Schedulability-Oriented WCET-Optimization of Hard Real-Time Multitasking Systems

Arno Luppold and Heiko Falk
Institute of Embedded Systems / Real-Time Systems
Ulm University
{arno.luppold|heiko.falk}@uni-ulm.de

ABSTRACT

In multitasking hard real-time systems, each task's response time must provably be lower than or equal to its deadline. If the system does not hold all timing constraints, WCET-oriented optimizing compilers can be used to improve each task's worst-case runtime behaviour. However, current optimizations do not account for a system's schedulability constraints. We provide an approach based on Integer-Linear Programming (ILP) for schedulability-oriented WCET optimization of hard real-time systems.

1. INTRODUCTION

One of the main issues for the compiler is to determine which tasks should be optimized to which amount in order to achieve a schedulable system while complying with the limited hardware resources on embedded platforms. Current approaches on sensitivity analysis are able to provide WCETs for each task which will result in a schedulable system. However, these approaches are performed on a system-level basis, without respecting the compiler's low-level optimization capabilities for each task or the system's specific hardware capabilities and restrictions. As an example which will be used throughout this paper, many embedded platforms provide a very fast but small memory, called scratchpad memory (SPM). Putting a task into the SPM reduces its execution time, but the SPM is usually too small to keep all tasks - sometimes even too small for one complete single task. We therefore aim for integrating schedulability analysis into an optimizing WCET-aware compiler framework. The key contributions of this paper are:

- We provide an ILP-based method to determine which tasks should be optimized to which amount in order to achieve a schedulable system.
- We provide approaches on both dynamic and fixed priority based systems.
- We provide initial evaluation results which show the potential of our approach.

This paper is organized as follows: Section 2 gives a brief overview of related projects. Section 3 shows our underlying approach. In Section 4 we integrate the approach into a WCET-oriented optimizing compiler framework and provide initial evaluation results. This paper closes with a conclusion and future challenges.

2. RELATED WORK

In [7], Liu and Layland proposed an iterative approach to calculate the response times of a hard real-time multitasking system.

$$r_i = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{r_j}{T_j} \right\rceil \cdot c_i$$ (1)

In their approach, each task $r_i$ is defined by a priority $i$, a net worst-case execution time (WCET) $c_i$, and the minimal period of two consecutive stimuli $T_i$. The system is defined to be schedulable, if each task's WCRT $r_i$ is lower than or equal to its deadline $d_i$. Since then, response time analysis has been intensively studied, and more flexible task models have been proposed. Huge progress has been made to provide tight bounds on event-driven task sets which are not strictly periodical [1, 5].

However, those approaches don't give hints on how an unschedulable system could be turned into a schedulable one. To the best of our knowledge, the first work covering the issue of modification of task and system parameters was [6]. In this approach, a constant factor was introduced which scaled all tasks equally up to the system's first deadline miss. Analysing how system parameters may be tuned to keep or to establish a schedulable system is called sensitivity analysis. Since that first approach, sensitivity analysis has been improved massively. A method to sustain schedulability when adding a new task to a feasible system which is scheduled using EDF is described by [11]. Arbitrary activation patterns are analyzed in [8], and [9] proposes a method to perform a sensitivity analysis on distributed systems. Additionally, theoretical approaches have been introduced for parameter tuning of rate-monotonic scheduling in [2].

Although providing a system-level view on a system's sensitivity and robustness, those approaches either focus on task stimuli relaxation or do not offer clear optimization hints for optimizing compilers. They do offer sets of valid system parameters but the system engineer is still required to manually choose a set and try and optimize the tasks accordingly. Depending on the system's size, this may result in a tedious challenge. We therefore aim at providing a new approach which can be tightly coupled to both existing and future WCET-oriented compiler optimizations, allowing for a largely automated design and compilation of hard embedded real-time systems.
3. APPROACH

3.1 Task Model

At the current state of our work we focus on strictly periodical preemptive systems. A task \( \tau_i \) is defined by its WCET \( c_i \), its deadline \( d_i \), period \( p_i \) and execution period \( T_i \). We currently assume that each task’s deadline \( d \) is lower than or equal to its respective period \( T \). In case of fixed priorities, the index \( i \) denotes the task’s priority, with 0 being the highest priority. We assume all timing values to be integers. This may be achieved without loss of generality e.g., by using CPU clock cycles as time unit.

3.2 ILP Model

We use the formulation provided by [10] which models the WCET of a task using integer-linear constraints. A task is split into its basic blocks, which are defined as an instruction sequence that must be traversed from top to bottom. Therefore, e.g., any branch instruction must be the very last instruction of a basic block. Variables are introduced for each basic block’s execution time. The accumulated execution time \( w_i \) of a basic block \( i \) is defined to be greater than or equal to \( i \)’s execution \( v_i \) plus the accumulated execution time of its successor \( j \):

\[
w_i \geq v_i + w_j\quad (2)
\]

Multiple successors are described by using multiple constraints. The ILP can then be solved using an objective function which tries to minimize the accumulated execution time of a task’s entry block. This block provides a safe overapproximation of the task’s WCET. The model can be accompanied by further constraints to perform ILP-based WCET optimizations. E.g., [3] used the model to perform a WCET-oriented scratch-pad memory (SPM) allocation. They perform static timing analysis on a given program, once with the whole program in slow Flash memory resulting in a worst-case execution time \( v_{i,F} \) for each basic block, and once with the whole program being assigned to the SPM, resulting in \( v_{i,S} \). Eq. (2) may then be extended to

\[
w_i \geq v_{i,F} - b_i \cdot (v_{i,F} - v_{i,S}) + w_j\quad (3)
\]

\( b_i \) denotes a binary decision variable which is set to 1 if the block is located in SPM, and 0 else. Additional constraints are added to respect the SPM’s overall size and additional jump instructions. Without size constraints, the ILP solver would simply assign the whole program to the SPM. When minimizing the accumulated execution time of the program’s entry block \( w_0 \), the ILP solver will minimize the overall WCET of the whole program. Additionally, \( w_0 \) will be a safe overapproximation of the program’s WCET. In a multitasking environment \( w_0 \) corresponds to the task’s WCET \( c \). Our approach integrates the model by Suhendra et al. and illustrates its usage using a slightly improved version of the SPM allocation as an example for multitasking optimizations. The improvements are mostly technical and will therefore not be discussed any further.

3.3 Dynamic Priority Systems

We will consider systems which are scheduled using Earliest Deadline First (EDF). For a system which is scheduled using EDF, [7] shows that the system is schedulable if

\[
\sum_i \frac{c_i}{T_i} \leq 1\quad (4)
\]

With \( c_i \) being the WCET of task \( i \) and \( T_i \) being the minimal period between two consecutive task executions.

For dynamic priority systems where each task’s deadline equals its period, the system may easily be optimized in a schedulability oriented approach by choosing Eq. (4) as the ILP’s objective function. If the optimized system’s workload is lower than or equal to 1, the system is schedulable using EDF. Timing overheads inflicted by the task scheduler may be modeled by additional tasks.

3.4 Fixed Priority Systems

Currently our approach for fixed priority systems covers periodical systems, but should be adaptable to more complex systems like event triggered multitasking systems. We define a set of \( n \) tasks \( \tau_i \), \( 0 \leq i \leq n - 1 \), each with a period \( T_i \), a WCET \( c_i \), and deadline \( d_i \). \( \tau_0 \) is defined as the highest priority task and \( \tau_{n-1} \) the lowest priority task, respectively. \( r_i \) is defined as the WCR of \( \tau_i \), i.e., the maximum time span between a task becoming ready for execution and its end of execution, including all blocking times by other tasks. In a first step, response time analysis is performed by iterating over eq. (1) until one of the following conditions occurs:

- The resulting \( r_i \) does not change for any \( \tau_i \), and each \( r_i \leq d_i \). In this case, the system is schedulable and no further steps have to be performed.
- The resulting \( r_i \) does not change for any \( \tau_i \), but for at least one task \( \tau_i \), \( r_i > d_i \). In this case, the system is stable but is not schedulable.
- At least one \( r_i \) might get larger than the task’s period \( T_i \). In this case, the fix point iteration is aborted and the system is said to be unstable.

We define the maximum number of preemptions of a task \( \tau_i \) by another task \( \tau_j \) as \( p_{i,j} \). If the task is not stable, we define \( p_{i,j} \) to be the maximum allowed number of preemptions:

\[
p_{i,j} = \begin{cases} \frac{r_i}{T_j} & \text{if } r_i \leq T_j, j < i \\ \frac{d_j}{T_j} & \text{if } r_i > T_j, j < i \\ 0 & \text{if } j \geq i \end{cases} \quad (5)
\]

Based on eq. (1), the schedulability of a system can be expressed as a set of inequations:

\[
\begin{align*}
c_0 & \leq d_0 \\
c_1 + \left[ \frac{r_1}{T_0} \right] & \leq d_1 \\
c_2 + \left[ \frac{r_2}{T_1} \right] + \left[ \frac{r_2}{T_1} \right] & \leq d_2 \\
& \vdots \\
c_{n-1} + \left[ \frac{r_{n-1}}{T_0} \right] & \leq d_{n-1} \quad (6)
\end{align*}
\]

The system is schedulable if all equations hold. We establish linearity by substituting \( \left[ \frac{r_i}{T_j} \right] c_j \) with the precalculated \( p_{i,j} \). The WCETs \( c_i \) are left as ILP variables and may be adapted to achieve a schedulable system. Our approach will now optimize the system performing the following steps:

1. Calculate the maximum number of preemptions \( p_{i,j} \) for the original system, as described in eq. (5).
2. Create an ILP as shown in equation (6).
3. Solve the ILP, with eq. (7) as the ILP’s objective function. The ILP will provide a \( c_i \) for each task \( i \).
4. Re-calculate all \( p_{i,j} \) with the newly calculated \( c_i \).
5. Modify the ILP as shown in eq. (8) and calculate the relaxed WCETs \( c_{i,relaxed} \).
6. Generate the new objective function for any subsequent WCET optimizations, and perform the optimization itself.

To avoid a quadratic problem, it is necessary to assume a constant number of preemptions during the first iteration of the ILP. This will inevitably lead to an underestimation of the maximum allowed WCET for each task. Therefore, optimizations like the aforementioned SPM allocation cannot be integrated in this step of the algorithm. To ease compiler optimizations and to reduce side effects, we use the minimization of the relative change in each task’s WCET as objective. The target function will be defined as:

\[
\min \sum_i \frac{c_{i,orig} - c_i}{c_{i,orig}} \quad (7)
\]

\( c_{i,orig} \) is the original WCET of task \( \tau_i \) without any optimizations. \( c_i \) is the optimized WCET of the task which will lead to a schedulable system. Using this method, the ILP solver will propose WCETs for each task which will lead to a schedulable system, while trying to modify the WCETs as little as possible.

Next, we relax the calculated maximum WCETs. We use eq. (1) to calculate updated values for each \( p_{i,j} \). We can now allow each \( c_i \) to increase if the increase in execution time will not change the number of maximum preemptions. To achieve this, we modify the set of linear equations from eq. (6). For each inequation, we substitute the task’s deadline on the right-hand side of the inequation by

\[
\min \left( d_i, \frac{r_i}{T_0}, T_0, ..., \frac{r_i}{T_i-1}, T_i-1 \right) \quad (8)
\]

Each task’s WCRT may still never exceed its deadline. As an additional constraint, the task’s response time may not be larger than the minimum stimulus interval of each higher-priority task. This prevents the ILP solver from increasing a task’s WCET to an amount that would allow for additional preemptions by a higher-priority task. Figure 1 illustrates this for the tasks \( \tau_2 \) and \( \tau_4 \) with \( \left\lceil \frac{r_4}{T_2} \right\rceil = 2 \). Given the new constraints, the ILP is solved a second time, leading to the relaxed WCET \( c_{i,relaxed} \) for each task \( i \).

ILP constraints denoting compiler constraints like platform dependent lower bounds on each task’s WCET can be directly added to the second ILP. In theory, constraints for WCET-optimized optimizations like [5] could be added as well. However, those optimizations usually do not model the architectural behaviour exactly. For complexity reasons, jump timing penalties are overapproximated, and other behaviour, like changes in the CPU’s pipelines, are not modeled at all. This leads to an overapproximation of a task’s WCET when moving individual basic blocks to the SPM. We use the results from our proposed sensitivity analysis to formulate a new objective function for WCET based optimizations. Given this objective function, the tasks will be optimized with regard to the system’s schedulability, without need for exact WCET estimates within the optimization algorithm. Normalizing the tasks to account for different execution times and considering the necessary relative decrease in the tasks’ WCET leads to:

\[
\min \left( \frac{\max(c_{0,orig}, ..., c_{n,orig})}{c_{0,orig}}, \frac{c_{0,orig}}{c_{0,relaxed}}, c_0 + \sum \frac{\max(c_{0,orig}, ..., c_{n,orig})}{c_{n,orig}}, \frac{c_{n,orig}}{c_{n,relaxed}}, c_n \right) \quad (9)
\]

This objective can now be applied to WCET-oriented compiler optimizations as proposed in [3].

4. EVALUATION

We illustrate the approach’s performance on a set of tasks from the MRTC benchmark suite [4]. We use the WCET aware C compiler framework WCC which was also used by [3]. The target platform was chosen to be the Infineon TriCore 1796 micro processor running at 150MHz. The SPM size was limited to 10% of the total program size. We did not apply any standard compiler optimizations (e.g., loop unrolling) to improve comparability for the reader. All timings printed in this section were obtained using AbsInt aiT (version 14.04, build b217166). Due to the limited space in this paper, we only chose a subset of MRTC benchmarks to show the possibilities of our approach. We tried to model a realistic system with both small, but frequently executed tasks (e.g., to poll sensor values) and much larger, but less frequently called tasks (e.g., I/O operations). We chose st, lus and matmult, as they had the largest WCET out of the MRTC benchmark suite on our platform. We then randomly chose two smaller benchmarks, in this case fibcall and sqrt.

To show the effects of our different optimization approaches, we chose the periods to differ by several magnitudes. For simplification reasons, we only evaluated a strictly periodic system where each task’s deadline \( d \) equals its period \( T \).

<table>
<thead>
<tr>
<th>i</th>
<th>Name</th>
<th>( T )</th>
<th>( c_{orig} )</th>
<th>( \sum c_i )</th>
<th>( \sum c_i^T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fibcall</td>
<td>50</td>
<td>4.84</td>
<td>612</td>
<td>2.30</td>
</tr>
<tr>
<td>1</td>
<td>sqrt</td>
<td>100</td>
<td>85.79</td>
<td>100.03</td>
<td>50.11</td>
</tr>
<tr>
<td>2</td>
<td>st</td>
<td>50,000</td>
<td>3401.87</td>
<td>2570.15</td>
<td>2481.53</td>
</tr>
<tr>
<td>3</td>
<td>lus</td>
<td>75,000</td>
<td>9125.07</td>
<td>7002.59</td>
<td>9125.07</td>
</tr>
<tr>
<td>4</td>
<td>matmult</td>
<td>100,000</td>
<td>2699.29</td>
<td>1831.07</td>
<td>2699.29</td>
</tr>
</tbody>
</table>

Table 1: Dynamic priorities, times are in \( \mu s \)

Table 1 shows the WCETs for each task, without optimization, with SPM optimization which tries to minimize the sum over all WCETs, and the EDF-oriented SPM optimization. Table 3 shows the corresponding system load. It can be seen that the system is not schedulable without the SPM optimization. When applying the SPM optimization without regarding tasks’ periods, the system load even goes up, because \( c_0 \) and \( c_1 \) increase. This somewhat strange behaviour...
Table 2: Fixed priorities, times are in µs.

<table>
<thead>
<tr>
<th>Prio.</th>
<th>Name</th>
<th>T</th>
<th>(c_{orig})</th>
<th>(c_{relaxed})</th>
<th>(c_{final})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fibcall</td>
<td>50</td>
<td>4.84</td>
<td>4.84</td>
<td>4.84</td>
</tr>
<tr>
<td>1</td>
<td>sqrt</td>
<td>100</td>
<td>88.79</td>
<td>54.99</td>
<td>51.35</td>
</tr>
<tr>
<td>2</td>
<td>st</td>
<td>50000</td>
<td>3401.87</td>
<td>3401.87</td>
<td>3401.87</td>
</tr>
<tr>
<td>3</td>
<td>lms</td>
<td>75000</td>
<td>9125.07</td>
<td>9124.61</td>
<td>7002.59</td>
</tr>
<tr>
<td>4</td>
<td>matmult</td>
<td>100000</td>
<td>2699.29</td>
<td>2699.29</td>
<td>2699.29</td>
</tr>
</tbody>
</table>

Table 3: System Load

<table>
<thead>
<tr>
<th></th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unopt.</td>
<td>1.20</td>
</tr>
<tr>
<td>(\min \sum c_i)</td>
<td>1.27</td>
</tr>
<tr>
<td>(\min \sum c_i/\tau_i)</td>
<td>0.74</td>
</tr>
<tr>
<td>Opt.</td>
<td>Fixed Prio.</td>
</tr>
</tbody>
</table>

Table 4: Response time using RMS, times are in µs

<table>
<thead>
<tr>
<th>Name</th>
<th>(r) opt dyn.</th>
<th>(r) fixed</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>fibcall</td>
<td>2.2</td>
<td>4.84</td>
<td>50</td>
</tr>
<tr>
<td>sqrt</td>
<td>54.51</td>
<td>61.03</td>
<td>100</td>
</tr>
<tr>
<td>st</td>
<td>5479.77</td>
<td>5772.84</td>
<td>50,000</td>
</tr>
<tr>
<td>lms</td>
<td>25562.02</td>
<td>26761.40</td>
<td>75,000</td>
</tr>
<tr>
<td>matmult</td>
<td>31477.60</td>
<td>33671.99</td>
<td>100,000</td>
</tr>
</tbody>
</table>

which circumstances, the fixed priority approach will outperform the system load oriented approach. Additionally, we are currently working at integrating the fixed priority approach into event-based systems to perform an improved optimization of systems which are not strictly periodical. In the long run, we are planning on extending the approach to cover scheduler overheads and inter-task dependencies like cache-related preemption delays.

6. REFERENCES