Multi-source software on multicore automotive ECUs - Combining runnable sequencing with task scheduling

Aurélien Monot, Nicolas Navet, Bernard Bavoux and Françoise Simonot-Lion

***Abstract*—As the demand for computing power is quickly increasing in the automotive domain, car manufacturers and tier-one suppliers are gradually introducing multicore ECUs in their electronic architectures. Additionally, these multicore ECUs offer new features such as higher levels of parallelism which ease the compliance with safety requirements such as the ISO 26262 and the implementation of other automotive use-cases. These new features involve greater complexity in the design, development and verification of the software applications. Hence, car manufacturers and suppliers will require new tools and methodologies for deployment and validation. In this paper, we address the problem of sequencing numerous elementary software modules, called runnables, on a limited set of identical cores. We show how this problem can be addressed as two sub-problems, partitioning the set of runnables and building the sequencing of the runnables on each core, which problems cannot be solved optimally due to their algorithmic complexity. We then present low complexity heuristics to partition and build sequencer tasks that execute the runnable set on each core. Finally, we address the scheduling problem globally, at the ECU level, by discussing how to extend this approach in the case where other OS tasks are scheduled on the same cores as the sequencer tasks.**

I. Introduction

Multi-source software running on the same ECU (Electronic Control Unit) is becoming increasingly widespread in the automotive industry. One of the main reasons being that car manufacturers want to reduce the number of ECUs which grew up above 70 for high-end cars. One major outcome of the AUTOSAR initiative, and more specifically of its operating system, is to help car manufacturers shift from the "one function per ECU" paradigm to more centralized architecture designs by providing appropriate protection mechanisms.

Another crucial evolution in the automotive industry is that chip manufacturers are reaching the point where they can no longer cost-effectively meet the increasing performance requirements through frequency scaling alone. This is one reason why multicore ECUs are being gradually introduced in the automotive domain. The higher level of performance provided by multicore architectures may help to simplify in-vehicle architectures by executing on multiple cores the software previously run on multiple ECUs. This possible evolution towards more centralized architectures is also an opportunity for car manufacturers to decrease the number of network connections and buses. As a result, parts of the complexity will be transferred from the electrical/electronic architecture to the hardware and software architecture of the ECUs. However, static cyclic scheduling makes it easy to add functions to an existing ECU. In practice, important architectural shifts are hindered by the carry-over of ECUs and existing sub-networks which is widely used by generalist car manufacturers. The extent to which more centralized architectures will be adopted remains thus unsure.

Multicore ECUs are also helpful for other use cases. For instance, they bring major improvements for some applications requiring high performance such as high-end engine controllers, electric and hybrid powertrains[1], [2], advanced driver assistance systems [3] sometimes involving real-time image processing [4]. Those multicore platforms offer also additional benefits, such as higher level of parallelism allowing for more segregation, which may help to meet the requirements of the upcoming ISO 26262 that concerns functional safety for road vehicles. Furthermore, in multicore architectures, some core can be dedicated to a specialized usage such as handling low-level services.

A. Monot is with LORIA/INPL and PSA Peugeot-Citroën, 615 rue du Jardin Botanique, 54600 Vandoeuvre, France, e-mail: aurelien.monot@inria.fr.

N. Navet is with INRIA and RealTime-at-Work (RTaW), 615 rue du Jardin Botanique, 54600 Vandoeuvre, France, e-mail: nicolas.navet@inria.fr.

B. Bavoux is with PSA Peugeot-Citroën, route de Gisy, 78 943 Vélizy-Villacoublay, France, e-mail: bernard.bavoux@mpsa.com.

F. Simonot-Lion is with LORIA/INPL, 615 rue du Jardin Botanique, 54600 Vandoeuvre, France, e-mail: simonot@loria.fr.

Now, the challenge is to adapt existing design methods to the new multicore constraints. The scheduling of the software components is one of the key issues in that regard and it has to be revamped.

Introduction of multi-source and multicore will induce drastic changes in the software architecture of automotive ECUs. The section II of this paper introduces the most likely scheduling choices as well as the literature relevant to the task scheduling in multiprocessor automotive ECUs. Then, section III presents solutions for the scheduling of numerous software modules when only a few OS tasks are allowed. This paper builds on the study published in [5] where it was assumed that only one sequencer task was running on each core of the ECU to schedule the runnables. Here, we consider in section V how to build several sequencer tasks while possibly scheduling other tasks on the same core and discuss how to analyze the schedulability of such systems globally.

From now on, for the sake of clarity, we refer to "sequencing" when talking about the scheduling of runnables while "scheduling" is solely used for tasks.

II. Scheduling in the automotive domain

A. Scheduling design choices for multicore ECUs

In this section, we explain and justify, especially in the light of predictability requirements, the multicore scheduling approach that is, to the best of our knowledge, the most widely considered in the automotive industry.

1) Partitioning scheduling scheme: In a multicore system, the tasks are either statically allocated to the cores or they can be distributed dynamically at runtime to balance the workload or migrate functions to increase availability. The later approach involves complex task and resource interactions which are difficult to predict and validate. For this reason, approaches relying on static allocation (*i.e.*, partitioning) and deterministic mechanisms such as periodic cyclic scheduling are more likely to be used in the automotive context and this is the option taken within the AUTOSAR consortium. Scheduling tasks on a multi-processor systems under the static partitioning approach has been well studied for a long time, see for instance [6] and [7], [8], [9]. However, the works we are aware of deal with online algorithms such as FPP or EDF, and do not consider the static cyclic scheduling of tasks.

2) Static cyclic scheduling: Static cyclic scheduling of elementary software modules, or runnables, is common because there are usually many more runnables than the maximum number of tasks allowed by automotive operating systems such as OSEK/VDX or AUTOSAR OS. For this reason, runnables must be grouped together and scheduled within a sequencer task (also called dispatcher task). In this paper, we focus on how to sequence large runnable sets on multicore platforms using a static partitioning approach. Indeed, the static task partitioning scheme is very likely to be adopted at least in a first step because it is conceptually simple and provides a better predictability for the ECU designers by comparison with a global scheduling approach. We aim to develop practical algorithms, whose performances can be guaranteed, to build the dispatcher tasks on each core and to schedule the runnables within these dispatcher tasks so as to respect sampling constraints and, as far as possible, uniformize the CPU load over time. This latter objective is of course important to minimize the hardware cost and to facilitate the addition of new functions, as typically done in the incremental design process of car manufacturers. This is achieved by desynchronizing the runnable release dates. Precisely, the first release date of each runnable, termed its offset, is determined so as to spread the CPU demand uniformly over time. The configuration algorithms developed in this paper are closely related to [10] (mono-processor scheduling of tasks with offsets) and [11] (scheduling of frames with offsets) but it is applied to multi-core and goes beyond as we provide lower-bounds on the performances. As the problem is of practical interest in the industry, there are in-house tools at the car manufacturers as well as commercial tools, such as RealTime-at-Work's RTaW-ECU [12], that have been developed for configuring the scheduling. However, the proprietary algorithms used in these tools can usually not be disclosed and they are sometimes specialized for some specific usage.

B. Model description

In this study, we consider a large set of *n* periodic elementary software modules, also called runnables, that are to be allocated on an ECU consisting in *m* identical cores. In practice, a runnable can be implemented as a function called, whenever appropriate, within the body of an OS task.

1) Runnable characteristics: The ith runnable is denoted by $\mathcal{R}_i = (C_i, T_i, O_i, \{R\}, P_i)$. Quantities C_i, T_i and O_i correspond respectively to the Worst-Case Execution Time (WCET), the period (i.e., the exact time between two successive releases) and the offset of the $\mathcal{R}_i$. As shown in Figure 1, the offset of a runnable is the release date of the first instance of that runnable, subsequent instances are then released periodically. The choice made for the offset values has a direct influence on the repartition of the workload over time.

A set of inter-runnable dependencies is denoted by $\{R\}$. Indeed, due to specific design requirements, such

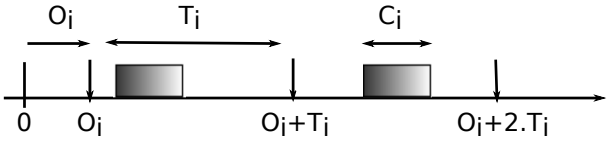

Figure 1. Model of the runnables. After its release, an instance of a runnable has to be executed before the next instance is released (*i.e.*, the deadline is set to the period).

as shared variables, some runnables may have to be allocated on the same core and the set $\{R\}$ is used to capture those constraints. In addition, some specific features, as I/O ports being located on a given core, may require a runnable to be allocated onto a specific core. This locality constraint is expressed by P_i.

2) Dispatcher task: Runnables are scheduled on their designated core using a dispatcher task, or "sequencer task", that stores the runnable activation times in a table and releases them at the right points in time. A dispatcher task is characterized by the duration of the dispatch table T_{cycle} that is executed in a cyclic manner, and by a quantum T_{tic} which is the duration of a slot in the table. Typically, one may have for instance $T_{cycle} = 1000ms$ and $T_{tic} = 5ms$. It should be noted that T_{cycle} must be a multiple of the *greatest common divisor* (*gcd*) of the runnable periods and the *least common multiple* (*lcm*) of these periods must be a multiple of T_{tic}. As a result, a dispatch table holds T_{cycle}/T_{tic} slots.

3) Assumptions: In this paper, we place a set of working assumptions, which, in our experience, can most often be met in today's automotive applications:

- Each runnable is executed strictly periodically. As a result, the whole trajectory of the system is defined by the first activation times of the runnables (*i.e.*, their offsets).
- The runnables are assumed to be offset-free, in the sense that the initial offset of a runnable can be freely chosen in the limit of its period (see [10]). Those offsets will be assigned during the construction of the dispatch table with the objective to uniformize the CPU load over a scheduling cycle.
- The worst case execution times of the runnables are assumed to be small compared to T_{tic}. Typical values for the case we consider would be $5ms$ for T_{tic} and $C_i \leq 300\mu s$.
- All cores are identical regarding their processing speed.
- There are no dependencies between runnables allocated on different cores. Therefore, all cores can be scheduled independently. This assumption is in line with the choices made by AUTOSAR regarding multicore architecture [13].

This last assumption allows to divide the overall problem into two independent sub-problems. A first part of the problem consists in allocating all of the *n* runnables onto the *m* cores with respect to their constraints with the aim of balancing the CPU load of the *m* resulting partitions (see §III-A). The second part of the problem consists in building the dispatch table for each core (see §III-B).

4) Schedulability condition: Assuming that we only consider runnable scheduling, the system is schedulable, and thus can be safely deployed, if and only if on each core:

1) The runnables are executed strictly periodically.
2) The initial offset of each runnable is smaller than its period.
3) The sum of the WCET of the runnables allocated in each slot does not exceed a given threshold, which is typically chosen as the duration of the slot, *i.e.* T_{tic}.

III. Runnable sequencing algorithms for multicore ECUs

In this section we present algorithms, and when possible derive lower bounds on their efficiency, to schedule large numbers of runnables on multicore ECUs.

Since automotive OSs can only handle a limited amount of OS-tasks, the sequencing of runnables has to be done within dispatcher tasks. A first step of the approach is to partition the runnable sets onto the different cores. The next and last step is to determine the offsets between the runnables allocated on each core so as to balance the load over time.

A. Building tasks as a bin-packing problem

It is assumed that the number of cores is fixed. We first distribute all the runnables on the cores. Assigning *n* tasks to *m* cores is like subdividing a set of *n* elements into *m* non-empty subsets. By definition, the number of possibilities for this problem is given by the Stirling number of the second kind (see [14]):$\frac{1}{m!}\sum_{i=0}^{m}(-1)^{(m-i)}\binom{m}{i}i^n$. Considering that the runnables may have core allocation constraints, the cores should be distinguished. Thus, the $m!$ combinations of cores must be considered. As a result, one has at most $\sum_{i=0}^{m}(-1)^{(m-i)}\binom{m}{i}i^n$ different possibilities for the partitioning problem alone. Such a complexity prevents us from an exhaustive search. For instance, with $n = 30$ and $m = 2$, the search space holds more than one billion possibilities.

Considering this complexity, to balance as evenly as possible the utilization of processor cores, we propose a heuristic based on the bin-packing decreasing worst-fit

scheme for a fixed number of bins (where "bins" here are processor cores). The heuristic is given in Algorithm 1.

Algorithm 1 Partitioning of the runnable set.

input: runnable set $\{\mathcal{R}_i\}$, number of cores m
(1) Group inter-dependent runnables into runnable clusters. Independent runnables become clusters of size 1.
(2) Allocate the runnable clusters which have a locality constraint to the corresponding cores.
(3) Sort the runnables clusters by decreasing order of CPU utilization rate $\rho = \sum_i \frac{C_i}{T_i}$.
(4) Iterate over the sorted clusters
  (a) Find the least loaded core,
  (b) Assign the current cluster to this core.

Step (1) runs in $\mathcal{O}(n)$. Step (2) runs in $\mathcal{O}(n)$ but all the runnables allocated in (2) will not have to go through the steps (3) and (4) that are algorithmically more complex. Step (3) runs in $\mathcal{O}(n \cdot \log n)$. Finally step (4) runs in $\mathcal{O}(n \cdot m)$. As a conclusion, algorithm 1 runs in $\mathcal{O}(n(m + \log n))$ which does not raise any issue in practical cases.

It is worth pointing out that $m \geq \left\lceil \sum_{i=1}^{m} \frac{C_i}{T_i} \right\rceil$ is a necessary schedulability condition which can be used to rule out configurations with too few processor cores.

B. Strategies for sequencing runnables

The next stage consists in building the dispatch table for the set of runnables. In a first step, it is assumed that there are no precedence constraints between the runnables and that a single sequencer table is needed per core. This latter assumption can be easily relaxed as done in section V.

1) Least-loaded algorithm: Considering a runnable R_i of period T_i, there are $\frac{T_i}{T_{tic}}$ possibilities for allocating this runnable (see schedulability condition #2 in §II-B4). As a result there are $\prod_{i=1}^{n} \frac{T_i}{T_{tic}}$ alternative schedules for the n runnables and, given the cost function, we are not aware of any ways to find the optimal solution with an algorithm that does not have an exponential complexity. Considering a realistic case of 50 runnables having their period as least twice as large as T_{tic}, it would be needed to evaluate a minimum of 2^{50} possible solutions. Once again, given the complexity, we have to resort to a heuristic. Here, we adapt to the problem of sequencing runnables the "Least-loaded" algorithm proposed by Grenier et al. in [11] for the frame offset allocation on a CAN network.

The intuition behind the heuristic is simple: at each step, we assign the next runnable to the least loaded slot, as described in Algorithm 2. The load of a slot is the sum of the C_i of the runnables $\{\mathcal{R}_i\}$ already assigned to this slot. This algorithm is further referred to as "Least loaded" or LL for short.

Algorithm 2 Assigning runnables to slots: the "Least-loaded" heuristic.

input: runnable set $\{\mathcal{R}_i\}$, T_{tic}, T_{cycle}
(1) Sort runnables $\mathcal{R}_i$ such that $T_{tic} \leq T_1 \leq \ldots \leq T_n \leq T_{cycle}$.
(2) For $i = 1 \ldots n$
  (a) Look for the least loaded slot in the $\frac{T_i}{T_{tic}}$ first slots,
  (b) Allocate $\mathcal{R}_i$ in every $\frac{T_i}{T_{tic}}$ slot starting from this slot.

Step (1) runs in $\mathcal{O}(n \cdot \log n)$. Step (2) iterates n times over steps (2a) and (2b) which run respectively in $\frac{T_i}{T_{tic}} \leq \frac{T_{cycle}}{T_{tic}}$ and $\frac{T_{cycle}}{T_i} \leq \frac{T_{cycle}}{T_{tic}}$. As a result, this algorithm runs in $\mathcal{O}(n(\log n + \frac{\max_i\{T_i\}}{T_{tic}} + \frac{T_{cycle}}{\min_i\{T_i\}}) \leq \mathcal{O}(n(\log n + 2\frac{T_{cycle}}{T_{tic}})$.

For practical applications, ties at step (1) are broken using highest WCET first and ties at step (2a) by choosing the central slot of the longest sequence of consecutive slots having the minimum load. Although the latter rule for breaking ties does not have any impact on the theoretical results that will be derived next, it helps to separate load peaks, which is important from the ECU designer point of view. As an illustration, applying the least-loaded heuristic to the set of runnables $\mathcal{R}_i(T_i, C_i)$: $\mathcal{R}_1(10, 2)$, $\mathcal{R}_2(10, 1)$, $\mathcal{R}_3(20, 4)$, $\mathcal{R}_4(20, 2)$ leads to the dispatch table shown in Figure 2. The resulting distribution of the load is shown in Table I.

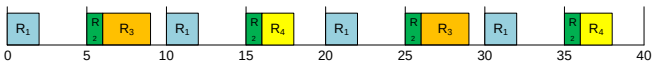

Figure 2. Example of dispatch table.

Slot	1	2	3	4	5	6	7	8
Load	2	4	2	3	2	4	2	3

Table I
LOAD REPARTITION CORRESPONDING TO THE DISPATCH TABLE IN FIGURE 2.

We consider two main metrics to evaluate the quality of a dispatch table. The first important criterion is to have the lowest maximum load in the cycle since this will determine the feasibility of the schedule and the possibility to add further functions later in the lifetime of the system. The maximum load over all slots is also referred to as the *peak load*. In a second step, a more fine-grained assessment of the uniformity of the load

balancing can be given by the standard deviation of the load distribution over all the slots.

2) Upper bound on the peak load: Here we derive an upper bound on the peak load which holds for runnable sets with harmonic periods (*i.e.*, each period is multiple of all smaller periods). From it, we consequently derive a closed-form sufficient schedulability condition. In this perspective, we first point out that the slots in which a runnable $\mathcal{R}_i$ will be periodically assigned are of equal load -which is the rationale behind step (2a) of algorithm 2.

Lemma 1. *Before the allocation of a runnable $\mathcal{R}_i$, the slot allocation induced by the previously allocated runnables repeats with a period $\frac{T_i}{T_{tic}}$.*

Proof: This is proved by induction. The property holds for $\mathcal{R}_0$ as all slots are empty. Assuming that the property holds for $\mathcal{R}_i$, this runnable will be periodically allocated in every $\frac{T_i}{T_{tic}}$ slots. Therefore, the slot allocation will still repeat with a period $\frac{T_i}{T_{tic}}$ after its allocation in the least loaded slot. Since runnables are sorted by increasing periods and that their periods are harmonic, $T_{i+1} = k \cdot T_i$ with $k \in \mathbb{N}^*$ and the slot allocation also repeats with a period $k \cdot \frac{T_i}{T_{tic}} = \frac{T_{i+1}}{T_{tic}}$ before the allocation of $\mathcal{R}_{i+1}$. ■

Therefore, the least loaded slot in the first $\frac{T_i}{T_{tic}}$ slots is the least loaded over the whole dispatch table and one does not need to look farther. As a second step, we show that the highest load in the slot where a runnable is going to be allocated arises when the load is equal in every slot.

Lemma 2. *The maximum load in the least loaded slot is obtained for a perfect load balancing, which corresponds to a constant load throughout the cycle.*

Proof: Reasoning with a constant allocated load, anything else than a perfect load balancing will result in a load below the average load per slot in some slot which will be eventually chosen to allocate the runnable under consideration. ■

As a result, the highest peak a runnable can create happens in the case of a perfect load balancing. Now, let us define $\rho_k = \sum_{i \in \{\mathcal{R}\}_k} \frac{C_i}{T_i}$ the total utilization of core k where $\{\mathcal{R}\}_k$ is the set of runnables allocated to core k and $\text{card}\{R\}_k$ the cardinality of $\{R\}_k$.

Theorem 3. *On processor k, an upper bound on the peak load of a slot allocation is*

$$PL_k = \max_{i \in \{\mathcal{R}\}_k} \{C_i + \rho_k T_{tic} - \sum_{j=i}^{\text{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j} T_{tic}\} \quad (1)$$

Proof: In the case of a perfect load balancing, before the allocation of $\mathcal{R}_i$, the load of a slot is given by: $\sum_{allocated\, runnables} wcet \cdot \frac{number\, of\, allocation\, slots}{total\, number\, of\, slots}$, *i.e.*

$$\sum_{j \in \{\mathcal{R}\}_k}^{i-1} C_j \cdot \frac{T_{cycle}}{T_j} \cdot \frac{T_{tic}}{T_{cycle}} \quad (2)$$

After the allocation of $\mathcal{R}_i$, the load in the corresponding slot is

$$C_i + \sum_{j \in \{\mathcal{R}\}_k}^{i-1} C_j \cdot \frac{T_{tic}}{T_j} \quad (3)$$

Moreover: $\sum_{j \in \{\mathcal{R}\}_k}^{i-1} \frac{C_j}{T_j} = \rho_k - \sum_{j=i}^{\text{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j}$

Consequently, the worst-case peak load on processor core k resulting of the allocation of $\mathcal{R}_i$ in a slot is

$$PL_k^i = C_i + \rho_k T_{tic} - \sum_{j=i}^{\text{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j} T_{tic} \quad (4)$$

Taking the max for all the runnables gives equation 1. ■

If the worst case peak load is below T_{tic} for all runnables, then the solution given by the algorithm is schedulable. Hence the following corollary:

Corollary 1. *From theorem 1, we derive the following sufficient schedulability condition:*

$$\rho_k \leq 1 + \frac{C_{min}}{T_{max}} - \frac{C_{max}}{T_{tic}} \quad (5)$$

with $C_{max} = \max_{i \in \{\mathcal{R}\}_k}\{C_i\}$, $T_{max} = \max_{i \in \{\mathcal{R}\}_k}\{T_i\}$ and $T_{min} = \min_{i \in \{\mathcal{R}\}_k}\{T_i\}$

Proof: $\forall i,\, C_i \leq C_{max}$ and $\forall i,\, \sum_{j=i}^{\text{card}\{\mathcal{R}\}_k} \frac{C_j}{T_j} \geq \frac{C_{min}}{T_{max}}$ gives :

$$\forall i, PL_k^i \leq C_{max} + \rho_k T_{tic} - \frac{C_{min}}{T_{max}} T_{tic} \quad (6)$$

The scheduling condition 3 in §II-B4 (*i.e.*, $PL_k^i \leq T_{tic}$) leads to the result. ■

This bound is achievable for $n \cdot k$ identical runnables with period equal to $k \cdot T_{tic}$ and load equal to C and a last runnable $\mathcal{R}_{n \cdot k+1}$ of period T_{max} and load C. With this setup, $C_{min} = C_{max} = C$ and the allocation of $n \cdot k$ first runnable results in a perfect load balancing of constant load $\rho_k \cdot T_{tic} - C \cdot T_{tic}/T_{max}$. As a result, allocating the last runnables induces the load $\rho_k \cdot T_{tic} - C_{min} \cdot T_{tic}/T_{max} + C_{max}$ in some slots.

3) Lower bound on the allocatable load: We introduce here a lower bound on the capacity of the core algorithm 2 guarantees to be able to use given a harmonic runnable set. This is referred to as the harmonic schedulability bound.

Theorem 4. *The harmonic schedulability bound is equal to* $(1 - \frac{C_{max}}{T_{tic}})$ *of the capacity of each core.*

Proof: Reasoning as done for Corollary 1, the worst case peak load is given by allocating a runnable $\mathcal{R} = (C_{max}, T_{max} = T_{tic})$ in a slot allocation with a perfect balance load. In the worst case, the system is still schedulable when this average slot load is equal to $T_{tic} - C_{max}$. In other words, when the system becomes no longer schedulable, every slot has an allocated load greater or equal to $T_{tic} - C_{max}$. As a consequence, at least $(1 - \frac{C_{max}}{T_{tic}})$ of the capacity of the considered core can been used by our algorithm. ■

For example, with $T_{tic} = 5ms$ and $C_{max} = 300\mu s$, at least 94% of the CPU is guaranteed to be usable. In practice, when C_{max} is small, this bound is very useful. Considering the problem of scheduling a given harmonic runnable set on a multicore ECU with an infinite number of cores using as few cores as possible, the following corollary gives a bound of the maximum number required by this algorithm.

Corollary 2. *Defining* $\mathrm{P} = \sum_i \frac{C_i}{T_i}$ *the total load to allocate for a runnable set with harmonic periods and* m_{min} *the number of core required by the "Least-loaded" algorithm to schedule it, theorem 2 gives*

$$m_{min} \leq \left\lceil \frac{\mathrm{P}}{1 - C_{max}/T_{tic}} \right\rceil \tag{7}$$

4) Dealing with non harmonic runnable sets: Usually, in practice, runnable sets do not have strictly harmonic periods. As a consequence, lemma 1 and lemma 2 do not hold anymore and equations 1 and 5 cannot be applied to provide bounds. In particular, placing a runnable in the least loaded slot of the dispatch table could induce peaks because of the runnable periodicity. Take the following runnable set for instance: $\mathcal{R}_1(10, 2)$, $\mathcal{R}_2(20, 3)$, $\mathcal{R}_3(20, 1)$, $\mathcal{R}_4(50, 2)$ with $T_{tic} = 5$ and $T_{cycle} = 100$. Figure 3 shows the dispatch table before the allocation of $\mathcal{R}_4$. The resulting distribution of the load is shown in Table II.

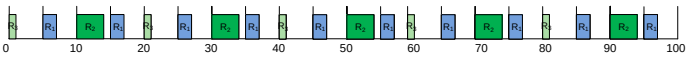

Figure 3. Dispatch table before the insertion of $\mathcal{R}_4$.

Slot	1	2	3	4	5	6	7	8	9	10	11	12	...
Load	1	2	4	2	1	2	4	2	1	2	4	2	...

Table II
LOAD REPARTITION CORRESPONDING TO THE DISPATCH TABLE IN FIGURE 3.

At that point, choosing one of the least loaded slots in the dispatch table with make the schedule fail because $\mathcal{R}_4$ will also have to be allocated in a slot with the highest load because of its periodicity. For example, if the first instance of $\mathcal{R}_4$ is allocated in slot 1, the next instance will be placed in slot 11 and make the system unschedulable. However, allocating $\mathcal{R}_4$ in any even slot is safe.

In order to deal with non-harmonic runnable sets, we need to go through a larger window of slots for the choice of the offsets. In the following, variable T_{window} is equal to the *lcm* of the periods of the runnables already scheduled at the current state of the algorithm. Instead of looking for the least loaded slot in the first T_i/T_{tic} slots, we try to create the lowest peak over T_{window}, knowing that the schedule repeats in cycle afterwards. This algorithm is further referred to as "Lowest peak" or LP for short.

Algorithm 3 "Lowest peak" heuristic.

input: runnable set $\{\mathcal{R}_i\}$, T_{tic}, T_{cycle}
(1) Sort runnables $\mathcal{R}_i$ such that $T_{tic} \leq T_1 \leq \ldots \leq T_n \leq T_{cycle}$.
(2) $T_{window} = T_{tic}$.
(3) For $i = 1 \ldots n$
  (a) $T_{window} = lcm(T_{window}, T_i)$,
  (b) In the first $\frac{T_i}{T_{tic}}$ slots, look for the slot such that the highest load in the slots where $\mathcal{R}_i$ is periodically allocated in the $\frac{T_{window}}{T_{tic}}$ first slots is the lowest,
  (c) Allocate $\mathcal{R}_i$ in every $\frac{T_i}{T_{tic}}$ slot starting from this slot.

Step (1) of algorithm 3 runs in $\mathcal{O}(n \cdot \log n)$. Step (3a) runs in $\mathcal{O}(\log T_{cycle})$. Step (3b) and (3c) respectively run in $\mathcal{O}(n\frac{T_{window}}{T_{tic}}) \leq \mathcal{O}(n\frac{T_{cycle}}{T_{tic}})$ and $\mathcal{O}(n\frac{T_{cycle}}{T_i}) \leq \mathcal{O}(n\frac{T_{cycle}}{T_{tic}})$. As a result, the whole algorithm runs in $\mathcal{O}(n(\log n + 2\frac{T_{cycle}}{T_i} + \log T_{cycle}))$.

5) Improvement: placing outliers first: The algorithms described in sections III-A and III-B construct the sequencing of runnables with arbitrary periods and possibly with locality and inter-runnable constraints. Experiments show that these algorithms sometimes do not always perform well with runnable sets where a few runnables with a low frequency have a very large WCET compared to the other runnables.

In practice, runnables with a large WCET tend to have a large period. As a result, runnables with large WCET are usually processed late in the runnable allocation process which explains the load peaks. In order to reduce those peaks, the scheduling algorithm is improved by processing some runnables with a large WCET first[1].

We define the WCET threshold $C_{critic} = \mu + k \cdot \sigma$ with μ and σ denoting respectively the average and the standard deviation of the distribution of $\{C_i\}$ and k an integer value. The runnables with C_i larger than C_{critic} are allocated first. Then, the rest of the runnables are processed as done in algorithm 3. This new version of the load-balancing algorithm is referred to as Lowest Peak k-sigma, or LP$_{k\sigma}$ for short.

IV. Experimentations

Here we evaluate the ability of the algorithms to uniformize the CPU load over time and to keep on providing feasible solutions at very high load level. For this purpose, the algorithms LL, LP, and LP$_{k\sigma}$, described respectively in §III-B1, §III-B4 and §III-B5, have been implemented in the freely available software RTaW-ECU [12].

A. Balancing performance

We applied the algorithms to sets of runnables that are realistic in the sense that their characteristics (*i.e.*, period, WCET) are drawn at random from distributions derived from an existing PSA body gateway ECU with about 200 runnables whose periods are close to harmonic (only about 5% of the runnables have non-harmonic periods).

In the experiments of the paper, the duration of the slot, T_{tic}, is set to $5ms$, the largest WCET is 30x the smallest and the periods are non-harmonic chosen in $\{10, 20, 25, 40, 50, 100, 200, 250, 500, 1000ms\}$. Random dependencies between runnables are also introduced through the following parameters:

- Interdependency ratio, that is the percentage of runnables that are dependent and thus must be executed on the same core, chosen equal to 30%.
- Maximum size of the clusters of dependent runnables is equal to 4.
- Core locality constraint ratio: percentage of runnables that are pre-allocated to a given core, chosen equal to 30%.

The following additional parameters are used for this experimentation $C_{max} = 300\mu s$, $T_{tic} = 5ms$, $T_{cycle} = 1s$ and there are over 4000 runnables to schedule on 3 cores inducing an average load of 95% of the capacity of the ECU. The parameters have been set so that the problem is challenging as we are above the harmonic schedulability bound which would be here 94%.

The distribution of the load obtained with the Lowest Peak and Lowest Peak 1-sigma (LP$_{1\sigma}$) algorithms are shown in figure 4. In the two sub-graphics, the X axis is the time-line and the Y axis is the load of the slots in percentage. The left-hand graphic shows that LP fails to provide a feasible schedule since the load is slightly above 100% in some slots (*e.g.*, slots 29, 30 of core 1). The few load peaks (corresponding to around 5 % of the core capacity) are due to runnables having a large WCET with a large period, and thus having been placed late in the allocation process. On the other hand, LP$_{1\sigma}$ is able to successfully schedule the runnable sets on the three cores with a well balanced distribution of load. In addition, around 5% of the capacity of each core remains available most of the time, which means that some more runnables can be added in future evolutions of the ECU.

B. Schedulability performances and robustness on automotive ECUs

The goal is to assess the extent to which the schedulability bound, even if it has been derived in the harmonic case, can provide guidelines for the non-harmonic case. Precisely, we measure the success rate of the algorithms in the non-harmonic case at load levels such that feasibility would be ensured in the harmonic case. In the existing body gateway ECU, the set of task periods is close to be harmonic since withdrawing only a few runnables ensures the harmonic property. To test the algorithms in a more difficult context, we build a "hard" non-harmonic case with more departure from the harmonic property. Precisely the periods are now chosen in the set $\{10, 20, 25, 40, 50, 100, 125, 200, 125, 500, 1000ms\}$.

max WCET (μs)	150	300	900
Schedulability bound in the harmonic case	97%	94%	82%
Success % of LL in the "hard" non-harmonic case	96%	96%	92%
Success % of LP in the "hard" non-harmonic case	100%	100%	100%

Table III
Performances of the scheduling algorithms in the non-harmonic case when the load is close to the harmonic schedulability bound. Statistics collected on 1000 random configurations for each max. WCET value. The schedulability bound is derived from theorem 2.

As can be seen in Table III, when the load is close to the harmonic schedulability bound the algorithms remain

[1]Allocating the runnables by decreasing order of WCET proves not to be an efficient approach in our experiments.

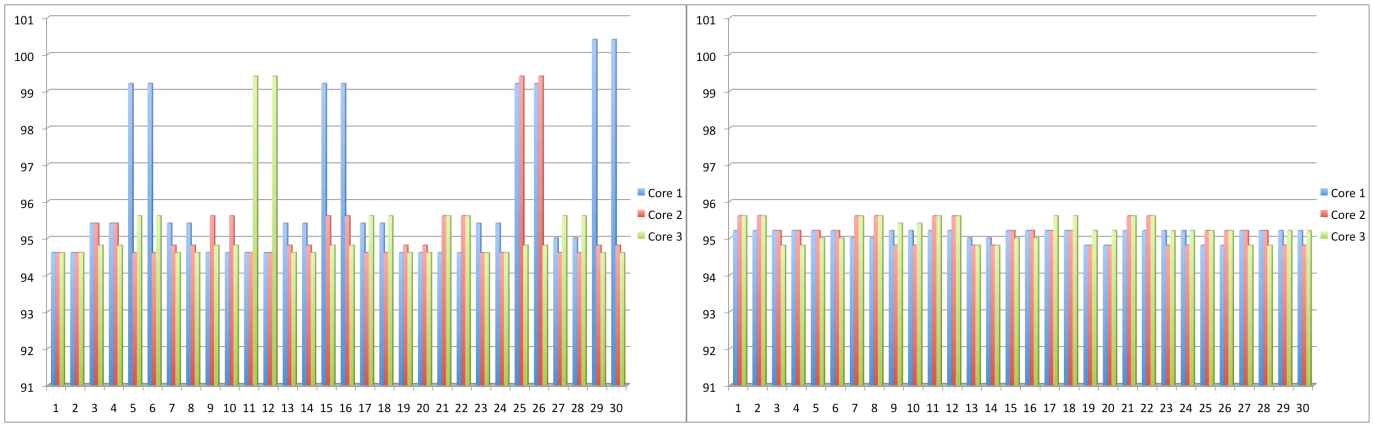

Figure 4. Distribution of the load percentage over time. The left-hand graphic shows the result of the "Lowest Peak" algorithm while right-hand graphic shows the result of the "Lowest Peak 1-sigma" algorithm. Only the first 30 slots are shown in the graphics but they are representative of the whole 200 slots of the sequencer tasks. The algorithms of this study have been implemented as plugins of RealTime-at-Work's RTaW-ECU [12] software.

efficient, in particular the LP which was able to successfully schedule the 1000 random configurations of the test. This suggests to us that the harmonic schedulability bound is a good dimensioning criterion also in the non-harmonic case.

Table IV presents the results obtained at higher loads, *i.e.* above the harmonic schedulability bound. Precisely sets of runnables with max. WCET equal to $300\mu s$ and $900\mu s$ and CPU loads equal to 95% and 97% are scheduled with LL, LP and $\text{LP}_{1\sigma}$.

CPU load	95%	97%	95%	97%
Schedulability bound in the harmonic case	94% WCET=300μs		82% WCET=900μs	
Success % of LL	64%	18%	12%	1%
Success % of LP	94%	94%	30%	5%
Success % of $\text{LP}_{1\sigma}$	100%	100%	97%	76%

Table IV
PERFORMANCES OF THE SCHEDULING ALGORITHMS IN THE NON-HARMONIC CASE WHEN THE LOAD IS GREATER THAN THE HARMONIC SCHEDULABILITY BOUND. STATISTICS COLLECTED ON 1000 RANDOM CONFIGURATIONS FOR EACH MAX. WCET VALUE.

As it was expected, the lower the schedulability bound, the harder it is to schedule the runnables (compare for instance the 97% columns). The second lesson is that $\text{LP}_{1\sigma}$ clearly outperforms all the other contenders especially when the WCETs are large.

V. COMBINING RUNNABLE SEQUENCING WITH TASK SCHEDULING

In this section, we address the global scheduling at the OS-level in an approach combining sequencer tasks with other OS tasks. We assume that the different tasks are scheduled by a fixed priority preemptive scheduler as it is the case in AUTOSAR ECUs. In the next subsections, two cases are distinguished: synchronized tasks and non synchronized tasks. In the first case, the initial offsets between tasks are known and the tasks are scheduled using a single clock. In the second case, the different sequencer tasks may be driven by different clocks. This latter case arises, for instance, in engine controllers in which some runnables are driven by the microcontroller clock while others are driven on the basis of the engine RPM which varies over time. For each case, we discuss how to build the slot allocation of the sequencer tasks so as to maximize the schedulability of the task set, before addressing how to verify the schedulability of the resulting solution.

A. Problem description

In this section, the focus is set at the core level and the assumption that only one sequencer task is scheduled on each core is relaxed. Now, it is assumed that one can have several sequencer tasks on a same core. This case arises when memory protection across runnables is needed. Indeed, memory protection, such as provided by AUTOSAR OS, cannot be ensured at the runnable level but at the task (or ISR and OS-application) level. Then, we are given now an extra set of periodic tasks that needs to be scheduled on the same core and described as usual by $\mathcal{T}_i = (C_i, T_i, D_i, J_i)$. Quantities C_i, T_i, D_i and J_i correspond respectively to the Worst-Case Execution Time (WCET), the period, the relative deadline and the release jitter of the task $\mathcal{T}_i$.

In the context of this problem, the runnables have already been allocated onto a set of sequencer tasks $\mathcal{S}_j$, described such as in II-B2, according to their source and

memory protection requirement but the slot allocation remains to be done. The schedulability condition 3 of II-B4 is also relaxed: it is too stringent for the case where one has multiple sequencer tasks per core since higher priority tasks may preempt the sequencer tasks during a whole slot duration. As of now, we only require runnables to have a single instance active at each point in time, which corresponds to the classical case where deadlines are equal to periods. Finally, though it can be handled in our framework, it is assumed that the release jitter of a sequencer task is negligible because its activations are usually driven by a high priority OS service. Given these hypotheses, the problem is to find a strategy to build the slot allocation of all the sequencer tasks in order to increase the schedulability of the system consisting in a task set and some runnable sets that are scheduled on the same core.

B. Synchronized tasks

Synchronized means here that the initial offsets of the different tasks are known and that all the tasks are scheduled using the same time basis. As a consequence, the tasks' phasings are known and can be used when building the slot allocation of the sequencer tasks. As in the case where one has a single sequencer, this latter problem cannot be solved optimally (see §III-B1) and a heuristic approach is required.

We propose a similar approach as done for the LP algorithm but here one has to take into account the interferences of higher priority tasks. When placing a runnable, we look for the slot that minimizes the highest response time of the runnable, throughout all the slots where it is allocated (until the schedule repeats itself). For that purpose, each slot is transformed in a task that captures the execution requirements of all the runnables allocated in the slot.

The response time needs to be calculated for each slot in the hyperperiod of the task set (possibly longer than T_{cycle} of the sequencer task). This can be done using the response time analysis for static priority tasks with offsets and jitters introduced in [15] and applying it for each slot to whom the runnable belongs. Integrating sequencer tasks into Redell's approach is done by transforming the slot allocations into a task set. Each of the slots is translated into a single task with the proper offset and a period equal to T_{cycle}. This does not further complexify Redell's analysis since the number of task instances remains the same.

Though this approach is extremely computation intensive[2], usual automotive applications lead to small hyperperiods since the task sets have almost harmonic periods. Furthermore the schedulability analysis is conducted at the same time as the sequencer tasks are built.

C. Non synchronized tasks

The previous approach cannot be applied to non synchronized tasks since their offset and time basis are not known. If the tasks are scheduled on different and varying time bases (*e.g*, CPU clock and engine RPM), whatever the way a sequencer task is constructed, every higher priority task can interfere with any of the sequencer task's slot, as all offset configurations between them are possible at run-time. This means that, on the contrary to the synchronized case of §V-B, we cannot take advantage of the characteristics of the higher priority tasks when building the slot allocation of a sequencer task. The maximum robustness against all possible asynchronisms between sequencer tasks is achieved by balancing the load of each of the tasks individually, as done in the basic use-case of the algorithms of §III-B.

Because of the possibly varying time bases, the schedulability of the slot allocation cannot be checked as in the previous cases. However, if it is possible to bound the clock speed of each sequencer task, the multi-frame task model (*i.e.*, periodic activations but varying execution times between instances [16]) can be used to check the feasibility of the schedule. The transformation of a sequencer task into a multi-frame task is loseless : the slots of a sequencer task become the task instances with their execution times depending on the runnables actually scheduled in each slot. Then, assuming the maximum clock speeds, which lead to the worst-case workload arrival, a multiframe schedulability test can be applied, for instance [17], [18] or [19] which integrates release jitters in the schedulability analysis.

VI. Conclusion

Multi-source software and multicore ECUs will drastically change the electrical/electronic architectures and should enable more cost-effective and more flexible automotive embedded systems. In our view, the OS protection mechanisms specified by AUTOSAR provide a sound basis to develop appropriate safety mechanisms and policies, despite the growing complexity and criticality of software functions. However, today's design methodologies need to be adapted to this new context and there is a wide range of technical problems to

[2]The problem of computing response times with offsets in the synchronized case is conjectured to be NP-hard, though, to the best of our knowledge, there is no proof in the literature.

be solved. Among these issues are the design of the software architectures and the scheduling of the software components, which have been considered in this paper.

The set of runnable sequencing algorithms proposed in this paper aims at uniformizing the load over time, and thus increase the maximum workload schedulable on the CPU. The algorithms also provide guaranteed performance levels in some specific contexts. Experimentations on realistic case-studies have confirmed the algorithms to be versatile and efficient in terms of CPU usage optimization.

We have presented practical solutions to schedule activities according to both the static cyclic and priority-driven paradigms, as it is becoming a need in automotive multicore ECUs and other complex embedded systems with dependability requirements such as are required in the aerospace domain. Our ongoing work is to extend this study to handle the constraints originating from the communication between runnables located on distinct ECUs. This first requires the precise modeling of the data exchanges and capture their timing constraints, for instance using the TIMMO-2-USE methodology [20], then to extend the scheduling algorithms.

REFERENCES

[1] A. Emadi, Y. Lee, and K. Rajashekara, "Power electronics and motor drives in electric, hybrid electric, and plug-in hybrid electric vehicles," *IEEE Trans. on Ind. Electronics*, vol. 55, no. 6, pp. 2237–2245, june 2008.

[2] F. Mapelli, D. Tarsitano, and M. Mauri, "Plug-in hybrid electric vehicle: Modeling, prototype realization, and inverter losses reduction analysis," *IEEE Trans.on Industrial Electronics*, vol. 57, no. 2, pp. 598 –607, feb 2010.

[3] D.-J. Kim, K.-H. Park, and Z. Bien, "Hierarchical longitudinal controller for rear-end collision avoidance," *IEEE Trans. on Industrial Electronics*, vol. 54, no. 2, pp. 805 –817, april 2007.

[4] T. Bucher, C. Curio, J. Edelbrunner, C. Igel, D. Kastrup, I. Leefken, G. Lorenz, A. Steinhage, and W. von Seelen, "Image processing and behavior planning for intelligent vehicles," *IEEE Trans. on Industrial Electronics*, vol. 50, no. 1, pp. 62 – 75, feb 2003.

[5] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion, "Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling," in *IEEE International Symposium on Industrial Electronics (ISIE)*, 2010.

[6] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Trans. on Computers*, vol. 44, no. 12, pp. 1429–1442, Dec. 1995.

[7] Y. Oh and S. Son, "Fixed-priority scheduling of periodic tasks on multiprocessor systems," Department of Computer Science, University of Virginia, Tech. Rep. CS-95-16, 1995.

[8] S. Lauzac, R. Melhem, and D. Mossé, "An improved rate-monotonic admission control and its applications," *IEEE Trans. on Computers*, vol. 52, no. 3, pp. 337–350, 2003.

[9] A. Karrenbauer and T. Rothvoss, "An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling," in *17th Annual European Symposium on Algorithms (ESA)*, 2009.

[10] J. Goossens, "Scheduling of offset free systems," *Real-Time Systems*, vol. 24, no. 2, pp. 239–258, March 2003.

[11] M. Grenier, L. Havet, and N. Navet, "Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost," in *European Congress of Embedded Real-Time Software (ERTS)*, 2008.

[12] RealTime-at-Work, "RTaW-ECU: Static cyclic scheduling of tasks," Available for download at http://www.realtimeatwork.com, 2011.

[13] AUTOSAR Consortium, "Specification of multi-core OS architecture v1.0," AUTOSAR Release 4.0, 2009.

[14] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions*. Dover Publications (ISBN 0-486-61272-4), 1970.

[15] O. Redell and M. Törngren, "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2002.

[16] A. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Trans. on Software Engineering*, vol. 23, pp. 635–645, 1996.

[17] S. Baruah, D. Chen, and A. Mok, "Static-priority scheduling of multiframe tasks," in *11th Euromicro Conference on Real-Time Systems*, 1999, pp. 38–45.

[18] A. Zuhily and A. Burns, "Exact response time scheduling analysis of accumulatively monotonic multiframe real time tasks," *Real-Time Systems*, vol. 43, no. 2, 2009.

[19] ——, "Exact scheduling analysis of non-accumulatively monotonic multiframe tasks," in *16th International Conference on Real-Time and Networked Systems (RTNS)*, 2008.

[20] ITEA2 TIMMO-2-USE Consortium, "Mastering timing tools, algorithms, and languages," available at http://www.timmo-2-use.org/overview.htm, 2011.

Aurélien Monot graduated from the engineering school Ecole des Mines de Nancy in 2008, majoring in Computer Science. After spending 6 months at the IBM research lab in Zürich, he started an industrial Ph.D at PSA Peugeot-Citroën co-supervised by the LORIA in Nancy on "end-to-end timing constraints in the AutoSar context". He is now INRIA research engineer in the ITEA2 Timmo-2-Use project. His main topics of interest are real-time embedded systems and model-based design approaches.graduated from the engineering school Ecole des Mines de Nancy in 2008, majoring in Computer Science. After spending 6 months at the IBM research lab in Zürich, he started an industrial Ph.D at PSA Peugeot-Citroën and co-supervised by the LORIA in Nancy on the topic of "End-to-end real Time constraints in the AutoSar context". He is now also involved in the ITEA project Timmo-2-Use as a member of INRIA. His main topic of interests are real-time embedded systems and model-based design approaches.

Nicolas Navet is a researcher at INRIA (LORIA Lab, France) since 2000 and head of the INRIA TRIO project (Real-Time and Interoperability). His research interests include real-time and embedded systems, communication protocols, fault tolerance and dependability assessment. For the last 17 years, he has worked on numerous projects with OEMs and suppliers in the automotive and avionics domains. He is the founder of RealTime-at-Work, a company that helps system designers build truly safe and optimized critical systems. He has a B.S. in Computer Science from the University of Berlin (1993) and a PhD in Computer Science from the Institut National Polytechnique de Lorraine (1999).

Bernard Bavoux is a 1984 Supélec graduate engineer. He began his career developing embedded microcontroller and printed circuits boards for spatial projects at Thales (formerly Thomson) during 5 years. Then he moved to the aeronautic domain at TEAM, a world leader equipment supplier. There during 9 years, he was in charge of the electronic design office, including research and development, for analog and digital audio intercommunication systems. Since 14 years he has been working in the automotive industry: During 5 years at Valeo, he managed an Electric and Electronic Architecture innovation team and an embedded Electronic Control Units research team. Currently at PSA Peugeot Citroën, he is leading an advanced research team in the fields of Electricity, Electronics and Opto-electronics in connection with the best research institutes in the world. Since 10 years, he is recognized as a senior expert in this field.

Françoise SIMONOT-LION is Professor of Computer Science at University of Lorraine (France). Between 1997 and 2010, she was the scientific leader of the Real Time and InterOperability (TRIO) LORIA/INRIA research team and since 2010, head of the LORIA laboratory. Her research topics are modeling and analysis of real time distributed systems. She is involved in several research collaborations with automotive industry and was co-chair of the subcommittee "Automotive Electronic and Embedded Systems" of the IEEE Industrial Electronic Society (IES) - TCFA. She coauthored more than 100 technical papers in the area of real-time systems modeling and analysis and is associate editor of the IEEE Transactions on Industrial Informatics.