# Non-preemptive multiprocessor static scheduling for systems with precedence and strict periodicity constraints

*Omar Kermia*[1], *Liliana Cucu*[2], *Yves Sorel*[1]

(1)INRIA Rocquencourt,
BP 105 - 78153 Le Chesnay Cedex, France
Phone: +33 1 39 63 52 60 - Fax: +33 1 39 63 51 93
*omar.kermia@inria.fr, yves.sorel@inria.fr*
(2)IPP Hurray Research Group,
Polytechnic Institute of Porto, Portugal
Phone: +351 22 8340502 - Fax:+351 228340529
*lcucu@dei.isep.ipp.pt*

**Abstract**

In this paper we propose a greedy heuristic to solve the non preemptive multiprocessor static scheduling problem with precedence and strict periodicity constraints. The system of periodic tasks is described by a graph where dependent tasks are connected by precedence constraints. First, each task is repeated within the LCM of all periods of tasks (hyper-period) allowing to unroll the graph over the hyper-period. We propose an algorithm adding some missing edges. Then, the tasks of the new graph are classified by their periods and assigned to the processors. Finally, we propose an extension of the SynDEx heuristic which already allocates and schedules tasks graph with precedence constraints, but tasks have the same period equal to the execution time of the complete graph. We apply this extended heuristic on the unrolled graph in order to allocate and schedule the periodic tasks onto the multiprocessor. We compute the complexity of the proposed heuristic, and perform a performance comparison which shows its effectiveness.

## 1 Introduction

Distributed real-time embedded applications found in domains such as avionic, automobile, autonomous robotics, telecommunication lead to non preemptive multiprocessor static scheduling problem with precedence and strict periodicity constraints. These applications are of crucial importance because if all the constraints are not satisfied this may have disastrous consequences.

The multiprocessor periodic scheduling problem was discussed by Liu in 1969 [1]. Given a set of $n$ tasks and $m$ resources, where each task $x$ has rational weight $x.w = x.e/x.p$ ($x.e$ and $x.p$ reffered to execution requirement and period), $0 < x.w < 1$, a periodic schedule according him is one that allocates a resource to a task $x$ for exactly $x.e$ time unit or slots in each interval $[x.p.k,x.p.(k+1)]$ for all $k \in N$. Scheduling decisions may be made only at integral times and a task may use either zero or one resources at a time. To solve this problem Baruah [2] proposed the fair scheduling that consider a relaxed version of the Liu's problem in which tasks are not restricted to using zero or one resource at a time (resource sharing). But this approach can not be applied to our problem because, contrary to Liu's problem, we have a machines instead of resources and in this case resource sharing would mean that one machine can execute more than one task on the same time. Xu [3] consider task sets where individual tasks are further divided into subtasks with precedence relation in order to give more parallelism within tasks, and in this idea Shirazi [4] developed an algorithm for static multiprocessor scheduling of periodic tasks which are independent and their subtasks are related. [5] gives a lot of known results to solve the periodic schedule but he considers that all the tasks are at the same period. In this paper we deal with a non preemptive multiprocessor scheduling with precedence and multiple strict periodicities constraint and we propose an algorithm which take into account the inter-processor communications.

This problem is solved here by a partitioning method and the partitioning problem is analogous to the bin-packing problem known NP-hard [6]. Thus, there are no optimal algorithms which solve it, whereas heuristics techniques are very fast despite their sub-optimal results. Because embedded distributed real-time systems involve rapid prototyping for useful solutions, we choose greedy heuristics which are very fast and produce good results.

## 2 Model and heuristic principles

[7] gives a model based on graph theory for systems of tasks with precedence constraints and gives also the definition of the strict periodicity constraint. Because it is possible but not necessary to use a resident real-time operating system for the execution of these tasks, afterward we shall use the term "operation", more generic, instead of "task". Figure 1 depicts the infinitely repeated pattern of the dependence graph where vertices are operations and precedence edges between operations due to data transfer. Let assume that these periodic operations have the periods: $T(A)$=2, $T(B)$=$T(C)$=3, $T(D)$=6. The execution time of all operations and of the inter-processor communications is equal to $C$=1. The multiprocessor architecture is also specified by a graph, in this example we assume it is composed of two identical processors $P1$ and $P2$ connected by a communication medium $M$. An allocation and scheduling of this system satisfying all the constraints (precedence and periodicity) is showed on figure 2. We notice that the operations of the graph are executed according to a scheduling pattern whose length is always the same. Into this time interval the operations are repeated, and the number of repetitions of each operation depends on its period. The length of this time interval is the least common multiple (LCM) of all operations periods that corresponds to a hyper-period [8]. Thus, the graph pattern given in figure 1 does not depict all repetitions of the operations within the hyper-period. Consequently, we consider a graph pattern where operations are repeated, that we call *unrolled graph*.

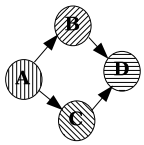


Figure 1: Dependence graph

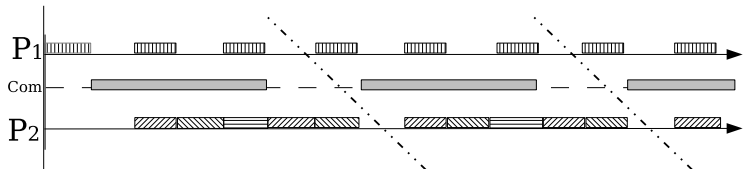


Figure 2: Allocation and scheduling

## 3 Heuristic

The proposed heuristic is composed of three steps.

### 3.1 Graph unrolling

We consider a scheduling frame whose length is the LCM of all periods of operations. We repeat each operation $n$ times, $n$ is equal to $n = (P/T)$ where the period of this operation is $T$ and $P$ is the hyper-period [9]. In order to maintain the data transfers between operations during the unrolling we must add new edges between the repetitions of the same operations, and between the repetitions of different operations.

Adding the edges is simple to do when two operations have a precedence constraint and the same period. But if they have different periods, there are several possibilities to add the edges. We choose an edges addition algorithm which computes the ratios between the hyper-period and the period of each of both operations, add as many edges as the smallest ratio, and fairly distribute them between the repetitions of these two operations. Consequently, some of the repeated operations become connected.

### 3.2 Classification and assignment

The operations are classified (clustered) by their periods in such way that each class contains the operations with the same period. These classes are assigned to the processors as follows. If the number of processors is greater than the number of classes we extend the number of classes by creating new ones to obtain as many classes as number of processors. We propose a load-balancing algorithm which divides recursively the class with the greater load into two classes that have the same load until the total number of classes is equal to the number of processors. If the number of processors is lower or equal to the number of classes it is not necessary to apply the previous load-balancing algorithm, and then we have more classes than the number of processors. Now, we assign the resulting classes to the processors according to their ascendant period order, possibly leading to non-assigned classes which will be assigned during the next step.

### 3.3 Extension of the SynDEx heuristic

The SynDEx heuristic [10] uses a cost function which tends to minimize the makespan (length of the critical path of the dependence graph) taking into account the worst case execution times of the operations and of the inter-processor communications, and the schedule flexibility of each operation. It allocates and

schedules operations graph with precedence constraints but in the case where operations have the same period equal to the execution time of the entire graph.

At each main step of the heuristic, one operation is selected from the set of schedulable operations (an operation becomes schedulable when all its predecessors in the graph are already scheduled). In order to select an operation, we first seek, for each schedulable operation, its best processor, i.e. which minimizes the cost function. Then, from these operations, we select the most critical one, i.e. which maximizes the cost function, and schedule it onto its best processor where it is finally allocated to. This operation is removed from the set of schedulable operations and all its successors that become schedulable are added to this set. The next main step is ready to begin with this new set, and the heuristic stops when it is empty.

Here, we extend the SynDEx heuristic in order to satisfy, in addition to precedences, several strict periodicities constraints. The extended heuristic is applied onto the unrolled graph instead of the initial one. At each main step, like in the original heuristic, one operation is selected from the set of schedulable operations. This set contains two kinds of operations: those belonging to an assigned class and those belonging to a non-assigned class. If the operation belongs to an assigned class, it is considered as allocated to this processor, and the cost function is computed relatively to this processor. If the operation belongs to a non-assigned class, the cost function is computed relatively to all the processors in order to find its best processor. Once every operation (assigned and non-assigned) has found its best processor, we select the most critical one, which becomes scheduled and allocated. Then, the heuristic proceeds like the original one. Moreover, in order to guarantee that the execution of the repeated operations satisfy the periodicity constraints, we verify at each main step that a schedulable operation fulfills two conditions given bellow.

Let $B$ a schedulable operation. $\Theta$ is the set of the periodic operations already scheduled whose successors, corresponding to the same repetition, was not yet scheduled. We denote by $O_1$ the last operation scheduled on the processor $P_1$ and $S(X)$ the start time of operation $X$.

▶ First condition: if the scheduling of the operation $B$ on the processor $P_1$ does not need data transfer, it is necessary that: $\forall A \in \Theta$ and $T(A)$ its period, $S(O_1) + C(O_1) + C(B) \leq S(A) + T(A)$.

▶ Second condition: if the scheduling of the operation $B$ on the processor $P_1$ needs data transfer (from the operation $O_2$ scheduled on the processor $P_2$ with $Com$ the execution time of this inter-processor communication), it is necessary that: $\forall A \in \Theta$ and $T(A)$ is its period, $S(O_1) + C(O_1) + C(B) \leq S(A) + T(A)$ and $S(O_2) + C(O_2) + Com + C(B) \leq S(A) + T(A)$.

### 3.4 Example

In order to illustrate how the proposed heuristic progresses we apply it to the graph of figure 1 with the same multiprocessor (two processors conected by a communication medium). The periods of the operations are: $T(A)$=4, $T(B)$=$T(C)$=5 and $T(D)$=20. The execution times are: $C(A)$=$C(B)$=$C(C)$=2 and $C(D)$=5 for the operations and $Com$=2 for the inter-processor communication.

• Graph unrolling: the LCM of the periods is $P$=20, $n_1$=20/4=5, $n_2$=20/5=4, $n_3$=20/5=4, $n_4$=20/20=1 are respectively the repetition numbers of the graph operations $A$, $B$, $C$ and $D$. There are four edges between the repetions of $A$ and those of $B$, the same number between the repetions of $A$ and those of $C$, one edge between $B$ and $D$, and one edge between $C$ and $D$. The resulting graph is depicted on figure 3.

• Classification and assignment: the operations are classified according to their periods in three classes: $c_1$=$\{A\}$, $c_2$=$\{B, C\}$, $c_3$=$\{D\}$. Because the multiprocessor architecture is composed of only two processors, we assign the first class to one processor and the second class to the other one, and the third class is not assigned.

• Execution of the extended SynDEx heuristic: by applying this heuristic on the unrolled graph (figure 3) we obtain the allocation and scheduling given on figure 4.

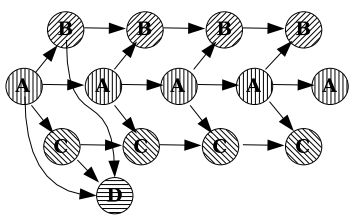


Figure 3: The unrolled dependence graph

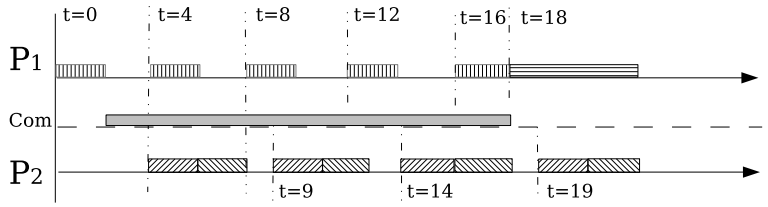


Figure 4: Allocation and scheduling

## 4 Heuristic complexity and performance

In order to compute the complexity of the proposed heuristic, we start by reminding the complexity of the SynDEx heuristic which is equal to $N^2 \times M/2$ with M the processors number, N the number of operations in the dependence graph, and compute the number of unrolled graph operations which is equal to $N_{inrolling} = \sum_{i=1}^{N} P/T(O_i)$ with $O_i$ an operation of the graph and $T(O_i)$ its period and $P$ the hyper-period. Consequently, the complexity of the proposed heuristic is $N_{unrolling}^2 \times M/2$.

We compared the execution times of our heuristic to the one given in [8] on a similar problem in terms of number of operations and processors. The results are summarized in the table given below.

| | Number of operations | Number of processors | Execution time (sec) |
|---|---|---|---|
| Simulated annealing heuristic | 624 | 9 | 5572 |
| Proposed heuristic | 681 | 9 | 32.61 |

## 5 Conclusion and further research

The paper proposes an heuristic to allocate and schedule systems of operations with precedence and periodicity constraints onto multiprocessor, and gives some complexity and performance results showing that it can be used in realistic embedded real-time applications. We plan to investigate other possibilities for adding edges in the unrolled graph, and propose a flexible method for adding these edges which deals with a large kind of applications and adapts to the users needs.

## References


[1] C. L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. In *JPL Space Programs Summary 37-60*, volume 2, pages 28–37, November 1969.

[2] S. K. Baruah. Fairness in periodic real-time scheduling. In *RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS '95)*, page 200, Washington, DC, USA, 1995. IEEE Computer Society.

[3] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transctions,on Parallel Distibuted Systems*, 19(2), February 1993.

[4] S. Rnngren and A. Shirazi. Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs. In *Proceeding of the 28th Annual Hawaii International conference on system Sciences*, Departement of Computer Science and Engineering, University of Texas, 1995.

[5] J. Leung. Handbook of scheduling. pages 7–4.7–5. Published by CRC Press, Boca Raton, FL, USA, May 2004.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability. A guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[7] L. Cucu and Y. Sorel. Schedulability condition for systems with precedence and periodicity constraints without preemption. In *Proceedings of 11th Real-Time Systems Conference, RTS'03*, Paris, March 2003.

[8] S-T. Cheng and A. Agrawala. Allocation and scheduling or real-time periodic tasks with relative timing constraints. In *Technical Report CS-TR-3402*, Computer science, University of Maryland, College Park, January 1995.

[9] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 6, University of Massachusetts, April 1995.

[10] T. Grandpierre, C. Lavarenne, and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *Proceedings of 7th International Workshop on Hardware/Software Co-Design*, May 1999.