precedence constraints, computation times and periods to the real problems.

Also we choose the number of operators equal to the number of operations in order to ensure that every problem randomly generated has a solution. Moreover, this allows to compare the number of operators used by the heuristic and the exact algorithm. The results given in table 2 show that the heuristic distributes better the operations on the operators than the exact algorithm, because the heuristic uses a larger number of operators than the algorithm.

Number Operations	Exact algorithm	Heuristic
100	62	64
1000	628	703
2000	1256	1405
3000	1868	2112
4000	2455	2786
5000	3071	3490
6000	3702	4214
7000	4272	4894
8000	4857	5549
9000	5528	6291
10000	6076	6958

Table 2: Comparison between the number of operators

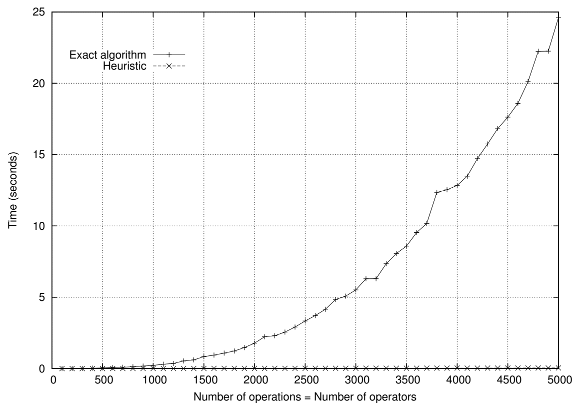

Figure 10: Comparison between heuristic and algorithm

The figure 10 gives the comparison between the execution times of heuristic and exact algorithm. For each number of operations we generate one random problem. For a small number of operations, the heuristic and the algorithm take almost the same time, but once we increase the number of operations, the heuristic is definitely much more interesting. In order to underline the differences, we present the same comparison but with a logarithmic scale (figure 11).

6 Conclusion and further research

In this paper, we present the problem of multiprocessor scheduling for strict periodic systems with precedences constraints. We prove that this problem is NP-hard and we propose a heuristic. The performances of this latter is compared to those of an exact algorithm of type branch & bound solving the same problem and show that the heuristic is definitely better.

We want to extend these results to systems with precedence and periodicity constraints allowing jitter modeled by a latency constraint [14]. Moreover we plan to study the complexity of the

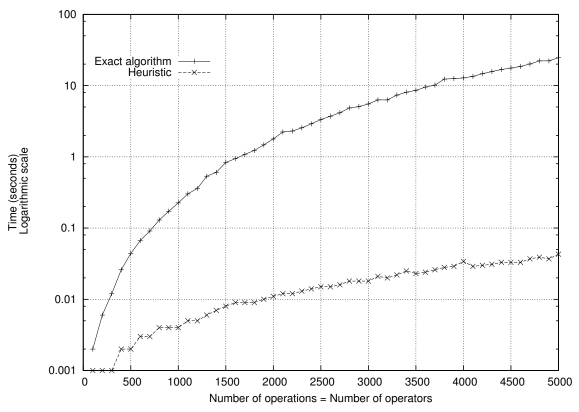

Figure 11: Comparison with logarithmic scale

preemptive problem and to propose a heuristic or an algorithm (depending on its complexity).

References

[1] D Harel and A Pnueli. On the development of reactive systems. In K. R. Apt, editor, *Logics and Models of Concurrent Systems*. Springer Verlag, New York, 1985.

[2] J.H.M. Korst, E.H.L. Aarts, and J.K. Lenstra. Scheduling periodic tasks. *INFORMS Journal on Computing 8*, 1996.

[3] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. *IEEE*, 1991.

[4] J. Leung and Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation(4)*, 1982.

[5] D.-I. Oh and T.P. Baker. Utilization bounds for n-processor rate monotone scheduling with static processor assignment.

[6] J. Goossens, S. Baruah, and S. Funk. Real-time scheduling on multiprocessors. *Real-Time Systems, Paris*, 2002.

[7] K. Tindell and J. Clark. Holistic schedulability for distributed hard real-time systems. *Microprocessing and Microprogramming*, 1994.

[8] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *TR*, 2002.

[9] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[10] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[11] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. *Real-time and Embedded Systems*, 2003.

[12] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real time embedded heterogeneous multiprocessors. *Codes'99 7th International Workshop on Hardware/Software Co-Design*, 1999.

[13] Y Sorel. Massively parallel computing systems with real time constraints, the "algorithm architecture adequation" methodology. *Proceedings of Massively Parallel Computing Systems, Italy*, 1994.

[14] L. Cucu, R. Kocik, and Y. Sorel. Real-time scheduling for systems with precedence, periodicity and latency constraints. *Real-time and Embedded Systems*, 2002.

[15] L. Cucu. *Ordonnancement non préemptif et condition d'ordonnançabilité pour systèmes embarquś à contraintes temps réel.* PhD thesis, University Paris XI, 2004.