A Novel Heuristic Algorithm for Mapping AUTOSAR Runnables to Tasks

Fouad Khenfri[1,2], Khaled Chaaban[1] and Maryline Chetto[2]

[1] *S2ET, ESTACA, Rue Georges Charpak, 53061 Laval, France*

[2] *IRCCyN, University of Nantes, 1 Rue de la Noe, 44321 Nantes, France*

Keywords: Optimization, Real-time scheduling, AUTOSAR.

Abstract: This paper describes a novel algorithm that permits to automate the process to map runnables to tasks in any AUTOSAR architecture. This enables to boost system performance by reducing the number of tasks to be implemented and while preserving system schedulability. Our algorithm uses some properties related to the activation offset for mapping runnables with distinct periods to the same task. We consider periodic, independent and fixed-priority tasks running on a single processor. The results of an experimental study are reported. First, they show that our algorithm reduces significantly the number of tasks with distinct periods while preserving system schedulability. And second, the system schedulability bound is increased by 34% compared to the typical periodic solution and the average response time of tasks is reduced by 30% related to ohers solutions.

1 INTRODUCTION

AUTOSAR (AUTomotive Open System ARchitecture) development partnership has been created by automotive manufacturers, suppliers and tool developers in order to establish an open industry standard for automotive E/E (Electrical/Electronic) architectures (Partnership, 2011). AUTOSAR provides a set of concepts and defines a methodology for the automotive software development process. Some key features of this standard are modularity and configurability of automotive architectures. AUTOSAR permits functional reuse of software modules from different suppliers and guarantees interoperability of these modules through standardized interfaces.

In AUTOSAR, runnables represent the internal behavior of each SoftWare Component (SWC) and are the smallest pieces of code to be scheduled. Each SWC contains at least one runnable. All runnables are triggered in response to an event such as a timing event (for periodic runnables), data receiving, or operation invoking of runnables. If no event is specified for the runnable, it will never be activated. Furthermore, runnables can be executed only if they are mapped to tasks. The process of mapping runnable to task is illustrated in Figure 1.

Developing an automotive application which is compliant to the AUTOSAR standard needs the initialization of many parameters. They are mainly related to SWCs, allocation of SWCs to the hardware architecture, and finally configuration of each Electronic Control Unit (ECU). In the ECU configuration phase, mapping runnables to OS (Operating System) tasks is crucial since it may impact system performance and its real-time behavior significantly. Several design decisions have to be considered including:

- How many tasks are needed to implement the entire set of runnables?
- How to specify priorities of tasks?
- How to define the execution order and activation offset of runnables which are mapped to the same task?

There are mainly three ways to address this problem as shown in Figure 2.

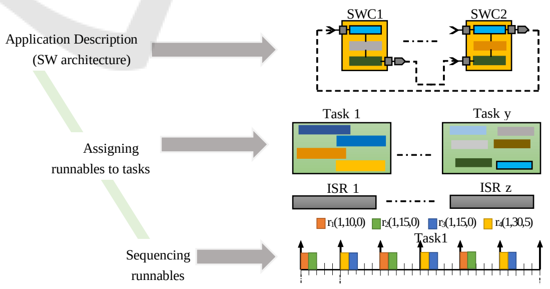

Figure 1: Runnable mapping process in AUTOSAR.

Khenfri F., Chaaban K. and Chetto M..
A Novel Heuristic Algorithm for Mapping AUTOSAR Runnables to Tasks.
DOI: 10.5220/0005234202390246
In *Proceedings of the 5th International Conference on Pervasive and Embedded Computing and Communication Systems* (PECCS-2015), pages 239-246
ISBN: 978-989-758-084-0
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

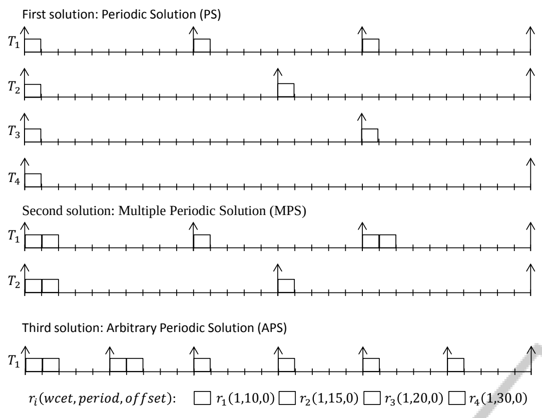
First solution: Periodic Solution (PS)
T_1
T_2
T_3
T_4
Second solution: Multiple Periodic Solution (MPS)
T_1
T_2
Third solution: Arbitrary Periodic Solution (APS)
T_1
$r_i(wcet, period, offset)$: □ $r_1(1,10,0)$ □ $r_2(1,15,0)$ □ $r_3(1,20,0)$ □ $r_4(1,30,0)$

Figure 2: Solutions to map runnables to different tasks.

The first and simplest solution is to map runnables with same period to one task. In this Periodic Solution (PS), the worst-case execution time (WCET) of the task is the sum of WCET of all runnables mapped to it. Hence, the number of tasks is equal to the number of the different periods assuming that a task WCET is less or equal than its period. However, tasks priorities are assigned according to the rate monotonic priority assignment (RMPA) or the deadline monotonic priority assignment (DMPA) in order to guarantee system schedulability.

Another solution consists in mapping runnables with multiple periods to the same task. In this Multiple Periodic Solution (MPS), the period of the task is the shortest period among different runnables mapped to it. Its maximum WCET is the sum of the WCET of all these runnables. The number of tasks in this solution is less than the first one when considering the same assumptions.

However, in order to obtain a minimal number of tasks, the best solution is to map the runnables with different periods to the same task using their activation offset. In this Arbitrary Periodic solution (APS), the period of a task is the greatest common divisor (GCD) of all runnables periods mapped to it. Task WCET in this case must be less than the task period. The advantages of this solution are multiple, namely the minimization of time overhead due to task context switch and the minimization of stack memory usage by minimizing the number of tasks.

For MPS and APS, system schedulability may be guaranteed using RMPA or DMPA by assuming that the task deadline is less than its period. Otherwise, if the deadline is arbitrary (i.e. the deadline may be higher than the period), the Audsley's algorithm (Audsley, 1991) may be used to assign tasks priorities. Nevertheless, *the issue here is how to determine the set of runnables that can be mapped to the same task in order to guarantee system schedulability.*

Currently and in industrial practices, system designers use the first two solutions (PS and MPS) and their best knowledge of the system in order to configure empirically ECU software. Nevertheless, this may reveal to be an error-prone and time-consuming process. Thus, an optimization approach appears to be necessary to realize this configuration phase.

The aim of the work presented in this paper is to propose an efficient algorithm, based on APS, for mapping runnables to tasks while preserving system schedulability. This algorithm allows determining simultaneously, the number of OS tasks needed to schedule the set of runnables, the priorities of tasks, and the sequencing of the different runnables inside tasks.

The paper is organized as follows: section 2 presents relevant related works. Section 3 describes the system model. Section 4 details our proposed solution to solve the mapping problem. Section 5 reports and discusses experimental results before we conclude the paper in section 6.

2 RELATED WORKS

In the context of AUTOSAR, there are many works that address the problem of mapping runnables to tasks (Rongshen et al.,)(Haibo and Di Natale,)(Zeng et al., 2012)(Monot et al., 2012)(Navet et al., 2010). In (Rongshen et al.,), the authors discuss several rules for mapping runnables to tasks based on intra-ECU communication (communication between runnables located on the same ECU), but there is no specified optimization method. Further, this work does not consider system schedulability. In (Haibo and Di Natale,), a MILP (Mixed Integer Linear Programming) method is applied to solve an optimization problem. This method chooses between protection mechanisms to ensure data consistency. It considers that runnables are already mapped to tasks and task priority assignment is given.

In (Zeng et al., 2012), a simulated annealing method is applied to find the optimal mapping and tasks priorities assignment using MPS so as to optimize stack memory usage. After each transition of the optimization loop, execution order and preemption threshold of runnables are computed using a heuristic method. Authors of (Monot et al., 2012; Navet et al., 2010) developed two heuristics for multi-core architectures. The first one distributes runnables to different cores in order to uniformize core loads. The second one maps runnables from the same core to one sequencer task using the activation offset of runnables. This heuristic assumes that the WCET of runnales is much smaller than their periods since it considers only one sequencer task per core. In (Ming and Zonghua,), a genetic algorithm is used to select protection mechanisms in order to guarantee data consistency of each ECU with the objective of minimizing the memory size. The authors use PS to map runnables to tasks. In (Wozniak et al., 2013), the authors propose a method using genetic algorithm for the synthesis of AUTOSAR architecture (to solve the SWCs to ECUs mapping problem, runnables to tasks mapping problem and data elements to bus frame mapping problem). In their work, MPS is used to map runnables to tasks.

To the best of our knowledge, there is no work uses the APS for mapping runnables to many tasks. By the following, we detail the system model considered in this study and the proposed optimization approach.

3 SYSTEM MODEL

This section describes the system model for both runnables and tasks. AUTOSAR OS is based on the industry standard OSEK OS. It is responsible for executing real-time tasks and ISRs (Interrupt Service Routines). It provides a fixed-priority scheduling policy. Each task is assigned a static priority and higher priorities are given to ISRs. During execution time, the scheduler decides which ready task is to be executed next, based on its priority. However, AUTOSAR does not specify the priority assignment scheme. Rate Monotonic or Deadline Monotonic schemes can be used within its context. The configuration of AUTOSAR OS is done statically. All the objects of the OS and their parameters have to be defined before the deployment of the software. Moreover, let's recall that two types of tasks/runnables exist in AUTOSAR: time-triggered and event-triggered. In this work we assume that all tasks/runnables are time-triggered and periodic. Dealing with non-periodic tasks/runnables may be more difficult because of the non-deterministic release times. Hence, we start by developing a solution for periodic tasks/runnables which can be later extended to cover non-periodic ones.

3.1 Runnables

Any runnable, say r_i, is the basic element to be executed in the context of any task, say τ_j, with priority P_{τ_j}. r_i is characterized by four timing parameters O_i, C_i, T_i, D_i and position parameter k_i:

- O_i, activation offset, i.e. triggering instant of the execution request.
- C_i, worst-case execution time (WCET), with $0 \leq C_i \leq D_i \leq T_i$.
- T_i, period.
- D_i, relative deadline, i.e. the maximum acceptable delay for its processing.
- k_i, execution order in task τ_j.

The two parameters O_i and k_i can be tuned to satisfy system schedulability.

3.2 Tasks

Let us denote by $\Delta = \{r_1, r_2, ..., r_n\}$, the set of runnables mapped to the same task τ_j. An execution order i (where 1 is the first execution order and n is the last) is assigned to each runnable r_i.

In order to map runnables with distinct periods to the same task, we need to determine the activation offset of each runnable. This task is called dispatcher (or sequencer) task which stores the runnables offsets in a scheduling table and releases them at the right points in time (Monot et al., 2012). The period of the dispatcher task (micro-cycle) T_{tic_j} is defined by the greatest common divisor (GCD) of all runnables periods mapped to it. Moreover, the dispatcher task executes these runnables in a cyclic manner (major-cycle) T_{cycle_j}, which is the least common multiple (LCM) of all runnables mapped to it (see Fig. 1). The WCET of dispatcher tasks $C_{\tau_j} = \{C_0, C_1, ..., C_s, ..., C_{N-1}\}$, is a vector, where $N = T_{cycle_j}/T_{tic_j}$ is the number of slots. The WCET C_s of the s^{th} slot is given by:

$$C_s = \sum_{i=1}^{n} x_i(s) * C_i \qquad (1)$$

where the function x_i(s) indicates the presence or not of the i^{th} runnable in the s^{th} slot. This function is given by:

$$x_i(s) = \begin{cases} 1 & if\ a_i(s) = s \\ 0 & if\ a_i(s) \neq s \end{cases} \qquad (2)$$

with,

$$a_i(s) = \left\lfloor \frac{s}{\gamma_i} \right\rfloor + \delta_i \qquad (3)$$

where $a_i(s)$ is the release time position of the i^{th} runnable in the s^{th} slot (see example 1). $\gamma_i = T_i/T_{tic_j}$ and $\delta_i = O_i/T_{tic_j}$ represent respectively the repetition factor and the first position of the i^{th} runnable in the slots .

Proposition 1. *The relative deadline D_{τ_j} of dispatcher task τ_j guarantees the timing requirements of all runnables mapped to it if, and only if, the worst case response time of τ_j is less than or equal to D_{τ_j} given by:*

$$D_{\tau_j} = \min_{s=0 \dots N-1} \{D_s\} \tag{4}$$

with,

$$\max_{s=0 \dots N-1} \{C_s\} \leq D_{\tau_j} \leq T_{cycle_j} \tag{5}$$

$$D_s = \begin{cases} \min_{k \in R(s)} \{D_k + \sum_{i=k+1}^{n} x_i(s).C_i\} & if\ R(s) \neq \emptyset \\ \infty & if\ R(s) = \emptyset \end{cases} \tag{6}$$

NOTE: *R(s)* denotes the set of runnables positioned in the slot s.

Proof. Let us consider that a set of runnables $\Delta = \{r_1, r_2, \dots, r_k, \dots, r_n\}$ is mapped to the same task τ_j. Each runnable r_k is assigned an execution order k (where 1 is the first execution order and n is the last) and a deadline D_k. When all runnables are mapped in the first slot (i.e., $\forall k, x_k(s) = 1$ with s=0), the deadline of dispatcher task (4) is rewritten as follows:

$$D_{\tau_j} = \min_{k=1 \dots n-1} \left\{ D_k + \sum_{i=k+1}^{n} C_i, D_n \right\} \tag{7}$$

To verify timing constraints for all runnables, we need to find the deadline of dispatcher task which respects the following inequalities:

$$\begin{array}{rcl} D_{\tau_j} & \leq & D_n \\ D_{\tau_j} - C_n & \leq & D_{n-1} \\ & \vdots & \\ D_{\tau_j} - (C_n + \dots + C_{k+1}) & \leq & D_k \\ & \vdots & \\ D_{\tau_j} - (C_n + \dots + C_2) & \leq & D_1 \end{array} \tag{8}$$

It can be observed that equation (7) satisfies the inequality above. When all runnables are positioned in different slots, the relative deadline of dispatcher task is the minimum of deadlines among all slots. These deadlines are computed by equation (7) applied for each slot given by (6). □

Example 1: Figure 3 illustrates a dispatcher task for a set of four periodic runnables $r_i(O_i, C_i, T_i, D_i, k_i)$: $r_1(0,1,10,8,1)$, $r_2(5,1,15,10,2)$, $r_3(0,1,15,12,3)$ and $r_4(25,1,30,20,4)$. The period of the dispatcher task is GCD(10,15,15,30)=5ms, and its major cycle is LCM(10,15,15,30)=30ms. Thus, the number of slots is 6.

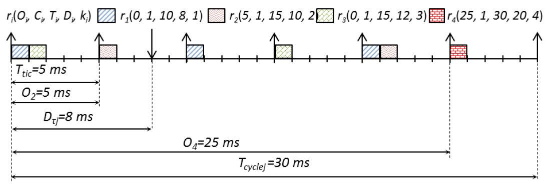

Figure 3: Illustration of activation offset for runnables with position in task.

Table 1 displays values for an example to compute the WCET C_s of equation (1) and the deadline of equation (6) of the s^{th} slot. From equation (4), the deadline of dispatcher task D_{τ_j} is 8 ms.

Table 1: Computation of WCET and deadline of s^{th} slot.

s^{th} slot	a_i(s)				$x_i(s)$				C_s	D_s
	r_1	r_2	r_3	r_4	r_1	r_2	r_3	r_4		
0	0	1	0	5	1	0	1	0	2	9
1	0	1	0	5	0	1	0	0	1	10
2	2	1	0	5	1	0	0	0	1	8
3	2	4	3	5	0	0	1	0	1	12
4	4	4	3	5	1	1	0	0	2	9
5	4	4	3	5	0	0	0	1	1	20

Note: If $T_{cycle_j} = T_{tic_j}$, there is one slot where all runnables are mapped to the same task τ_j. Thus, runnables are activated at the same instant and have the same period. In this case, the dispatcher task model comes close to the task model of Liu and Layland $\tau_j(C_{\tau_j}, T_{\tau_j}, D_{\tau_j})$ (Liu and Layland, 1973). Further, the task WCET equals to the sum of the WCET of all runnables mapped to it, its period T_{τ_j} equals to the period of its runnables, and its deadline D_{τ_j} is given by (7).

Let us note that the Liu and Layland's task model $\tau_j(C_{\tau_j}, T_{\tau_j}, D_{\tau_j})$ is a particular case of the dispatcher task model $\tau_j(C_{\tau_j}, T_{tic_j}, T_{cycle_j}, D_{\tau_j})$.

In the following section, we introduce a new heuristic that relies on the APS solution, inspired by Audsley's priority assignment approach (Audsley, 1991). A Least Loaded (LL) algorithm (Monot et al., 2012) is adapted to find the activation offset of every periodic runnable that respects its execution order.

4 APS-BASED HEURISTIC ALGORITHM

In this section, a heuristic is developed to solve the next optimization problem:

Find:	Runnables assigned to tasks, Activation offset for runnables, Execution order of runnables, Tasks priorities.
Minimize:	Number of tasks
Subject to:	Schedulability of the task set

Our heuristic is inspired by Audsley's priority assignment in order to establish a minimal set of tasks $\tau = \{\tau_1, ..., \tau_m\}$. Furthermore, each task τ_j is assigned a priority j (where 1 is the lowest priority and m is the highest). The cardinal number m represents the number of tasks. The task τ_j is built from a set of runnables Π_j which is a subset of the entire input set of runnables Δ to be mapped. The activation offset and execution order parameters are specified to assign each runnable set Π_j to task τ_j. Thus, for each priority level j (starting with the lowest priority) there are the two following steps:

1. The first step tries to find an optimal set of runnables Π_j to assign it to the same task τ_j. A scheduling feasibility test is applied to find Π_j of Δ as detailed in the next subsection (4.1).
2. The second step permits to specify activation offset and execution order of runnable set Π_j, in order to ovoid overload situation in the slots (i.e., the max of WCET of dispatcher task is greater than its period T_{tic_j} or its deadline D_{τ_j}). Otherwise, a reject set Λ_j will contain the first runnables leading to overload situations.

At priority level j+1, a task τ_{j+1} assigned to this priority level is created of Π_{j+1}, which is a subset of $(\Delta \backslash \Pi_j) \cup \Lambda_j$. Hence, the set of tasks $\Gamma = \{\tau_1, ..., \tau_m\}$ is set up until obtaining an empty set of Δ, where $\Pi_1 \cap \Pi_2, ..., \cap \Pi_m$ is empty and $\Pi_1 \cup \Pi_2, ..., \cup \Pi_m$ is equal to Δ (see Algorithm 1).

Algorithm 1: Heuristic for mapping runnables to tasks.

```
Data: runnable set $\Delta$
$\Gamma \leftarrow \emptyset$ ;
$Failed \leftarrow False$ ;
$j \leftarrow 1$ ;                          /* index of lower priority */
while $((\Delta \neq \emptyset) \&\& (Failed = False))$ do
    $\Pi_j \leftarrow Algorithm2(\Delta)$ ;
    if $Pi_j \neq \emptyset$ then
        $(\Lambda_j, \tau_j) \leftarrow Algorithm3(\Pi_j, j)$ ;
        $\Gamma \leftarrow \tau_j$ ;
        $\Delta \leftarrow (\Delta \backslash \Pi_j) \cup \Lambda_j$ ;
        $j \leftarrow j + 1$ ;
    end
    $Failed \leftarrow True$ ;    /* The system is unschedulable */
end
return $\Gamma$
```

4.1 Assignment of Runnables to Tasks

We detail here the first step of Algorithm 1 whose objective is to find, for each priority level j, an optimal set of runnables $\Pi_j \subseteq \Delta$ to be assigned to the same dispatcher task τ_j. The following proposition explains how the set of runnables Π_j is selected.

Proposition 2. *Let us assume a set of runnables $\Pi_j = \{r_1, r_2, ..., r_p\}$ of cardinality p. If each runnable $r_i \in \Pi_j$ is schedulable by assigning r_i to a lower priority level, where the rest of runnables $(\Delta - r_i)$ are assigned a higher priorities level, then all runnables of Π_j could be allocated to the same task.*

By Proposition 2, Algorithm 2 finds the set of runnables $\Pi_j \subseteq \Delta$ which can be allocated to the same task. Algorithm 2 is based on the feasibility test (9) used by the Audsley's algorithm:

$$\exists\, t_0 \leq t \leq D_i : \frac{C_i}{t} + \frac{I_i(t)}{t} \leq 1 \tag{9}$$

where,

$$I_i(t) = \sum_{j \in \Delta - r_i} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{10}$$

Algorithm 2: Assignment of runnables to tasks.

```
Data: runnable sets $\Delta$
$\Pi_j \leftarrow \emptyset$ ;
$t_0 \leftarrow \sum_{i \in \Delta} C_i$ ;
for $r_i$ in $\Delta$ do
    $t \leftarrow t_0$ ;
    $continue \leftarrow True$ ;
    while continue do
        if $\left(\frac{C_i}{t} + \frac{I_i(t)}{t} \leq 1\right)$ then
            if $(t \leq D_i)$ then
                $\Pi_j \leftarrow r_i$ ;        /* $r_i$ is schedulable */
            end
            continue $\leftarrow$ False;
        end
        t= $I_i(t) + C_i$ ;
        if $(t > D_i)$ then
            continue $\leftarrow$ False ;  /* $r_i$ is unschedulable */
        end
    end
end
return $\Pi_j$
```

4.2 Sequencing of Runnables

The second step of Algorithm 1 is described by Algorithm 3 which builds the dispatcher task τ_j using the activation offsets between the runnables of Π_j given by Algorithm 2. For each runnable $r_i \in \Pi_j$, we have $\gamma_i = T_i / T_{tic_j}$ possibilities to define the first position δ_i of runnable r_i, where $0 \leq \delta_i \leq \gamma_i - 1$. Thus, the offset of the runnable r_i is $\theta_i = \delta_i * T_{tic_j}$, since the max WCET of slots is lower than T_{tic_j}. Our algorithm adapts the Least Loaded (LL) heuristic developed by Monot *et al* in (Monot et al., 2012) so as to integrate deadlines of runnables. These ones enable us to find the deadline of the task through equation (4) with $\max_{s=0\ldots N-1}\{C_s\} \leq D_{\tau_j} \leq T_{cycle_j}$ and the execution order of runnables sorted in increasing order of laxity. Laxity $L_i = D_i - C_i$ represents the maximum time during which a task can be delayed while still completing before deadline.

Algorithm 3: Sequencing runnables inside task.

```
    Data: $Input$ : $runnable\ set\ \Pi_j \& j$
1   $\wedge_j \leftarrow \emptyset$ ;
2   $T_{tic_j} \leftarrow GCD(\Pi_j)$ ;
3   $l \leftarrow 1$ ;                        /* index of the first runnable */
4   Sort $\Pi_j$ by increased laxity order ;
5   for $i \leftarrow 1\ldots|\Pi_j|$ do
6       Look for $\delta_i$ that represents the least loaded slot in the $\gamma_i$ first
        slot ;
7       Allocate $r_i$ in every $\gamma_i$ slot, starting from this slot $\delta_i$ ;
8       Calculate the vector of WCET $C_{\tau_j}$ by the equation (1) ;
9       if $max(C_{\tau_j}) > T_{tic_j}$ or $max(C_{\tau_j}) > D_{\tau_j}$ then
10          Reject the first runnable $\wedge \leftarrow r_l$ ;
11          $l \leftarrow l+1$ ;
12          Update allocation for $r_l$ to $r_i$ in the slots using lines 7 & 8
            ;
13      end
14  end
15  Calculate the deadline $D_{\tau_j}$ by the equation (4) ;
16  return $\wedge_j \& \tau_j(C_{\tau_j}, T_{tic_j}, D_{\tau_j})$
```

5 EXPERIMENTATION

This section presents and discusses several experimental results. Our study has been conducted in order to evaluate the proposed approach and validate its efficiency. To correctly assess the different methods under different execution scenarios, a set of runnables is randomly generated according to a realistic distribution of WCETs and periods for covering typical automotive applications.

5.1 Set Up

A system is characterized by the following parameters: number of runnables n, processor utilization factor U, configuration number, and deadline intervals. In order to randomly generate the n runnables, we use the UUnifast algorithm developed in (Bini and Buttazzo, 2005). This algorithm distributes the total utilization factor U to smaller utilization factor u_i for n runnables. For each runnable r_i, we specify its WCET C_i by the following equation:

$$C_i = T_i \times u_i \tag{11}$$

where T_i is the period of the i^{th} runnable randomly chosen from a set of non-harmonic periods 1, 2, 3, 5, 8, 10, 15, 20, 25, 40, 50, 100, 150, 200, 250, 300, 500, 900, 1000 ms.

Deadline D_i of the i^{th} runnable is determined from the following equation:

$$D_i = (T_i - C_i) \times rand(a,b) + C_i \tag{12}$$

Where rand(a,b) generates values from the uniform distribution on the interval [a, b] with $0 \leq a \leq b$. In the case of a=b=1, the deadline is equal to period.

5.2 Results and Discussion

This subsection reports experimental results obtained by applying our approach using APS and compares it to both PS and MPS solutions.

Overall, 304 systems have been generated using the above method and corresponding to 16 cardinalities of period sets and 19 deadline intervals. Each generated system is composed of 100 runnables that are generated randomly with a processor utilization factor fixed to 69%.

Figure 4 shows the evolution of number of schedulable systems according to the range of runnable deadlines. Using MPS and APS, we obtained 197 schedulable systems for an overall of 304 analyzed systems. i.e., we have 64.80% of schedulable systems for both MPS and APS approaches. However, using PS approach, we obtained 30.26% of schedulable systems. Thus, our approach increases by 34.54% the system schedulability compared to PS approach. Moreover, APS outperforms both MPS and PS approaches regarding the number of tasks and the avearage response time as discussed thereafter.

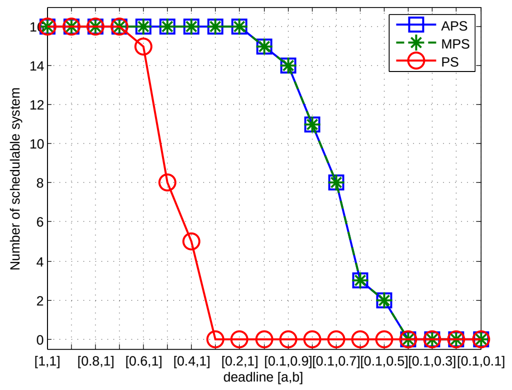

Figure 4: Number of schedulable systems with PS, MPS and APS.

From Figure 4, we could identify three different zones based on the strictness of the deadline constraint: zone A corresponds to a system with slack constraints of deadline' interval [1,1] until [0.4,1], where B corresponds to a system with tight constraints of deadline' interval [0.5,1] until [0.1,0.5], and zone C corresponds to systems with firm constraints of deadline' interval [0.1,0.4] until [0.1,0.1]. In the following, we discuss the performance results obtained by our APS approach for the three zones and we compare it to PS and MPS. To do that, we define for each period cardinality, two performance metrics:

- The maximum number of tasks among all deadlines intervals for a given zone,
- The benefit percentage of the average response time (ART) among deadlines intervals for a given zone.

Zone A: in this zone, we note that APS approach reduces by 1 to 16 the total tasks number compared to PS and by 1 to 3 compared to MPS (see Figure 5(a)). On the other side, Figure 5(b) shows the relative improvement (in benefit %) of APS approach regarding the average response time (ART). This improvement increases with period cardinality.

Zone B: in this part, the PS approach is unschedulable and thus it is excluded from comparisons. Figure 6(a) shows the evaluation of the number of tasks. We note that APS approach reduces by 3 to 9 the tasks number compared to MPS. Moreover, APS approach outperforms MPS approach regarding the improvement (in benefit %) of average response time (see Figure 6(b)).

Zone C: in this part, there is no system that can be schedulable by any approach due to the tight deadline constraints.

6 CONCLUSION

In the AUTOSAR software design methodology, mapping runnables of software components to OS tasks represents a central design decision. This may impact system performance, real-time schedulability and execution determinism of the application. Several approaches have been proposed to solve this design issue, ranging from the simplest solution that map together runnables with same periods (PS) or multiple periods (MPS) to the same task, up to a more ad-

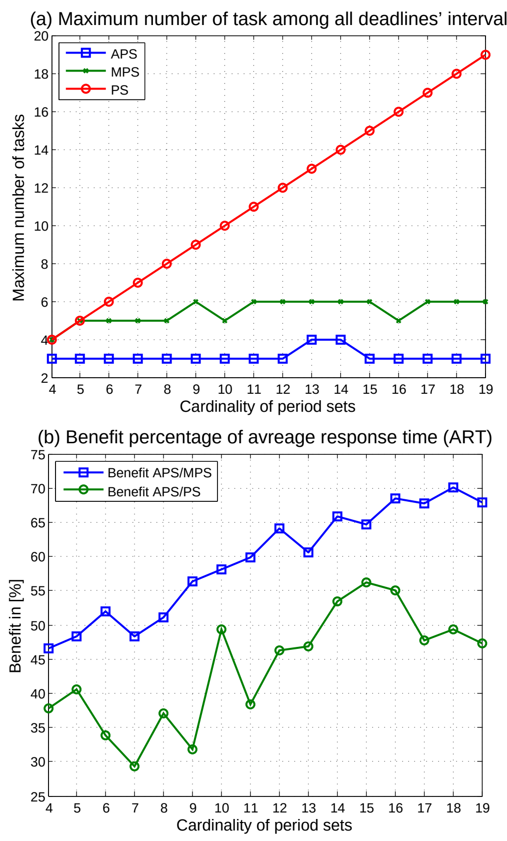

Figure 5: Performance results for zone A.

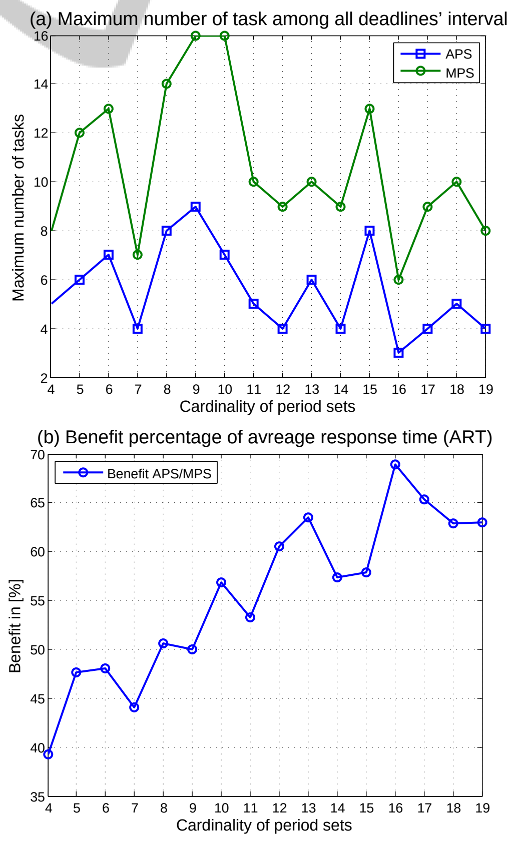

Figure 6: Performance results for zone B.

vanced solution based on task sequencer with arbitrary periods (APS). However, to our knowledge no efficient solution has been proposed to optimize both the number of tasks and system schedulability.

In this paper, a novel method has been proposed for mapping runnables to tasks based upon the APS approach. This method uses the activation offset property of runnables in order to minimize the resulting number of tasks. Experimental results show that the number of tasks is reduced significantly. Furthermore, our method increases by 34% the system schedulability bound in comparison to PS approach. It can be outlined that the PS approach exhibits a better performance in terms of system response time compared to the MPS approach but this is in the detriment of the tasks number. Simulations demonstrate that our APS approach outperforms both PS and MPS ones regarding both the number of tasks and the system response time. We aim also to investigate the experiments setup by extending it to cover more complex system configuration. In particular, we would like to measure the performance of our approach when varying both utilization factor and tasks deadline. Further, a work in progress has been started to test our approach on a real board (NEC V850) with AUTOSAR 3.2. The preliminary obtained results are satisfactory but have to be also investigated to cover more system configuration.

Precedence and shared resources constraints will be also considered in the ongoing work. Finally, we plan to apply this heuristic to multi-cores architectures in a short future.

REFERENCES

Audsley, N. (1991). *Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times*. Technical report. University of York, Department of Computer Science.

Bini, E. and Buttazzo, G. C. (2005). Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154.

Haibo, Z. and Di Natale, M. Efficient implementation of autosar components with minimal memory usage. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 130–137.

Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61.

Ming, Z. and Zonghua, G. Optimization issues in mapping autosar components to distributed multithreaded implementations. In *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*, pages 23–29.

Monot, A., Navet, N., Bavoux, B., and Simonot-Lion, F. (2012). Multisource software on multicore automotive ecus: Combining runnable sequencing with task scheduling. *Industrial Electronics, IEEE Transactions on*, 59(10):3934–3942.

Navet, N., Monot, A., Bavoux, B., and Simonot-Lion, F. (2010). Multi-source and multicore automotive ecus - os protection mechanisms and scheduling. In *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, pages 3734–3741.

Partnership, A. (2011). Technical overview. *http://www.autosar.org/*, V2.2.2 R3.2 Rev 1.

Rongshen, L., Hong, L., Wei, P., Yi, Z., and Minde, Z. An approach to optimize intra-ecu communication based on mapping of autosar runnable entities. In *Embedded Software and Systems, 2009. ICESS '09. International Conference on*, pages 138–143.

Wozniak, E., Mehiaoui, A., Mraidha, C., Piergiovanni, S. T., and Gerard, S. (2013). An optimization approach for the synthesis of AUTOSAR architectures. In *Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA 2013, Cagliari, Italy, September 10-13, 2013*, pages 1–10.

Zeng, H., Di Natale, M., and Zhu, Q. (2012). Optimizing stack memory requirements for real-time embedded applications. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8. IEEE.