Schedulability-Aware SPM Allocation for Preemptive Hard Real-Time Systems with Arbitrary Activation Patterns

Arno Luppold and Heiko Falk
Institute of Embedded Systems
Hamburg University of Technology
Hamburg, Germany
Email: {Arno.Luppold,Heiko.Falk}@tuhh.de

***Abstract*—In hard real-time multi-tasking systems each task has to meet its deadline under any circumstances. If one or several tasks violate their timing constraints, compiler optimizations can be used to optimize the Worst-Case Execution Time (WCET) of each task with a focus on the system's schedulability. Existing approaches are limited to single-tasking or strictly periodic multi-tasking systems. We propose a compiler optimization to perform a schedulability-aware static instruction Scratchpad Allocation for arbitrary activation patterns and deadlines. The approach is based on Integer-Linear Programming and is evaluated for the Infineon TriCore TC1796 microcontroller.**

I. Introduction

Using fast but small onchip Scratchpad Memories (SPM) is a powerful way to improve on the Worst-Case Response Times (WCRT) of tasks in a multi-tasking hard real-time system. Unfortunately, compilers like GCC or LLVM do not offer a timing model and are not able to assign program parts to the SPM efficiently. In multi-tasking systems, the compiler has to decide which parts of which tasks should be optimized to which extent in order to achieve schedulability. Previously, we proposed an SPM optimization for strictly periodic systems [1] based on the WCET-Aware C Compiler (WCC) [2].

The key contributions of this paper are:

- Our approach uses an event-based task model to optimize single-core multi-tasking systems with arbitrary activation patterns and deadlines.
- It works for both fixed and dynamic priority scheduling.
- The approach is evaluated on the Infineon TriCore TC1796.

This paper is organized as follows: Section II presents an overview of the related work. Section III explains the system model and notational conventions used throughout the paper. Section IV gives a brief overview of the existing single-tasking ILP framework. Section V describes our multi-tasking model. In Section VI, we show and discuss our evaluation results. A conclusion and a brief discussion of future challenges close the paper.

II. Related Work

Modern schedulability analysis approaches [3], [4] use event models to precisely analyze systems with arbitrary activation patterns. However, the approaches work at the system level. The approaches do not aim at optimizing tasks on a code level in order to fix a system which does not meet its real-time constraints. Several works in the field of sensitivity analysis exist to determine new activation patterns to achieve schedulability with minimal changes of the tasks' periods [5], [6], [7]. The approach presented by Bini et al. [5] can also calculate new values for each task's WCET which may lead to a schedulable system. However, all these approaches lack knowledge of the internal structure of a task, its actual machine operations and the features and drawbacks of the underlying hardware. Therefore, any proposed changes to the tasks' WCETs are speculative.

Suhendra et al. [8] provide an approach based on Integer-Linear Programming (ILP) to minimize a task's WCET by assigning data to a fast SPM. Falk and Lokuciejewski [2] present a compiler framework that includes several WCET-aware code optimizations, including an instruction SPM optimization [9] based on an ILP model of the program's control-flow graph. Although not described in the paper, the optimization could theoretically be used for multi-tasking systems by performing the allocation for each task separately and then reloading the SPM at each preemption. However, this will heavily increase preemption costs and is therefore often not useful in practice. Instead, the SPM should be shared between all tasks and be programmed only once during the system's initialization routine. In this case, an optimization must be aware of each task's activation pattern and deadline in order to use the limited SPM as efficient as possible. Recently, an ILP-based optimization was proposed [1] to tackle this issue, but is limited to strictly periodic multi-tasking systems. The approach has further restrictions on the tasks' deadlines: For fixed-priority scheduling, deadlines must be lower than or equal to the corresponding task's period. Additionally, for dynamic priority scheduling like Earliest Deadline First (EDF), only implicit-deadline systems with the deadline equal to the period can be handled.

III. Preliminaries

Table I shows the abbreviations which are used throughout this paper. Uppercase letters describe values which are calculated outside the upcoming ILP model or physical constants. Lowercase letters denote values which are added to the ILP

978-3-9815370-8-6/17/$31.00

TABLE I
NOMENCLATURE USED WITHIN THIS PAPER

Abbreviation	Description
$b_{N,i,j}$	Binary ILP variable set to 1 if τ_j can preempt τ_i at least N times.
B_i	A basic block.
i	Index variable.
c_i	WCET of task τ_i.
$C_{i,Flash}$	WCET of one execution of the single basic block B_i if it is assigned to flash memory.
$C_{i,SPM}$	WCET of one execution of the single basic block B_i if it is assigned to the SPM.
D_i	Deadline of task τ_i.
$\hat{D}$	Maximum relative deadline of all tasks.
Δt	A time interval.
ΔT_B	The busy window.
$\epsilon(n, \Theta)$	Interval function for the event function Θ. n denotes the number of events.
$\eta(\Delta t, \Theta)$	Event density function.
e_j	Safe bound on the eviction delay inflicted to the system if τ_j is executed.
Γ	A set containing all tasks of the the system.
J_i	Activation jitter of task τ_i.
K	Number of times the currently analyzed task is executed within the analyzed time interval.
L	A large constant which defines a lower bound.
M	A large constant which defines an upper bound.
P_i	Period of task τ_i.
r_i	ILP integer variable expressing the WCRT of task τ_i.
S_i	Net size of basic block B_i.
S_{SPM}	Size of the SPM.
$t_{K,i,j}$	ILP integer variable expressing the maximum preemption penalties added by τ_j to the WCRT of K instances of τ_i.
τ_i	Task i.
Θ_i	Event function for task τ_i.
u	Overall load of the system.
w_i	ILP integer variable which holds maximum execution time from the beginning of basic block B_i until the end of the block's function.
x_i	Binary ILP variable set to 1 if basic block B_i is assigned to the SPM.

as variables and are calculated by the ILP solver. For the task model used in this work, we make the following assumptions:

- Time is expressed as multiples of CPU clock cycles and can thus be treated as an integer value.
- We assume that flow facts which describe maximum loops iterations have been calculated previously.

We use an event-based task model to formulate our approach. A task is defined as a tuple $\tau_i = (c, D, \Theta)$. c is the WCET of the task, D its relative deadline. Θ is a place holder which describes a concrete but arbitrarily modeled activation pattern of the task. It follows the event model introduced by Gresser in [3]. For fixed-priority systems, the tasks' indices equal their priority, with 0 being the highest priority. E.g., for a system Γ consisting of 3 tasks, τ_0 is the task with the highest and τ_2 is the task with the lowest priority.

To allow practical usage, the following two functions must be provided: The density function $\eta(\Delta t, \Theta)$ and the interval function $\epsilon(n, \Theta)$, which can be seen as the inverse of the density function:

Definition 1. *The density function $\eta(\Delta t, \Theta)$ denotes the maximum number of events for a given Θ in the right-open time interval Δt [3].*

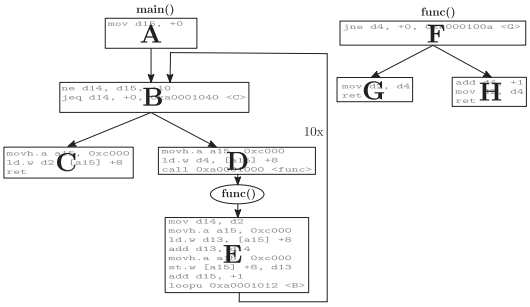


Fig. 1. Control Flow of an Exemplary System

Definition 2. *The interval function $\epsilon(n, \Theta)$ defines the minimal time interval in which n events are generated by Θ [3].*

Θ can represent any event-based model. E.g., event streams as proposed by Albers et al. [10] may be used to model arbitrary task stimulations. In our upcoming approach, the concrete event model is completely arbitrary, as long as density and interval functions can be given. For the sake of simplicity, we use periodically triggered tasks with jitter when illustrating examples or evaluating the approach:

A periodic event function with jitter for a task τ_i is defined by a period P_i and a maximum jitter J_i. Let Θ_i represent this event function. As shown by Richter [4], the maximum event density as well as the interval function can be expressed as:

$$\eta(\Delta t, \Theta_i) = \left\lceil \frac{\Delta t + J_i}{P_i} \right\rceil \quad (1)$$

$$\epsilon(n, \Theta_i) = \max\left((n-1)\, P_i - J_i, 0\right) \quad (2)$$

IV. SINGLE-TASK SPM OPTIMIZATION

This section gives a brief overview of the ILP-based SPM allocation for single-tasking systems [9]. The model operates at the basic block level on the Control Flow Graph (CFG) of a program. Prior to creating the ILP formulation, the program's timing behavior is evaluated twice using a WCET analyzer like AbsInt aiT [11]. In the first run, the WCET is analyzed if the whole program is assigned to the main Flash memory. From this analysis data, the WCET $C_{i,Flash}$ of each basic block B_i is extracted. For the second WCET analysis, the SPM is virtually increased such that the whole program can be fully assigned to the SPM. $C_{i,SPM}$ is the WCET of block B_i if it is assigned to the SPM. These data are then used to build the ILP constraints used to determine which blocks should really be assigned to the SPM. Figure 1 shows the control flow of a minimal program with two functions. First, the achievable timing gain G_i is calculated for each basic block B_i:

$$G_i = C_{i,Flash} - C_{i,SPM} \quad (3)$$

The ILP model is then built on a function level, starting at the exit nodes of the CFG, implicitly summing up the WCET of the whole program starting at each basic block. For function `func()`, the accumulated WCETs w_G and w_H from the start of each exit block to the end of the function equal the execution time of the exit blocks themselves:

$$w_G = C_{G,Flash} - x_G \cdot G_G \quad (4)$$

$$w_H = C_{H,Flash} - x_H \cdot G_H \quad (5)$$

For each basic block B_i, an additional binary decision variable x_i is introduced. If the ILP solver chooses x to be 1, then the basic block will be assigned to SPM, otherwise the block remains in the flash memory. The function's entry block B_F is then calculated as

$$w_F \geq C_{F,Flash} - x_F \cdot G_F + w_G \quad (6)$$
$$w_F \geq C_{F,Flash} - x_F \cdot G_F + w_H \quad (7)$$

This way, the accumulated WCET for one call to `func()` is constrained by the longest path through the function.

In the `main()` function, the call to `func()` by block B_D is simply added as another summand:

$$w_D \geq C_{D,Flash} - x_D \cdot G_D + w_F + w_E \quad (8)$$

The loop starting at block B_B cannot be modeled directly using this scheme. The back edge going from B_E to B_B is removed and a meta block B_{Entry} is inserted. This block expresses the execution time of one iteration of the loop, starting at its entry node:

$$w_{Entry} \geq C_{B,Flash} - x_B \cdot G_B + w_D \quad (9)$$
$$w_D \geq C_{D,Flash} - x_D \cdot G_D + w_F + w_E \quad (10)$$
$$w_E = C_{E,Flash} - x_E \cdot G_E \quad (11)$$

The loop's execution time c_{Loop} can now be modeled by multiplying the loop's entry variable w_{Entry} with the worst-case number of loop iterations. In this example, this is 10, as annotated in Figure 1.

$$c_{Loop} \geq 10 \cdot w_{Entry} \quad (12)$$

Now, the accumulated WCET of block B_A can be modeled:

$$w_A \geq C_{A,Flash} - x_A \cdot G_A + w_{Loop} \quad (13)$$
$$w_{Loop} \geq c_{Loop} + C_{B,Flash} - x_B \cdot G_B + w_C \quad (14)$$
$$w_C = C_{C,Flash} - x_C \cdot G_C \quad (15)$$

It can be observed that 11 executions of block B_B are accounted for: 10 times the loop is entered, and at the 11th execution, the loop is exited via block B_C.

Finally, one additional constraint must restrict the number of basic blocks which are assigned to the SPM to the actual size of the SPM S_{SPM}. If the size of a basic block A is defined as S_A, this can be achieved by:

$$S_{SPM} \geq S_A \cdot x_A + S_B \cdot x_B + S_C \cdot x_C + \cdots + S_E \cdot x_E \quad (16)$$

Costs of additionally needed jump instructions as well as platform specific characteristics like branch predictions have been modeled by previous works [9], [1]. We included both for our evaluation. However, due to the lack of novelty and the limited space, we will not further describe them here. The model's quality can further be improved by, e.g., including basic blocks' execution contexts. However, this would increase the ILP's complexity, resulting in longer solving times.

For single-tasking systems, the ILP's objective may simply be set to minimize the overall WCET w_A. For multi-tasking systems, the ILP constraints outlined above can be added independently to the ILP for each task. This leads to an ILP variable w_i for the WCET of each task i. In addition, a schedulability test has to be added to the ILP formulation and must be connected to each task's WCET variable. This general idea was previously proposed [1] for strictly periodic systems with deadlines lower than or equal to a task's period.

V. Multi-Task Model

We extend the single-tasking ILP optimization model to handle arbitrarily activated tasks with arbitrary deadlines. Our approach does not change any existing inequations. Instead, the previously described modeling of the WCET is performed for each task τ_i. As a result, there will be an ILP variable w_i for each task τ_i which holds a safe upper bound for the task's WCET c_i. We solely introduce additional constraints which limit the ILP solver to only return a solution which will result in a schedulable system. If no such solution exists, the inequation system is not feasible and the ILP solver will return with an appropriate error. We provide solutions for both fixed and dynamic priority systems. The approach inherently supports shared code. If two tasks share a common function `func()` with entry block B_F, they can both simply use the same variable w_F in their respective ILP model.

A. Fixed Priority Systems

For event-triggered fixed priority systems, Kollmann et al. [12] adapted the WCRT analysis by Lehoczky et al. [13]. The WCRT $r(\tau_i)$ of a task τ_i can be calculated by using a fixed-point computation as:

$$r(\tau_i) = \max_{\forall K \in [1,\ldots,\eta(\Delta T_B, \Theta_i)]} \{r(K, \tau_i) - \epsilon(K, \Theta_i)\} \quad (17)$$

$$r(K, \tau_i) = \min \left(\Delta t \,|\Delta t = K \cdot c_i + \sum_{j=0}^{i-1} (\eta(\Delta t, \Theta_i) \cdot c_j) \right) \quad (18)$$

The additionally introduced factor K accounts for multiple instances of the currently analyzed task τ_i itself. It denotes the number of times that τ_i is executed within the analyzed time interval. The WCET c_i of τ_i corresponds to the accumulated WCET of the task's entry block in Section IV. For the example given in Figure 1, $c_{\text{main}} = w_A$.

To simplify notations, we express the maximum possible K for a specific task τ_i from equation (17) as $\hat{K}_i$:

$$\hat{K}_i := \eta(\Delta T_B, \Theta_i) \quad (19)$$

For $K = 2$, $r(K, \tau_i)$ returns the maximum time needed until two subsequent instances of τ_i have finished. Equation (17) then holds the WCRT, i.e., the maximum response time of one execution of task τ_i, over all analyzed K.

The system is not schedulable if $r(\tau_i) > D_i$ for any $\tau_i \in \Gamma$. The maximum time interval Δt is the so called busy window or busy period ΔT_B, which was introduced by [14]:

Definition 3. *The busy window is the maximum interval in which only the currently analyzed task or any task with a higher priority is executed and the processor is not idle [14].*

Proposition 1. *A safe upper bound on Δt is given by the hyper-period of all tasks. For a task τ_i with a deadline D_i*

lower than or equal to its respective minimal period, i.e., if $\eta(2, \Theta_i) \geq D_i$*, it is sufficient to use* D_i *as maximum* Δt*.*

Proof. The proof for the first part is given in [12]. For the second part of the proposition, assume that the currently analyzed task τ_i is schedulable, i.e., $r(\tau_i) \leq D_i$. Then, the busy window must by definition be smaller than or equal to the task's deadline: $\Delta T_B \leq D_i$. If $r(\tau_i) > D_i$, then the system is not schedulable, and no bigger time interval has to be investigated. □

If a task's deadline is lower than or equal to its minimal period (i.e., $D_i \leq \epsilon(2, \Theta_i)$), then $\hat{K}_i \equiv 1$, as the task cannot be blocked by itself. In this case, the task's deadline D_i can be used as a safe upper bound on ΔT_B. Otherwise, the busy window is defined by the hyper-period which may easily become very big. To reduce the busy window, tasks' periods may be adjusted [15] to minimize the hyper-period. This will introduce more pessimism, thus decreasing the amount of task sets which can be repaired by the optimization model.

To perform a schedulability oriented code optimization, the schedulability test expressed in equations (17) and (18) has to be described as a set of integer-linear inequations which can be added to the ILP framework.

The WCRT $r(\tau_i)$ for each task $\tau_i \in \Gamma$, as well as the result of each $r(K, \tau_i)$ will be expressed by the integer variables r_i and $r_{K,i}$ in the ILP. Modeling the maximum function from equation (17) is straightforward: For each argument, an inequation is added with a lower-bound constraint for r_i. Eq. (17) therefore results in the following inequation system:

$$r_i \geq r_{K,i} - \epsilon(K, \Theta_i), K = 1, \ldots, \hat{K}_i \tag{20}$$

$\epsilon(K, \Theta_i)$ is constant for any given task, as it is only dependent on the event function describing the task stimulation but not on any task's WCET and can therefore be calculated prior to solving the ILP.

Example 1. *For periodic tasks with jitter,* $\epsilon(n, \Theta_i)$ *has been defined in equation* (2)*. Using this function, equation* (20) *can be rewritten and added as inequations to the ILP:*

$$r_i \geq r_{K,i} - \max((K-1)P_i - J_i, 0), K = 1, \ldots, \hat{K}_i \tag{21}$$

For tasks with a deadline lower than or equal to its minimal activation distance, $\hat{K}_i \equiv 1$. In this case, inequations for higher numbers of K do not have to be modeled in the ILP.

An additional constraint ensures that the response time has to be lower than or equal to the task's deadline:

$$r_i \leq D_i \tag{22}$$

This constraint ensures that the inequation system will be unsolvable if the system will not hold all timing constraints. As a result of our constraint based approach, the ILP's objective function may be chosen arbitrarily.

For the ILP formulation, the iterative formula given by Equation (18) must be rewritten to a set of linear inequations. A safe lower bound on the time interval in which τ_i may be interrupted by any higher priority task is given by the task's WCET c_i. Because all execution times are positive, it is obvious that $r(K, \tau_i)$ is monotonically increasing. This means that larger values of Δt will never result in a smaller value of $r(K, \tau_i)$. Therefore, a safe lower bound and initial value of Δt is given by c_i. To express $r_{K,i}$ by a set of linear inequations, we substitute the preemption penalty that is added to $r_{K,i}$ by a higher priority task j for each K by the ILP variable $t_{K,i,j}$. This leads to the following ILP inequations:

$$r_{K,i} = 1 \cdot c_i + \sum_{j=0}^{i-1} t_{K,i,j}, K = 1, \ldots, \hat{K}_i \tag{23}$$

For each $K \in [1, \ldots, \hat{K}_i]$, we can express the preemption penalty $t_{K,i,j}$ as a set of conditional inequations:

$$t_{K,i,j} \geq \begin{cases} 1 \cdot c_j + 1 \cdot e_j & \text{if } r_{K,i} \geq \epsilon(1, \Theta_j) \\ N \cdot c_j + N \cdot e_j & \text{if } r_{K,i} \geq \epsilon(N, \Theta_j) \\ & \ldots \\ \hat{K}_i \cdot c_j + \hat{K}_i \cdot e_j & \text{if } r_{K,i} \geq \epsilon\left(\hat{K}_i, \Theta_j\right) \end{cases} \tag{24}$$

The underlying idea is analogous to that given in [1] for strictly periodic systems: If the response time r_i of task τ_i is greater than or equal to the minimal distance between N subsequent activations of a higher priority task τ_j, then τ_i may be preempted N times by this task τ_j. Due to the fact that the number of evictions of task τ_i by τ_j is known, the eviction penalty e_j which models the overhead of a preemption can easily be added to the inequation system. If preemption costs can be neglected, e_j may be set to 0.

Equation (24) can be expressed in the ILP by adding a set of inequations for each $N \in [1, \ldots, \hat{K}_i]$:

$$t_{K,i,j} + b_{N,i,j} \cdot M_{N,j} \geq N \cdot c_j \tag{25}$$

$$r_{K,i} \geq \epsilon(N, \Theta_j) + L_{K,i} \cdot (1 - b_{N,i,j}) \tag{26}$$

$b_{N,i,j}$ is a binary decision variable. Eq. (26) enforces the ILP solver to set $b_{N,i,j}$ to 0 if $r_{K,i}$ is greater than the minimal time interval in which task τ_j may be triggered N times. This forces the preemption penalty $t_{K,i,j}$ in eq. (25) to be greater or equal to $N \cdot c_j$. The "greater than or equal to" in eq. (26) stems from the fact that according to definition 1 the density function operates on a right-open interval. $M_{N,j}$ and $L_{K,i}$ are sufficiently large constants. $M_{N,j}$ ensures that eq. (25) will always hold if $b_{N,i,j}$ is chosen to be 1 by the ILP solver. In a schedulable system, the WCET c_j must be smaller than the respective deadline D_j, therefore a safe bound on $M_{N,j}$ is:

$$M_{N,j} = N \cdot D_j \tag{27}$$

$L_{K,i}$ ensures that eq. (26) is always fulfilled, if $b_{N,i,j}$ is 0. In analogy to $M_{N,j}$, $r_{K,i} < K \cdot D_i$ for a schedulable system. Therefore, a sufficiently large $L_{K,i}$ can be chosen as:

$$L_{K,i} = K \cdot D_i \tag{28}$$

Any ILP solver can now be used to solve the proposed inequation system. As the schedulability conditions are added as ILP constraints, an arbitrary optimization objective may be chosen by the user. If the ILP solver returns a solution, this solution is guaranteed to lead to a schedulable system. If no schedulable system can be established, either due to a too

pessimistic modeling of the WCET behavior of the system or due to the limited hardware capabilities, the ILP will be infeasible and the solver will return an appropriate error.

B. Dynamic Priority Scheduling

For systems scheduled under an optimal dynamic scheduler like Earliest Deadline First (EDF), the previously described inequations cannot be used. Instead, the so-called processor demand test proposed by Baruah [16] may be used to calculate whether a system is feasible or not:

Definition 4. *A system is schedulable under EDF, if the following condition holds for any time interval Δt [16]:*

$$\Delta t \geq \sum_{\forall \tau_i \in \Gamma} \{\eta\left(\Delta t - D_i, \Theta_i\right) \cdot c_i\} \tag{29}$$

Analogously to fixed priority systems, a safe upper bound for Δt is given by the busy window for arbitrary systems. For tasks with a deadline lower or equal to its minimal stimulation period, the deadline is a safe upper bound. Obviously, the event density function is not dependent on whether fixed or dynamic priorities are used. Therefore, $\eta\left(\Delta t, \Theta_\tau\right)$ is still piecewise constant. As a result, it is sufficient to evaluate equation (29) at its discontinuities only. To express equation (29) as a set of linear inequations, we first define Θ_Γ as the union over all event functions of all tasks:

$$\Theta_\Gamma = \bigcup_{\forall \tau_i \in \Gamma} \Theta_i \tag{30}$$

The schedulability test provided in equation (29) can now be expressed as a set of ILP inequations:

$$\begin{aligned} \epsilon\left(1, \Theta_\Gamma\right) \geq \sum_{\tau_i \in \Gamma} \{&\eta\left(\epsilon\left(1, \Theta_\Gamma\right) - D_i, \Theta_i\right) \cdot c_i \\ &+\eta\left(\epsilon\left(1, \Theta_\Gamma\right) - D_i, \Theta_i\right) \cdot e_i\} \end{aligned} \tag{31}$$

$$\begin{aligned} \epsilon\left(2, \Theta_\Gamma\right) \geq \sum_{\tau_i \in \Gamma} \{&\eta\left(\epsilon\left(2, \Theta_\Gamma\right) - D_i, \Theta_i\right) \cdot c_i \\ &+\eta\left(\epsilon\left(2, \Theta_\Gamma\right) - D_i, \Theta_i\right) \cdot e_i\} \end{aligned} \tag{32}$$

$$\cdots$$

$$\begin{aligned} \Delta T_B \geq \sum_{\tau_i \in \Gamma} \{&\eta\left(\Delta T_B - D_i, \Theta_i\right) \cdot c_i \\ &+\eta\left(T_B - D_i, \Theta_i\right) \cdot e_i\} \end{aligned} \tag{33}$$

By using the union over all event functions Θ_Γ, all points in time at which any task will be activated and therefore all discontinuities of the piecewise constant density function $\eta\left(\Delta t, \Theta_\Gamma\right)$ are tested. If a modeled system cannot be repaired, the required CPU time is greater than the available time for at least one time interval. As a result, the inequation system will be infeasible. Similar to fixed priority scheduling, e_i may be used to model additional costs of a context switch.

The upper bound on the interval function $\epsilon\left(n, \Theta_\Gamma\right)$ is given by the busy window ΔT_B. For systems where no task's minimum activation period is smaller than the respective deadline, i.e., $\epsilon\left(2, \Theta_i\right) \geq D_i \forall \tau_i \in \Gamma$, the maximum deadline $\hat{D}$ can be used as a safe upper bound:

$$\hat{D} = \max_{\forall \tau_i \in \Gamma} D_i \tag{34}$$

VI. Evaluation

We exemplarily chose to evaluate our approach for periodic tasks with jitter. We evaluate our approach on the Infineon TriCore TC1796 running at 150 MHz as target platform. We use the WCET-aware C Compiler WCC as compiler framework for our evaluation. Accesses to the TriCore's SPM are completed in 1 CPU cycle. Each access to the flash memory takes 6 cycles. All timing analyses are performed using AbsInt aiT version b217166 [11]. The ILPs are solved on an Intel Xeon server using IBM CPLEX 12.5. Prior to the SPM allocation, we apply the compiler's `-O2` flag which enables several standard ACET-oriented optimizations. To evaluate our approach, we created multiple task sets by randomly clustering benchmarks from the MRTC benchmark suite [17]. [18] provides a version of the MRTC benchmarks which is completely annotated with flow facts. We choose task set sizes of 2, 5, 15, 20 and 25 tasks. Because the MRTC benchmarks are quite small, we artificially resized the available SPM size to 40% of the size of each task set to avoid trivial solutions. For each task set size, we randomly assembled 30 different task sets. In conclusion, we generate and evaluate 180 different task sets. We then randomly generate periods, deadlines and jitter for each task, with deadlines up to factor 1.5 of each task's period. To account for higher scheduling overhead, we now assumed a penalty of 100 clock cycles for each preemption.

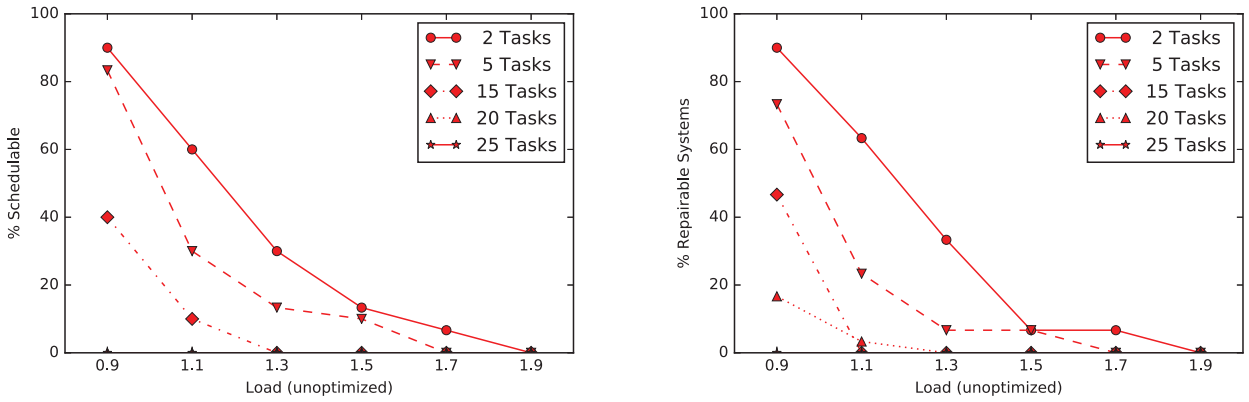
Our compiler framework then automatically adapts the linker script with the solution returned by the ILP solver, to allocate the basic blocks to their final memory region.

To get an impression of the time needed to find *one* valid solution, we set a dummy optimization target for the ILP solver. This way, the ILP solver exits as soon as it finds a solution which satisfies all ILP constraints. This is sufficient, as we do not analyze the robustness of the repaired systems.

Figure 2 shows how many systems can be repaired with both EDF and Deadline Monotonic Scheduling (DMS) for each task set size. WCET analyses were performed using aiT, ensuring that each repaired system was not schedulable prior to our optimization but was schedulable afterwards. It can be seen that for small task sets, even systems with an initial load of 1.5 may be repaired by our approach. With growing task sets, the impact of the constant scheduling overhead increases, thus the optimization's repair rate decreases accordingly. Figure 3 shows the corresponding ILP solving times for the different target loads. Except from some outliers, the ILP solver mostly needs less than 2 minutes.

VII. Conclusion

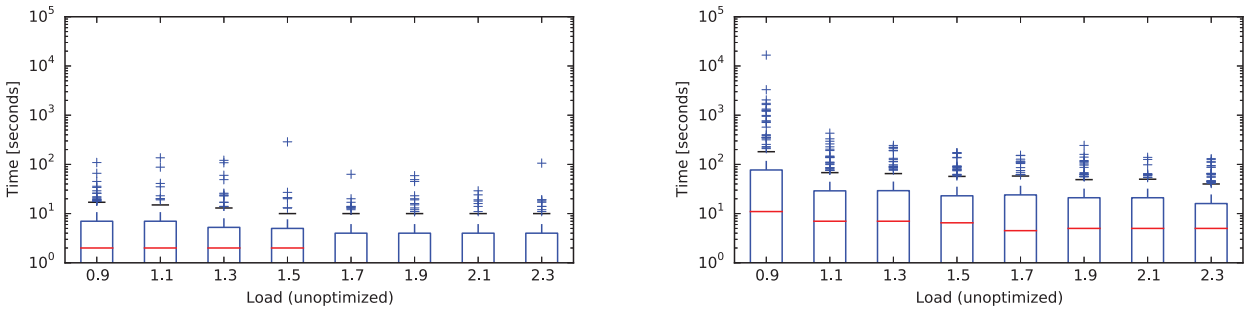
We propose a method for a compiler-based schedulability-oriented static SPM allocation. Our approach focuses on timing-critical single-core systems. The usage of an event-based task model allows us to repair both fixed and dynamic priority systems with arbitrary task activation patterns and deadlines. In the future, we aim at extending our framework to integrate instruction and data caches, and to account for caches and cache-related preemption delays (CRPD). Additionally, we aim at integrating support for inter-task dependencies and outline a multi-core approach.



(a) EDF Scheduling Constraints

(b) DMS Scheduling Constraints

Fig. 2. Repairable Systems Dependant on the Number of Tasks in a Task Set



(a) EDF Scheduling Constraints

(b) DMS Scheduling Constraints

Fig. 3. ILP solving times for MRTC task sets for different original loads and with no objective function. For each load, 180 task sets with multiple number of tasks were evaluated. Each task's deadline may be greater than its respective period. Preemption penalty was chosen to be 100 clock cycles per preemption.

ACKNOWLEDGMENTS

This work received funding from Deutsche Forschungsgemeinschaft (DFG) under grant FA 1017/1-2. This work was partially supported by COST Action IC1202: Timing Analysis On Code-Level (TACLe).

REFERENCES

[1] A. Luppold and H. Falk, "Code Optimization of Periodic Preemptive Hard Real-Time Multitasking Systems," in *Proceedings of ISORC*, 2015, pp. 35–42.

[2] H. Falk and P. Lokuciejewski, "A Compiler Framework for the Reduction of Worst-Casease Execution Times," *RTS*, vol. 46, no. 2, pp. 251–300, 2010.

[3] K. Gresser, "An Event Model for Deadline Verification of Hard Real-Time Systems," in *Proceedings of ECRTS*, 1993, pp. 118–123.

[4] K. Richter, "Compositional Scheduling Analysis Using Standard Event Models. The SymTA/S Approach," Ph.D. dissertation, Technical University Carolo-Wilhelmina of Braunschweig, 2005.

[5] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity Analysis for Fixed-Priority Real-Time Systems," in *Proceedings of ECRTS*, 2006, pp. 13–22.

[6] M. Neukirchner, S. Quinton, T. Michaels, P. Axer, and R. Ernst, "Sensitivity Analysis for Arbitrary Activation Patterns in Real-time Systems," in *Proceedings of DATE*, 2013, pp. 135–140.

[7] F. Zhang, A. Burns, and S. Baruah, "Sensitivity Analysis of the Minimum Task Period for Arbitrary Deadline Real-Time Systems," in *Proceedings of PRDC*, 2010, pp. 101–108.

[8] V. Suhendra, T. Mitra, A. Roychoudhury, and T. Chen, "WCET Centric Data Allocation to Scratchpad Memory," in *Proceedings of RTSS*, 2005, pp. 223–232.

[9] H. Falk and J. C. Kleinsorge, "Optimal Static WCET-aware Scratchpad Allocation of Program Code," in *Proceedings of DAC*, 2009, pp. 732–737.

[10] K. Albers and F. Slomka, "An Event Stream Driven Approximation for the Analysis of Real-Time Systems," in *Proceedings of ECRTS*, 2004, pp. 187–195.

[11] AbsInt Angewandte Informatik, GmbH. (2016) aiT Worst-Case Execution Time Analyzers.

[12] S. Kollmann, V. Pollex, K. Kempf, and F. Slomka, "A Scalable Approach for the Description of Dependencies in Hard Real-Time Systems," in *Leveraging Applications of Formal Methods, Verification, and Validation*, ser. Lecture Notes in Computer Science. Springer, 2010, vol. 6416, pp. 397–411.

[13] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," in *Proceedings of RTSS*, 1989, pp. 166–171.

[14] J. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," in *Proceedings of RTSS*, 1990, pp. 201–209.

[15] J. Xu, "A Method for Adjusting the Periods of Periodic Processes to Reduce the Least Common Multiple of the Period Lengths in Real-Time Embedded Systems," in *Proceedings of MESA*, 2010, pp. 288–294.

[16] S. K. Baruah, "Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks," *RTS*, vol. 24, no. 1, pp. 93–128, 2003.

[17] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET Benchmarks – Past, Present and Future," in *Proceedings of WCET*, 2010, pp. 137–147.

[18] H. Falk, S. Altmeyer, P. Hellinckx, B. Lisper, W. Puffitsch, C. Rochange, M. Schoeberl, R. B. Sørensen, P. Wägemann, and S. Wegener, "TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research," in *Proceedings of WCET*, 2016.