Static Mapping of Hard Real-Time Applications Onto Multi-Processor Architectures Using Constraint Logic Programming*

Christophe Guettier
SAGEM, 178 rue de Paris
F-91300 Massy, France
Christophe.Guettier@sagem.com

Jean-François Hermant
INRIA, Rocquencourt, B.P. 105
F-78150 Le Chesnay, France
Jean-Francois.Hermant@inria.fr

Abstract

The increasing complexity of embedded real-time architectures, as well as the design of critical systems, has become both difficult to manage and costly. Mapping statically the set of tasks onto a multi-processor system is one of the most crucial issues. The designer must guarantee system feasibility, based on the system model and consideration of design decisions with respect to the requirement set. Our approach combines well-known online preemptive scheduling algorithms and their associated feasibility conditions, with problem solving techniques that are based on Constraint Logic Programming (CLP). We address a large class of online preemptive scheduling algorithms including so called fixed priority policies (Highest Priority First - HPF), as well as dynamic priority policies (specifically, Earliest Deadline First - EDF). The paper presents how to solve the mapping problem on representative examples, considering globally both task placement and hard real-time schedulability constraints. Optimization techniques are also experimented in order to minimize system dimensions. Lastly, we outline different recommendations for the design of efficient search strategies. Several benchmarks from various domains are considered as running examples throughout the paper.

Introduction

Deficiencies in the design or dimensioning of a critical and real-time high-performance system can cause fatal failures. In order to prove that for an entire system, real-time and architectural size requirements are met, it is necessary to prove its dimensioning. A relevant example of correct system dimensioning is ensuring that the processing resources are sufficient. This is a huge task as the increasing complexity of systems involves designs of combinatoric complexity. To master the complexity of system specification and design, we consider a proof-based methodology, like TRDF[1] as discussed in (Le Lann 97)(Le Lann 98). TRDF allows translation of the (incomplete and/or ambiguous) description of an application problem into a precise specification. To be acceptable, computer-based systems must come with proofs that all the decisions made during the system design satisfy system specification.

The complexity of target architectures has increased too, with many functional units incorporated on a single chip, the placement of tasks onto the set of processors remains a complex problem raising combinatorial explosions. It is necessary to consider simultaneously several interdependent subproblems such as task scheduling and placement. Usually highly combinatoric, each of these problems is characterized by feasibility constraints and design decisions (scheduling policy, data allocation strategy, communication protocols). Therefore, applying a proof-based method requires multiple formulations for the representation of the global system. The modelling phase of the method captures the invariants of the system specification, but also decomposes and simplifies its design complexity from coarse to fine grain. The designer has to solve a task placement problem, and must guarantee real-time schedulability. Evenmore, the designer needs to optimize the architecture to fit some space requirements or to optimize system performances.

Our approach relies on Constraint Logic Programming on finite parts of $\mathbb{N}$ CLP(FD) with all its classical operators (Van Hentenryck et al. 95). We claim that this language can solve and possibly optimize the design of complex hard real-time systems. One area of experimentation for Constraint Programming techniques has been the automatic data-layout of High-Performance FORTRAN programs onto parallel computers, using $0-1$ modelling, CPLEX and branch-and-bound searches (Bixby and al. 94). Using CLP, relevant results have also demonstrated the efficiency of the approach for modelling and solving Digital Signal Processing placement problems (Ancourt and al. 97),(Thiele 97). Another area of investigation has been the engineering of real-time (multi-processor) applications (Bacik and al. 98),(Guettier and Hermant 99). For all these examples, it has been necessary to solve the global problem globally using a composite model.

This paper presents a global approach for solving or optimizing automatically the mapping of real-time tasks onto multi-processor architectures using CLP. Focusing on the feasibility conditions of real-time scheduling algorithms, generic models and search methods are proposed to tackle the mapping feasibility, as well as the system size opti-

[1]TRDF is the French acronym of Real-Time Distributed Fault-Tolerant Computing.

mization. The different modelling steps required to express tractable feasibility conditions in CLP are detailed for an exhaustive class of online scheduling policies. It encompasses necessary and sufficient feasibility conditions for online scheduling algorithms such as Earliest Deadline First (EDF), Highest Priority First (HPF), as well as its specializations, Deadline Monotonic (HPF/DM), and Rate Monotonic (HPF/RM). In spite of their complexity, we demonstrate that analytical feasibility conditions can be expressed as equivalent feasibility constraints with CLP. Different composite search strategies are also proposed, combining load balancing and priority assignement (Audsley 1991) techniques. This association of offline solving techniques with online scheduling solutions is a new approach to the proof-based design of complex distributed and real-time systems.

Problem Specification

In this section, we specify the problem under consideration according to the TRDF method (Le Lann 97),(Le Lann 98). First, system requirement capture is summarised. Requirements are modeled in a fairly standard fashion, taking into account the problem models. Second, we state the major problem property, i.e. the timeliness property. As a result, it is possible to meet hard real-time constraints that would not be satisfied with a single processing element.

Models of System Requirements

In subsequent sections of the paper, demonstrations are performed using the assumptions that follow. System requirements define a global design problem that must be broken down into several sub-problems of lower complexity. Each sub-problem can be modeled with its own mathematical invariants. Finding a set of feasible solutions to a given sub-problem requires the instantiation of all the variables of the sub-problem. Relations resulting from this decomposition correlate model variables. They maintain the consistency of different local solutions and thus feasibility of the global solution.

Computational Model We assume the synchronous computational model (task durations have known bounds). Without loss of generality for the models described in this paper, we will use a discrete time model: $t \in \mathbb{N}$.

Task Model Let τ be the task set to execute. We consider a level of specification where each task $i \in \tau$ is represented by a sequence of steps, each step being mapped onto exactly one processor. A task has a worst case computation time $\forall i,\ C_i \in \mathbb{N}$ which can be constant or variable offline. When it is variable, upper and lower bounds exist and their values are known.

External Event Types Models All tasks are ready to execute whenever requested. Activation demands of a task are constrained by a periodic or sporadic arrival model (Mok 83).

Periodic	**Sporadic**
$\forall i, \exists \phi_{i,0} \in \mathbb{N},\ \forall a_i \in \mathbb{N},$	$\forall i, \forall j \in \mathbb{N},\ \exists \phi_{i,j} \in \mathbb{N},\ \forall a_i \in \mathbb{N}$
$d(a_i) = \phi_{i,0} + a_i T_i,$	$d(a_i) = \sum_{j=0}^{a_i} \phi_{i,j} + a_i T_i$
The $(a_i+1)^{th}$ activation date of task i is denoted $d(a_i)$. The first activation date of task i corresponds to $a_i = 0$.	
The period/minimal inter-arrival time of task i is denoted $T_i \in \mathbb{N}^*$	
The concrete or non-concrete attributes of task i:	
Concrete: $\phi_{i,0}$ known.	$\{\phi_{i,j}\}_{j\in\mathbb{N}}$ known.
Non-concrete: $\phi_{i,0}$ unknown.	$\{\phi_{i,j}\}_{j\in\mathbb{N}}$ unknown.

where $\phi_{i,0}$ is a phase difference/$\{\phi_{i,j}\}_{j\in\mathbb{N}}$ are phase differences. The sporadic model is more general than the periodic model (Mok 83) and our approach holds for both arrival models.

In subsequent sections of the paper, we consider a **periodic/sporadic non-concrete traffic** τ, which is a finite set of n periodic/sporadic non-concrete traffics τ_i. A periodic/sporadic non-concrete traffic τ_i captures an infinite set of periodic/sporadic concrete traffics ω_i. A periodic/sporadic concrete traffic ω_i is caracterized by its known activation dates $\{d(a_i)\}_{a_i\in\mathbb{N}}$, its worst case computation time C_i, its period/minimal inter-arrival time T_i, and its relative deadline D_i. $\forall a_i \in \mathbb{N},\ d(a_{i+1}) - d(a_i) \geq T_i$ and $0 \leq C_i \leq Min\{T_i, D_i\}$. The term traffic is commonly used in the telecommunication community to refer to a task set. These two terms can be used indifferently.

Task Placement Model For each task i, we use a coarsed grain placement form in order to size the task workload according to the whole target architecture. At a first glance, this can be formulated using classical set partitioning formulations.

Architectural Model The architectural model is MIMD "Multiple Instruction Flow, Multiple Data Flow", where the different processors can execute multiple instructions in parallel. The architecture is homogeneous, worst case computation time C_i does not depend on which processor is i is placed.

Property

To be feasible, the solution must satisfy the timeliness property, under the models described above.

Tasks are assigned timeliness constraints: latest termination deadline. A solution is said to be feasible if for every possible system run, every timeliness constraint is met. Values of deadlines are dictated by application considerations. A deadline can be expressed as a natural number: $\forall i \in \tau,\ D_i \in \mathbb{N}^*$

Example of real world specifications

This section presents three realistic specifications extracted from real world applications discussed in (Tilman et al. 01),(Guettier and al. 01), that are used as on-going examples throughout this paper. Althought various cost functions could be investigated, finding a feasible solution on a minimal set of processors is considered to be the engineering cost objective.

Task	$C(ms)$	$T(ms)$	$D(ms)$
insert_target	50	250	100
distance_eval	100	500	150
pursuit_target	150	500	300
suppress_target	20	200	500

Figure 1: Example of a `detection system`.

Task	$C(ms)$	$T(ms)$	$D(ms)$
FDIR	20	100	100
energy_manager	100	500	400
camera_controler	40	100	100
mmemory_controler	400	600	1000
telecom_protocol	100	400	200
antena_controler	40	100	200
unload_protocol	200	400	200

Figure 2: Example of an `observation spacecraft` *system.*

Detection system: As a simpler example, we propose a detection system (typical of airborne radars or sonars) that manages a list of targets. A Digital Signal Processing system (out of the scope of the example) processes a beamforming algorithm (Ancourt and al. 97).

Spacecraft system: This system performs some observations and saves its data on a mass memory (Tilman et al. 01). Controled remotely from earth, the spacecraft can nevertheless perform Fault Detection Isolation and Recovery. Data can be unloaded or new pictures ordered using a communication antenae.

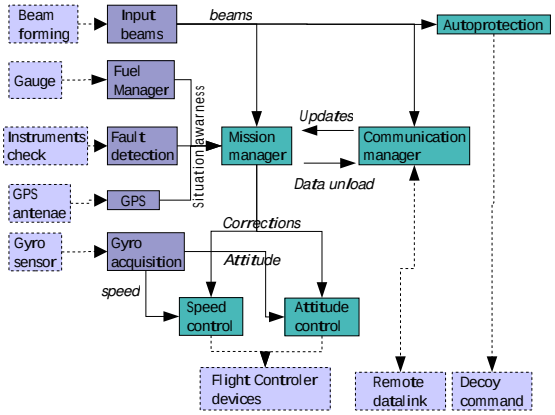
UAV Avionic system: It is a typical (although very simplified) core functional architecture (fig. 3) of a Unmanned Aerial Vehicle (UAV) system (Guettier and al. 01). It performs some observations that are downloaded, following a predefined mission plan. According to situation awarness, the mission manager task sends limited corrections to the speed and altitude controller tasks. It can also send situation information to a remote operator.

Hard real-time scheduling & mapping models

This section presents the constraint based model of the hard real-time scheduling and mapping problem. Feasibility constraints of scheduling algorithms are extracted from the state of the art in hard real time computing and modeled using a CLP(FD) language.

The mapping model, based on a set partitioning formulation is at first defined. A more generic formulation than well-known uniprocessor scheduling feasibility conditions is then detailed. These necessary and sufficient constraints are involving variables $C_i,\ T_i,\ D_i$ as well as task partitioning. The modelling includes Highest Priority First (HPF) like Deadline Monotonic (DM), Rate Monotonic (RM) as well as Dynamic Priority classes of scheduling such as Earliest Deadline First (EDF).

In order to improve the problem solving efficiency, several decisions in problem representation have to be made. In the sequel, we explain how the modelling is refined in order to extract sufficient conditions, used as heuristic constraints.



Task	$C(ms)$	$T(ms)$	$D(ms)$
attitude_control	3	8	8
fuel_manager	2	30	20
mission_manager	4	20	11
gps_update	5	40	11
autoprotection	6	40	16
fault_detection	5	20	11
speed_controler	2	10	11
gyro_acquisition	6	10	15
beam_input	4	10	15
com_manager	2	15	14

Figure 3: Example of an `UAV avionic` *system.*

Furthermore, throughout the modelling, constraints are refined in order to be tractable with CLP capabilities.

Mapping models

The mapping of real-time applications can be decomposed into task placement, architectural and relational models. Set partitioning is expressed using $0-1$ formulation and represents task distribution. The architectural model is then expressed as a resource constraint. Additional relational statements ensure the global solution consistency.

Task placement as set partitioning We use a $0-1$ formulation to specify whether the processor p is allocated to a given task i. This model has been widely investigated for the representation of various combinatorial problems involving set partitioning (Gondran and Minoux 95).

$$\forall i \in \tau,\ \forall p \in [0, \mathbf{P}_{max}),\ m_p^i \in \{0,1\}$$
$$\forall i \in \tau,\ \sum_{p=0}^{\mathbf{P}_{max}-1} m_p^i = 1 \qquad (1)$$

Statement (1) specifies that a task i is allocated to a single processor p iff $m_p^i = 1$. Using this formulation, the actual partitioning may lead to a lower number of busy processors.

Architectural resource constraints The number of busy processors $\mathbb{P}$ has an upper-bound determined by the constant $\mathbf{P}_{max}$. It defines the number of processors required to map the entire task and is given by $\mathbb{P} = card\{p \in [0, \mathbf{P}_{max}) | \exists i \in \tau / m_p^i = 1\}$.

$$\mathbb{P} = \sum_{p=0}^{\mathbf{P}_{max}-1} \underset{i\in\tau}{max}\{m_p^i\} \quad (2)$$

Relational Constraints Relational statements between models are required to retrieve global consistent solutions and to globally optimize cost functions such as the system size.

$$\forall i \in \tau,\ \forall p \in [0, \mathbf{P}_{max}),\ C_i(p) = C_i.m_p^i \quad (3)$$

Modelling feasibility constraints of online scheduling algorithms

The scheduling algorithms under consideration belong to the class of online real-time scheduling algorithms. In this class, there are two subclasses: that of deadline-driven scheduling algorithms and that of fixed-priority scheduling algorithms. EDF belongs to the former subclass and dominates any other scheduling algorithm belonging to the latter subclass, such as, for instance, Highest Priority First/Deadline Monotonic (HPF/DM) and Highest Priority First/Rate Monotonic (HPF/RM) (Dertouzos 1974).

This section gives feasibility conditions for EDF and HPF policies, which allows us to conduct a comparative study in terms of constraints complexity.

Basic concepts This section defines the basic concepts introduced in (Liu and Layland 73). Further sections extend the approach and recast feasibility conditions into tractable constraints for CLP languages.

The workload $W(p,t,\tau)$: By definition, the workload $W(p,t,\tau)$ for each processor p is the amount of time that is needed to run all the tasks whose activation times are in $[0,t)$ (Baruah et al. 1990b). To give the expression of $W(p,t,\tau)$, we consider the synchronous concrete traffic $\omega \in \tau$, where $\forall i, \forall j \in \mathbb{N},\ \phi_{i,j} = 0$.

$$W(p,t,\tau) = \sum_{j\in\tau} W(p,t,j) = \sum_{j\in\tau} \left\lceil \frac{t}{T_j} \right\rceil C_j(p). \quad (4)$$

A necessary feasibility condition (NC): This well-known necessary condition can be used as a heuristic to solve the global problem.

$$\tau \text{ is feasible by EDF, HPF} \Rightarrow \forall p \in [0, \mathbf{P}_{max}), \sum_{j=1}^{n} \frac{C_j(p)}{T_j} \leq 1. \quad (5)$$

sketch of the proof:

We derive the utilization factor $U(p,\tau)$ from the workload $W(p,t;\tau)$:

$$U(p,\tau) = Lim_{t\to\infty} \left\{ \frac{W(p,t;\mathcal{P}_p(\tau))}{t} \right\} = \sum_{j=1}^{n} \frac{C_j(p)}{T_j}.$$

By definition, the utilization factor $U(p,\tau)$ for processor p is the fraction of time that is needed to run all the tasks over $[0,\infty)$, i.e., the limit of $W(p,t;\tau)/t$ as t tends to infinity. If $\mathcal{P}_p(\tau)$ is feasible by EDF, then $U(p,\tau) \leq 100\%$.

Using exact feasibility conditions for EDF

The EDF policy works as follows (Liu and Layland 73). At any time $t \in \mathbb{R}^+$, if there are pending tasks (i.e., tasks which have been previously activated but which have not been fully completed yet), EDF runs the task which has the earliest absolute deadline. The processor is then said to be busy. To decide between tasks having the same absolute deadline, EDF makes use of a tie-breaking rule (e.g., a arbitrary order). If there are no pending tasks, EDF runs no task. The processor is then said to be idle (see example in figure 4). In this paper, we consider preemptive versions for EDF and its associated analytical feasibility conditions.

The processor demand $h(p,t,\tau)$:

$$h(p,t,\tau) = \sum_{j=1}^{n} h(p,t,j)$$

$$= \sum_{j\in\tau}^{n} Max\left\{0, 1 + \left\lfloor \frac{t-D_j}{T_j} \right\rfloor\right\} C_j(p) \quad (6)$$

By definition, the processor p demand $h(p,t,\tau)$ is the amount of time that is needed to run all the tasks whose activation times and absolute deadlines are in $[0,t]$ (Baruah et al. 1990b). To give the expression of $h(p,t,\tau)$, we consider the synchronous concrete traffic $\omega \in \tau$.

A necessary and sufficient feasibility condition (NSC):

$$\tau \text{ is feasible by EDF} \Leftrightarrow \forall t \in \mathbb{R}^+,\ h(p,t,\tau) \leq t; \quad (7)$$

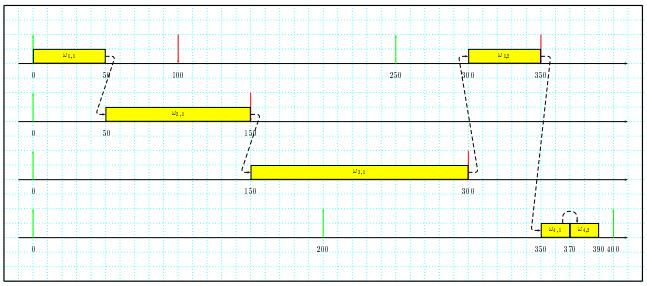
$$\tau \text{ is feasible by EDF} \Leftrightarrow Sup_{t\in\mathbb{R}^{+*}} \left\{ \frac{h(p,t,\tau)}{t} \right\} \leq 1. \quad (8)$$

sketch of the proof:

By definition, the processor demand $h(p,t,\tau)$ is the amount of time that is needed to run all the tasks whose activation times and absolute deadlines are in $[0,t]$. τ is feasible by EDF, if and only if, $\forall p \in [0,\mathbf{P}_{max}), \forall t \in \mathbb{R}^+,\ h(p,t,\tau) \leq t$, i.e., if and only if, $Sup_{t\in\mathbb{R}^{+*}} \{h(p,t,\tau)/t\} \leq 100\%$.

Study interval: In order to be operationally satisfied, feasibility constraints cannot be stated on $\mathbb{R}^{+*}$. Although the processor busy period λ (Hermant et al. 96),(Hermant 98) is a candidate interval, the resulting constraints are not tractable in a CLP model. Instead, the study interval is set to $L = \underset{i\in\tau}{gcm}(T_i)$ and is preprocessed. Therefore, equation (8) becomes:

$$\tau \text{ is feasible by EDF} \Leftrightarrow Sup_{t\in(0,L)} \left\{ \frac{h(p,t,\tau)}{t} \right\} \leq 1 \quad (9)$$



$$U(0,\tau) = \sum_{i=1}^{n} \frac{C_i}{T_i} = 0.8 \quad (10)$$

$$\lambda = \sum_{i=1}^{n} \left\lceil \frac{\lambda}{T_i} \right\rceil C_i = 390ms \quad (11)$$

$$\forall t \in [0, 390),\ h(0,t,\tau) \leq t \quad (12)$$

Figure 4: Mapping of the `detection system` onto a single processor

Example of feasible mapping: A feasible mapping of the `detection system` onto a single processor can be derived. The utilization factor $U(0,\tau)$ is given in (10). Instead of L, the length of the busy period λ is used as the study interval (11). Lastly, the necessary and sufficient feasibility condition for EDF (12) holds true, illustrated by a feasible schedule of the synchronous activation scenario (worst cases) in fig. 4.

Using exact feasibility constraints for HPF At any time $t \in \mathbb{R}^+$, if there are pending tasks (i.e., tasks which have been previously activated but which have not been fully completed yet), HPF runs the task which has the highest priority (Liu and Layland 73). Priorities can be allocated according to deadlines (said to be deadline monotonic) or period (also called rate monotonic). The highest priority correponds respectively to the lower deadline or the lower period. Priorities can also be computed statically in order to optimize the execution (Audsley 1991), for example, by maximizing the workload. As for EDF, we consider a periodic/sporadic non-concrete traffic τ, which is a set of n periodic/sporadic non-concrete traffics τ_i. When a set of tasks is allocated to the same processor, a priority order is associated to τ, as follows.

Priorities:

$$\forall i \in \tau, \forall j \in \tau / i \neq j,\ \exists p\ /$$
$$(m_i^p = 1) \wedge (m_j^p = 1) \Leftrightarrow\ (j \prec i) \nabla (i \prec j) \quad (13)$$

where ∇ the exclusive or.

Response time $r(p,t,i,\tau)$**:** To establish necessary and sufficient conditions as schedulability constraints for HPF (Lehoczky 90), (Tindell and al. 94), one has to consider the worst case response time $r(p,t,i,\tau) \in \mathbb{N}$ associated to task i and processor p. The task set is feasible, if and only if, for each task i, at any activation t, the response time meets the deadline D_i:

$$\tau \text{ feasible by HPF} \Leftrightarrow$$
$$\forall p \in [0, \mathbf{P}_{max}), \forall i \in \tau, \forall t \in [0,\infty), r(p,t,i,\tau) \leq D_i$$

Workload $w(p,t,i,\tau)$**:** For each task, the response time can be rewritten using the workload $w(p,a,i,\tau) \in \mathbb{N}$, as follows:

$$\forall i \in \tau, \forall p \in [0, \mathbf{P}_{max}), \forall t \in [0,\infty),$$
$$r(p,t,i,\tau) = w(p,t,i,\tau) - t$$

Let us consider a task i (of priority i). In $[0,t]$, the maximum number of executions of task i is $1 + \lfloor t/T_i \rfloor$. For each task j (of lower priority $j \prec i$), in $[0, w(p,t,i,\tau))$, the maximum number of executions of task j is $\lceil w(p,t,i,\tau)/T_j \rceil$. Hence, the workload can be written as follows:

$$w(p,t,i,\tau) = \left(1 + \left\lfloor \frac{t}{T_i} \right\rfloor\right) C_i + \sum_{j \in \tau / j \prec i} \left\lceil \frac{w(p,t,i,\tau)}{T_j} \right\rceil C_j(p) \quad (14)$$

The resulting necessary and sufficient condition: By replacing $r(p,t,i,\tau)$, it follows that:

$$\tau \text{ feasible by HPF} \Leftrightarrow \forall i \in \tau, \forall t \in [0,\infty), \forall p \in [0, \mathbf{P}_{max}),$$
$$w(p,t,i,\tau) \leq t + D_i \quad (15)$$

This set of equations converges to a fixed point, which can be solved using the fixed point semantics of CLP languages. A tractable interval for activations $[0,t)$ must be specified for implementing the constraint using a CLP language. First, a maximal study interval can be specified using the greater common multiple of the periods/sporadicities. Second, for each task, activations can be formulated using the associated period/sporadicity according to the worst case analysis. This leads to:

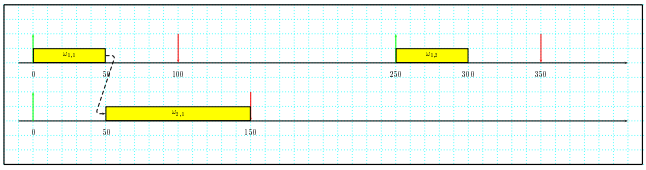
$$\forall i \in \tau,\ t \in [0, \underset{j \in \tau}{lcm}(T_j))$$
$$\forall i \in \tau, \forall t \in [0, \underset{j \in \tau}{lcm}(T_j)), \exists q_i \mid t = q_i.T_i \Rightarrow \quad (16)$$
$$\forall i \in \tau, \forall q_i \in [0, \underset{j \in \tau}{lcm}(T_j)/T_i) \quad (17)$$

As for EDF (in eq. 9), the lcm constraint, which has not received efficient implementation yet, is preprocessed. Here again, upper-approximation may be found in order to keep the global problem tractable by CLP. Using constraints (16,17) constraint (14) can be simplified:

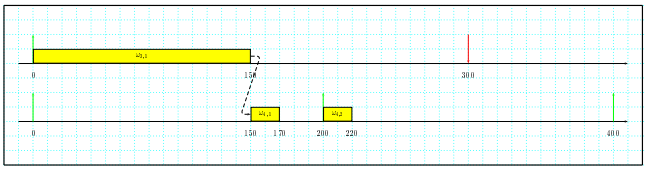
$$w(p,q_i,i,\tau) = (1+q_i)C_i + \sum_{j \in \tau / j \prec i} \left\lceil \frac{w(p,q_i,i,\tau)}{T_j} \right\rceil C_j(p) \quad (18)$$

such that the NSC (15) becomes:

Processor 0:



Processor 1:



$$U(t,0,\tau) = U(t,1,\tau) = 40 \quad (21)$$

$$\lambda_0 = 150 \text{ ms and } \lambda_1 = 170 \text{ ms} \quad (22)$$

Figure 5: Mapping of the `detection_system` using HPF/DM or HPF/RM and HPF/DM onto two processors.

$$\tau \text{ feasible by HPF} \Leftrightarrow$$
$$\forall i \in \tau, \forall q_i \in [0, \underset{j\in\tau}{lcm}(T_j)[, \forall p \in [0, \mathbf{P}_{max}),$$
$$w(p,q_i,i,\tau) \leq q_i.T_i + D_i \quad (19)$$

Lastly, the priority order (13) may not be instantiated. Therefore, the formulation (18) leads to the enumeration of all priority order without the consideration of constraint propagation. A more efficient approach is to model the priority order using a permutation matrix P, such that statement (18) $\Leftrightarrow$ (20):

$$P \in \{0,1\}^{card(\tau)\times card(\tau)}$$
$$\vec{C'} = P.\vec{C},\ \vec{T'} = P.\vec{T},\ \vec{w'}(p,q_i,\tau) = P.\vec{w}(p,q_i,\tau)$$
$$w'(p,q_i,i,\tau) = (1+q_i)C'_i + \sum_{j\in[0,i)} \left\lceil \frac{w'(p,q_i,i,\tau)}{T'_j} \right\rceil C'_j(p) \quad (20)$$

Example of feasible mapping: An optimal mapping (in terms of the number of processors) of the `detection system` can be given on two processors, using a HPF policy (fig. 5). It considers tasks `insert_target` and `distance_eval` placed onto processor 0, while tasks `pursuit_target` and `suppress_target` are placed onto processor 1. For processor 0, task `insert_target` is assigned the highest priority, while `task distance_eval` is assigned the lowest. This assignment corresponds to the Deadline Monotonic (DM) or the Rate Monotonic (RM) fixed priority assignment. For processor 1, task `pursuit_target` is assigned the highest priority, while task `suppres_target` is assigned the lowest. This assignment corresponds to the Deadline Monotonic fixed priority assignment.

This placement defined can also be computed automatically using the CLP models implementation.

Automatic Solving Using CLP Language

Several problems can be solved using models presented. Specific periods, deadlines, response or computation times that would be related to an applicative function (such that the worst case durations of the UAV control tasks) can be solved. However, within the scope of this paper, the focus is on a non-functional requirement: the minimal number of required processors $\mathbb{P}$.

Search strategies

This section presents the main design concepts for search strategies dedicated to the scheduling and mapping problem under consideration.

Elementary search The elementary search strategy is based on the `labeling` and Branch & Bound predicates, provided in most of CLP implementations (Carlsson et al. 97). Even when problem models are tractable, these basic strategies are too weak to cope with the largest problem instances.

Load balancing strategies So called load balancing heuristics can be introduced as static heuristics, or as a dynamic search strategy. These heuristics and strategies can be used for both HPF and EDF scheduling policies.

Static heuristic: This strategy is similar to elementary search, but disjunctive constraints on the utilization factor are added in order to statically decompose the search space. This strategy structures the search space in favour of assignements that do not under-utilize processors (also called no starvation heuristic).

Dynamic balancing search strategy: Instead of using standard `labeling` predicates, this strategy reorders dynamically[2] the set of variables to explore. Each time a placement variable m_q^j is successfully instantiated (which indicates that task j is placed on processor q), a set of variables $M_p = \{m_p^1 \dots m_p^n\}$ is selected according to the least utilized processor p. The selection function tests the minimal bounds of constraint variables $\{U(k,\tau)\}_0^{\mathbf{P}_{max}}$. Then, the strategy attempts to instantiate one variable of the set M_p. If one variable can be successfully instantiated, the strategy starts over until all the tasks are placed. Otherwise, it backtracks and another processor is selected according to the increasing order of utilization factors.

Uniprocessor optimal priority assignment heuristic: The last search strategy to be investigated relies on the uniprocessor optimal priority assignment (PA) (Audsley 1991). Using this method, only the problems involving HPF scheduling policies can be improved. Far from being optimal on multi-processor problems, this PA method can nevertheless be used as a heuristic. The search strategy first

[2]Here, the term 'dynamic' does not mean that the strategy is performed online, but that the construction of the search tree is dynamic (e.g. during the search itself).

enumerates the set of placement variables, for each processor and each task m_p^i. Then, for each processor, the optimal PA is performed. It consists of instantiating tasks of a highest priority first, without backtracking on these assignments (Audsley 1991). Still complete, this technique obviously simplifies the search complexity. The PA heuristic can be used in combination with both static and dynamic balancing strategies but may conflicts with the load balancing strategies.

Experiments

Six combinations of strategies (figure 6) and scheduling policies have been experimented on 522 problem instances, optimising for the number of processors. The elementary strategy without any heuristic does not give meaningful results in reasonable time, until the static load heuristic is added.

Problem generation method

The experimentation method relies on the real world examples. A problem generation technique operates by iteratively decreasing of 5% the task deadlines until problem instances becomes infeasible for the maximum number of processors. This way, problem instances become more difficult by decreasing the global laxity of the hard real-time constraints. Experiments are performed on a 1Ghz Intel III processor, with 256 MBytes of main memory and WindowsXP Pro and SICStus Prolog 3.9.0, using finite domain constraint library (Carlsson et al. 97).

Global results

The subset of experiments under consideration (figure 6) enables the comparison of the different policies (namely EDF and HPF) for the mapping problem, as well as the various strategies proposed. Figure (figure 6) includes data gathered from the three benchmarks. On this figure, the column *best solutions* is the number of instances for which at least one solution is retrieved. The column *proof of completeness* is the number of instances for which the problem is solved optimally or for which it can be proved that no feasible solution exist.

policy, strategy	number of instances	best solution	proofs of completeness
EDF, static load	87	82	62
HPF, static load	87	65	26
HPF, static load, PA	87	76	39
EDF, dynamic load	87	86	66
HPF, dynamic load	87	66	26
HPF, dynamic load, PA	87	57	41

Figure 6: Experiments under consideration

At first glance, experiments suggest that solving the mapping problem with an EDF scheduling policy is easier than with its HPF equivalent. Globally, the priority assignment heuristic improves the search efficiency (for HPF cases), when the static load heuristic is activated. For HPF cases, and without the priority assignment heuristic, the strategies fail to prove completeness for any of the problem instances in benchmark `spacecraft_system` and `uav_avionic` (completeness can be proved only for the 26 easy instances of the `detection_system` benchmark).

When considering the EDF policy, the number of problems solved is 5% better (6% better for the proofs of completeness) by replacing the static load balancing heuristic with a dynamic search. This is also the case for the HPF policy using the priority assignments heuristic, but only for the proof of completeness (41 versus 39 instances). As a result of the conflicting strategies, the dynamic load strategy combined with the PA heuristic gives a reduced performance on the number of problems solved (57 versus 66 instances).

Evaluation of solutions

Figure 7 compares the number of processors found for the two non-trivial benchmarks (`spacecraft_system` and `uav_avionic`), when decreasing the global laxity. Using the EDF policy, search strategies find a smaller number of processors. Furthermore, using this policy, more problems can be (optimally) solved in the time limit imposed. By convention, when the strategy fails to give any feasible solution or to prove that the problem instance is not feasible, a vertical line is drawn. As for the global results, the priority assignment heuristic improves the costs, except for the conflicting load balancing strategy.

Strategy performances

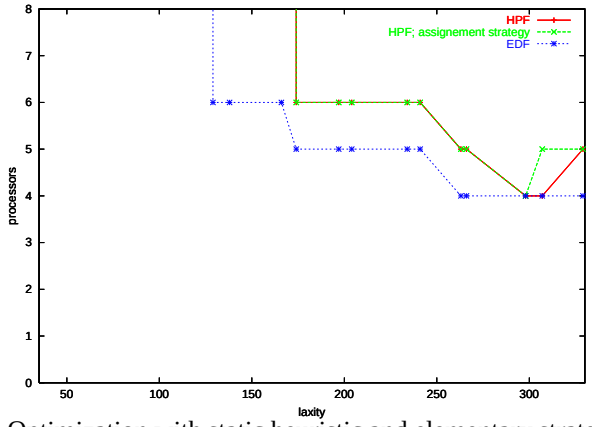
For the purpose of the experiments, a time-out is imposed on the different runs. Therefore, experiments that successfully yield to a proof of optimality (mainly on benchmarks `detection_system` and `spacecraft_system`) are separated from experiments that lead to optimized solutions, without guarantee of optimality (mainly on benchmarks `spacecraft_system` and `uav_avionic`).

Time for proving optimality Figure 8 gives the different completion times for benchmark `spacecraft_system` to prove solution optimality according to the different search strategies. Concerning the benchmark `detection_system`, the proof of optimality can be performed for all the experiments with 2 and 4 processors. However, due to the little differences between solving duration and the small problem size, the curves are not represented in this paper.

With the EDF policy, proof of completeness can be performed on all the problem instances of the `spacecraft_system` in figure 8, and similarly on parts of the problem instances of the `uav_avionic`. In general, the time for proving the optimality is not modified when replacing the load balancing heuristic with the dynamic load balancing strategy. For the benchmark `spacecraft_system`, and using the priority assignment heuristic, proofs of optimality can be performed for a subset of instances. The dynamic load balancing strategy improves the solving time only for two instances. Without the priority assignement heuristic, strategies fail to prove completeness for the HPF scheduling policy in the time limit (7 seconds).

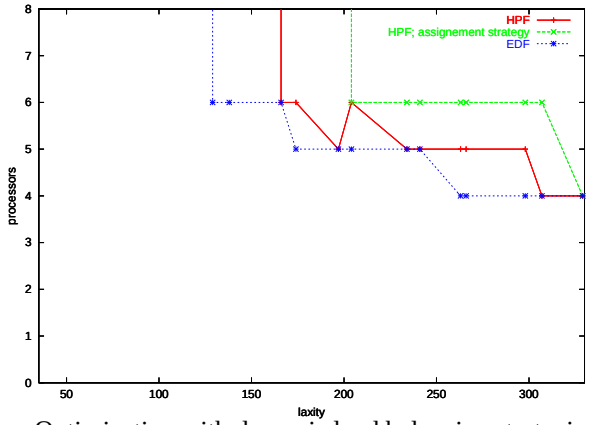
Time for finding an optimized solution Figure 9 gives the completion times for benchmark `uav_avionic` to find

a.



Optimization with static heuristic and elementary strategy

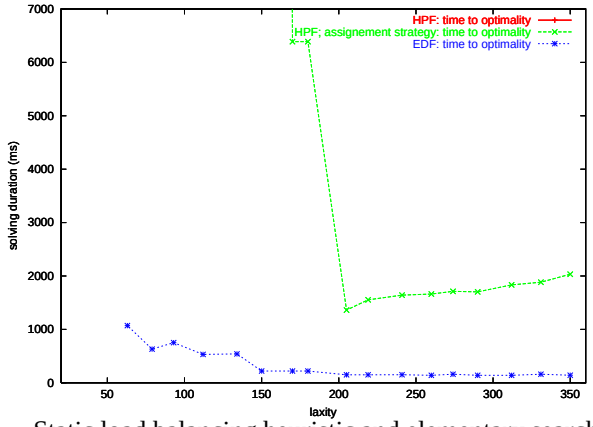
b.



Optimization with dynamic load balancing strategies

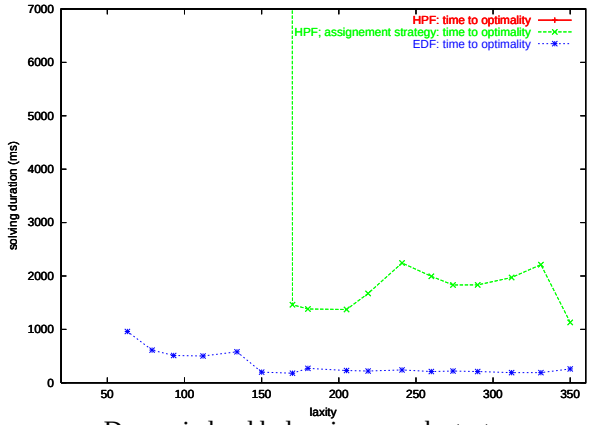
Figure 7: The `uav_avionic` example (6 processors): comparison of the minimal number of processors with different search strategies

a.



Static load balancing heuristic and elementary search

b.



Dynamic load balancing search strategy

Figure 8: Time to complete proof of optimality on the `spacecraft_system` example (4 processors)

an optimized solution. The time out range is 20 seconds. Again by convention, a vertical line is drawn when the solving strategy fails to produce any feasible solution or to prove that the problem instance is not feasible. It is not obvious that the dynamic load balancing strategy improves the solving duration on 9-a. Furthermore, as shown on figure 9-b, the conflict of this strategy with the priority assignment heuristic gives a counter-performance (instances under 200 of laxity cannot be solved). In contrast, for the EDF policy, strategies are improved with the load balancing heuristic, even for the set of non-feasible instances (as shown on figures 9-a and 9-b).

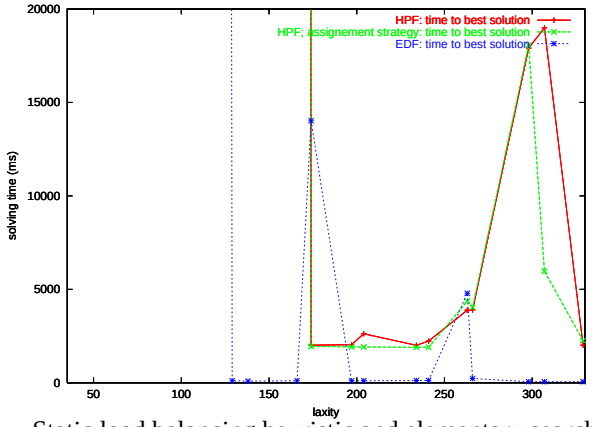
Conclusions and Further Work

This paper demonstrates that the necessary and sufficient conditions can be preserved while modelling the feasibility of HPF and EDF preemptive scheduling policies using a CLP language. From a Constraint Programming point of view, this represents an interesting alternative to the fully static approach generally considered, where preemptive scheduling is solved completely offline. The constraint-based modelling described in this paper has a larger applicative impact, as many real-time operating systems for embedded applications make use of HPF policies. Furthermore, using these constraint-based models, various mapping problems can be solved. On realistic benchmarks, experiments have shown that proofs of completeness can be found in reasonable time.

The paper illustrates how to support the engineering of distributed real-time systems using a proof-based method. As a matter of fact, a generic heuristic, such as load balancing performs better with EDF. With other HPF policies, the benefit of this heuristic is not clear, even using dynamic search techniques that are generally recognized as strong strategies. It is very difficult to conclude on the gain of the priority assignment heuristic, although dedicated to the HPF scheduling policy.

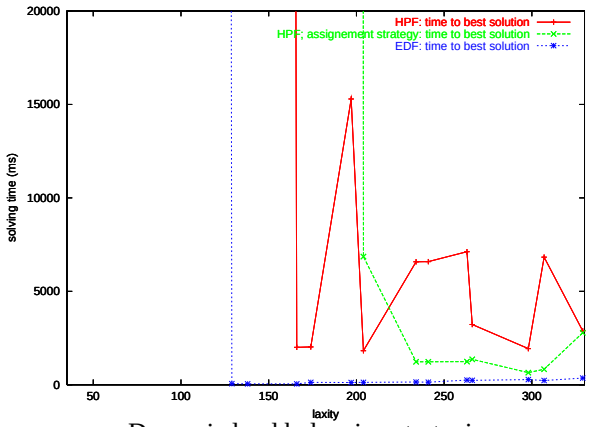
Most importantly, the paper shows for real world examples that the design of distributed real-time architectures is simpler using EDF policies. On various problem instances with the EDF policies, strategies have been able to prove search completeness. As a result, the solution optimality

a.



Static load balancing heuristic and elementary search

b.



Dynamic load balancing strategies;

Figure 9: The `uav_avionic` example (6 processors): time to retrieve optimized solutions using the different strategies

or the non feasibility of these problem instances can be decided, proven and guaranteed. For the same instances, strategies fail to prove search completeness using HPF policies. This paper is a first step towards the promising combination of offline and online multi-processor scheduling techniques. The modelling of communication protocols shall be part of further works.

References

N. C. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times*, Tech. Report YCS 164, Dept. of Computer Science, University of York, December 1991.

S. Baruah, A. Mok, L. Rosier, *Preemptively scheduling hard real-time sporadic tasks on one processor*, 11th Real-Time Systems Symposium, pp. 182-190, 1990.

M. Dertouzos, *Control Robotics: the procedural control of physical processors*, Proceedings of the IFIP congress, pp. 807-813, 1974.

J.-F. Hermant, *Analysis of Real-Time Distributed Scheduling Algorithms*, PhD Thesis, University of Paris VI, 1999.

J.-F. Hermant, L. Leboucher, N. Rivierre, *Real-time fixed and dynamic priority driven scheduling algorithms: theory and experience*, INRIA Research Report 3081, 142 p., 1996.

J. P. Lehoczky, *Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines*, 11th IEEE Real-Time Systems Symposium, pp. 201-209, Dec. 1990.

G. Le Lann, *Proof-Based System Engineering for Computing Systems*, 1^{st} Joint ESA/INCOSE Conference on Systems Engineering - The Future, IEE/ESA Pub., Vol. WPP-130, 5a.4.1-5a.4.8., Nov. 11-13, 1997.

G. Le Lann, *Proof-Based System Engineering and Embedded Systems*, School on Embedded Systems, Veldhoven (NL), Nov. 1996, Lecture Notes in Computer Science, vol. 1494, Springer Verlag Pub., pp. 208-248, 1998.

C.L. Liu, J.W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the Association for Computing Machinery, 20(1), pp. 40-61, 1973.

A.K. Mok, *Fundamental design problems for the hard real-time environments*, PhD, MIT/LCS/TR-297, 1983.

K. W. Tindell, A. Burns, A. J. Wellings, *An extendible Approach for Analyzing Fixed Priority Hard Real-Time Tasks*, The Journal of Real-Time Systems, Vol. 6(2), pp. 133-152, Klüwer Academic Publishers, March 1994.

C. Ancourt, D. Barthou, C. Guettier, F. Irigoin, B. Jeannet, J. Jourdan, and J. Mattioli, *Automatic mapping of signal processing applications onto parallel computers*, In Proc. ASAP 97, Zurich, July, 1997.

R. Bixby, K. Kennedy, and U. Kremer, *Automatic Data Layout Using 0-1 Integer Programming*, In Proc. PACT94, Montreal, Canada, August, 1994.

A. Bakic, M. W. Mutka, D. T. Rover, *Using Constraint Logic Programming for Engineering of Real-Time Systems*, Michigan University Technical Report, MSU-CPS-98-23, 1998.

C. Guettier, J.-F. Hermant *A Constraint-Based Model for High-Performance Real-Time Computing*, In Proc. of the Intl. Conf. on Distributed and Parallel Systems, Florida, September, 1999.

C. Guettier, B. Patin and J.-F. Tilman *Validation of Autonomous Concepts using the ATHENA environment*, In Proc. of the European Space Technology and rEsearch Center (ESTEC) Workshop On-board Autonomy, October, 2001.

U. Kremer, Optimal and Near-Optimal *Solutions For Hard Compilation Problems*, Parallel Processing Letters 7(4), World Scientific Publishing Company, 1997.

L. Thiele, *Partioning Processor Arrays under Resource Constraints*, In Journal of VLSI Signal Processing Systems, 15, pp. 5-21, 1997, Klüwer Academic Publishers, Boston.

J.-F. Tilman, S. Truong and J.-L. Teraillon, *Optimized Distribution of Real-Time Tasks with Resource Constraints* European Space TEchnology and research Center (ESTEC), research contract report number 15235/01/NL/SVH Noordwjik, Netherlands.

M. Carlsson, G. Ottosson and B. Carlson, *An Open-Ended Finite Domain Constraint Solver3*, In Proc. of Programming Languages: Implementation, Logics, and Programs, 1997.

M. Gondran and M. Minoux, *Graphes et algorithmes*, ed. Eyrolles, 1995.

P. Van Hentenryck, V. Saraswat, and Y. Deville, *Design, Implementation and Evaluation of the Constraint Language CC(FD)*, In Constraint Programming: Basics and Trends, A. Podelski Ed., Chatillon-sur-Seine, Springer-Verlag LNCCS 910, pp. 68-90, 1995.