WCET-aware Scheduling Optimizations for Multi-Core Real-Time Systems

Timon Kelter
TU Dortmund University
Otto-Hahn-Strasse 16
44227 Dormund, Germany
Email: timon.kelter@tu-dortmund.de

Hendrik Borghorst
TU Dortmund University
Otto-Hahn-Strasse 16
44227 Dormund, Germany
Email: hendrik.borghorst@tu-dortmund.de

Peter Marwedel
TU Dortmund University
Otto-Hahn-Strasse 16
44227 Dormund, Germany
Email: peter.marwedel@tu-dortmund.de

***Abstract*—In real-time systems, the WCET (worst-case execution time) of tasks is of utmost importance. For multi-cores, the WCET has been shown to be hard to determine due to task interactions on shared memory and shared buses. This problem is usually addressed by spatial or temporal partitioning of the resources, but both lead to lower utilization if the partitioning is not done optimally. We examine two approaches for optimizing resource usage in a temporally partitioned multi-core system and show that these techniques can reduce the WCET by more than 30% on average, leading to better schedulability and higher system utilization.**

I. Introduction

Most of today's high-performance processors are multi-cores, not only in the desktop and server but also in the embedded systems market. Though the increased overall computational power is beneficial to the average-case application, multi-cores pose a fundamental problem for safety-critical real-time applications. Since some of the hardware components in a multi-core system are shared between cores, tasks that execute on different cores may interfere with each other during accesses to shared components. This breaks the isolation between tasks and makes their worst-case execution time (WCET) harder or even impossible to predict. Since the WCET is needed for schedulability analysis and certification of safety-critical systems, the current industrial practice is to deactivate all but a single core to bring the system back into a predictable state [1].

To overcome this unsatisfactory state, it is necessary to know how the shared resources are arbitrated among contending requests from multiple cores. Also, a precise definition of the timing behavior of this arbiter must be given.

Recent publications have discussed the implications of different types of arbitration methods on the achievable analysis precision [2]. Time-triggered arbitration methods were found to be suited best for tight WCET estimation, but their performance is highly dependent on their parameterization and on the structure of the examined programs.

To overcome these problems, we present two novel optimizations that can significantly improve the WCET but also the average-case execution time (ACET) of programs running on timing-predictable multi-cores. The first is an evolutionary optimization of the shared resources' schedule parameters, whereas the second is a multi-core WCET-aware instruction scheduling which re-structures the input programs to increase their performance on a given time-predictable multi-core platform. Both optimizations result in lower WCETs, which in turn leads to improved schedulability and increased resource utilization for multi-core real-time systems.

In Section II we will given on overview on existing approaches and related work and Section III introduces the system model that we use for the experiments. Sections IV and V present the aforementioned novel optimizations and Section VI closes the paper with a summary and directions for future work.

II. Related Work

The standard approach to WCET analysis [3] has recently been extended towards the analysis of multi-core systems [4], [5], [2], which makes it possible for us to consider multi-core WCET as an optimization target.

The optimization of bus schedules has been the topic of a range of previous publications, but the vast majority either is restricted to TDMA schedules or uses ad-hoc WCET computations instead of an analyzer following established design principles [3]. The optimization in [6] and [7] by the same authors is based on search heuristics (simulated annealing) and is similar to our evolutionary optimization in this respect. It also integrates system-wide task scheduling with optimization, but on the other hand, it is restricted to TDMA schedules, whereas we also consider more flexible schedule variants. TDMA slot length allocation is also done in [8], but the employed WCET analysis framework is less precise and it is again restricted to TDMA. Concerning the employed evolutionary variation operators we use a similar approach as [9], but [9] is restricted to TDMA and considers the optimization at a far more coarse-grained level, i.e. the scheduling of tasks as a whole. Finally, [10] also examines bus schedule optimization, but only for the special case of *Harmonic Round-Robin* schedules and for additive WCET models.

The majority of previous publications on WCET-aware instruction scheduling is focused on optimizing the WCET of a *single-core system* [11], [12]. As an exception, [13] discusses several access models for time-predictable multi-cores on an abstract level, but requires manual restructuring of the tasks. In contrast, the instruction scheduler, presented in this paper, can be used to automatically implement these models on a micro-architectural scale.

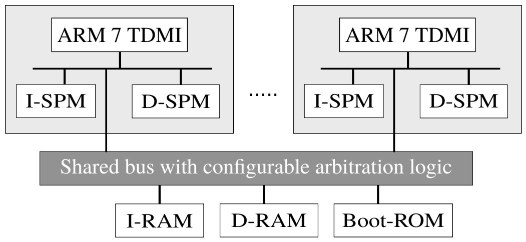


Fig. 1: The employed system model.

To the best of our knowledge, previous work has neither addressed the optimization of real-time bus schedules including TDMA *and* more flexible methods nor the scheduling of instructions according to the requirements of a time-predictable multi-core platform. We will see in the following that especially the consideration of schedule types other than TDMA is important to achieve the highest WCET and ACET gains. This aspect has not been considered in previous publications on real-time system scheduling optimizations.

III. System Model

Our optimizations are based on a system model as shown in Figure 1, containing n highly predictable ARM7TDMI cores. Each of the cores has access to local scratchpad memories and to shared RAM. The shared memory is accessed via a shared bus, which is responsible for arbitrating requests of the cores. Caches can be integrated, but are not considered in the current work, since they decrease the predictability of the system. For the experiments, the whole system including the bus arbiter was implemented in the cycle-accurate simulator CoMET from Synopsys Inc. [14].

An *application* that runs on the platform is a set of n parallel tasks with one task executed on each of the cores. This restriction is due to the limitations of the WCET analysis framework. Once the WCET analysis can handle additional scenarios the optimizations will be immediately applicable to the new scenarios, too. Our results will also at least remain valid for multiple non-interruptible tasks per core, since this is a purely technical extension of the current scenario. The exact WCET of the tasks, denoted $WCET_{real}$, is not computable in general, therefore whenever we refer to WCETs this is equivalent to the *estimated* $WCET_{est} \geq WCET_{real}$, which is a safe upper bound on the $WCET_{real}$ [3]. The WCET of the application is the maximum of the task WCETs, whereas the applications's ACET is the sum of the task ACETs. We also measure the total utilization of the shared bus, which is the number of cycles in which bus transfers were done divided by the application's runtime. ACETs and utilization are always determined by a CoMET simulation, whereas WCETs are computed by static analysis of the application.

The bus schedule dictates the order in which shared memory accesses from the cores are granted during the execution of their assigned tasks. This determines the arbitration delay for individual accesses which in turn contributes to the ACET and WCET of the tasks. Thus the simulation and the WCET analysis must be aware of the bus schedule.

The first schedule type that we consider here is *fair arbitration* (FAIR), where each of the n cores is cyclically given the chance to access the bus. Cores which do not want to access the bus are skipped. This method is also called *Round-Robin* and has no parameters. In *static priority-based arbitration* (PRIO) each core i is assigned a unique priority $p_i \in \{1, \ldots, n\}$ and only the request from the core with the highest priority is granted in case of contending requests.

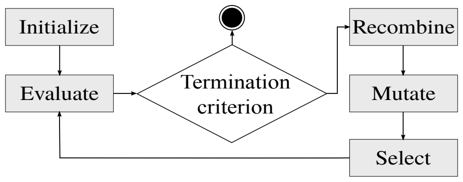


Fig. 2: The structure of the evolutionary bus schedule optimization.

Time division multiple access (TDMA, also known as *time slicing*) is a cyclic schedule type, partitioned into s slots where each slot i is assigned a slot length l_i (measured in bus clock cycles) and an owner core $o_i \in \{0, \ldots, n-1\}$, which has exclusive access during the slot. The last option, *Priority division* (PD), is a generalization of TDMA, where each slot has an owner and each core i gets a priority value $p_i \in \{0, \ldots, n\}$. If the owner does not occupy its slot in the current bus clock cycle, all other cores i with $p_i > 0$ may perform their accesses as in PRIO mode. Details on all schedule types can be found in [2].

For the WCET analysis, both PRIO and FAIR have the property that the arbitration delay can only be bounded when all possible interleavings of all threads from the cores are considered. This would lead to a combinatorial state space explosion during the analysis. Therefore, for FAIR we resort to worst-case assumptions (maximal delay) during the WCET analysis. For PRIO we cannot determine WCET values for all cores other than the core with the highest priority. This means that PRIO will never outperform other schedules in terms of WCET, nevertheless we consider it here, since it might still outperform others in terms of ACET. The time-triggered variants TDMA and PD have the advantage that the analysis can work locally on a per-core basis and determine the arbitration delay only with the help of the information about *when* the access is made (in which slot) and which slots belong to the currently analyzed core. Further details on the analysis framework can be found in [2]. Generally, TDMA lends itself more to WCET analysis, but has a negative impact on utilization and ACET due to cores not fully using their slots. PD produces better utilization values at the cost of less precise WCET estimates. In both cases, the number of slots, the slot lengths and the slot priorities have a major influence on the achieved WCET and ACET performance.

IV. Multi-objective Bus Schedule Optimization

The manual selection of an optimal schedule type and its parameterization for a given application is a hard and error-prone task. Therefore, in this section we present a multi-objective evolutionary search algorithm which automatically determines a range of well-suited schedules for an application and enables users to choose a solution which balances WCET, ACET and utilization according to their needs.

The structure of the optimization is depicted in Figure 2. It starts with a set of initial schedules. In the evolutionary

t	s	$\vec{p} = (p_0, \ldots, p_{63})$	$\vec{l} = (l_0, \ldots, l_{63})$	$\vec{o} = (o_0, \ldots, o_{63})$

Fig. 3: The evolutionary algorithm's genome.

optimization context these are also called *individuals*. For all individuals, the WCET, ACET and bus utilization values are determined. Then, promising individuals are recombined and mutated with a certain probability. After these steps, the optimizer selects those individuals that should survive into the next generation and the optimization continues with them. The steps are repeated until a user-definable termination criterion is met.

Individuals are represented by the genome shown in Figure 3. It contains the scheduling policy t (one of FAIR, PRIO, TDMA or PD), the number of slots s and the vectors of priorities, slot lengths and slot owners ($\vec{p}$, $\vec{l}$ and $\vec{o}$). For an efficient recombination and mutation, the genome needs a fixed length, therefore each vector is limited to 64 entries, which limits the solutions space to 64 slots. Since we will examine systems with up to 8 cores, this is a reasonably large range.

Initialization

The optimization process starts with a set of schedule candidates also called the *population*. It contains a FAIR schedule, a uniform PRIO schedule, a uniform TDMA schedule and a uniform PD schedule. "Uniform" here means that all cores get one slot and all slot lengths are set to the minimum allowed size, since from our experience this reduces the bus arbitration delay. Slot priorities are distributed such that the cores get a priority equal to their core ID.

The rest of the population is filled up with candidates for which the parameters are randomly chosen according to a uniform distribution. The randomness is needed to appropriately cover the search space and is a standard approach in evolutionary optimization.

Recombination and Mutation

Similar to [9] we use *arithmetic operators* which do not treat the genome as a bit string and flip individual bits, but which perform arithmetic operations on the contained parameter values. This is done to limit the degree of randomness in the optimization, since otherwise flipping a high-order bit of a parameter might cause the optimization to unguidedly "jump around" in the parameter space.

The recombination works piecewise on two genomes, with a multi-point crossover. That is, during the recombination of A and B, for each segment $\sigma \in \{t, s, \vec{p}, \vec{l}, \vec{o}\}$ from Figure 3 a recombination point $r \in \{0, \ldots, l_\sigma\}$ is determined randomly with uniform distribution, where l_σ is the length of σ. The new segment σ^C for the resulting individual C is then given as the concatenation of the substrings $\sigma^A_{[0:r)}$ and $\sigma^B_{[r:l_\sigma)}$. l_σ always denotes the *effective* length of the segment, e.g. we may have up to 64 slots, but if A and B only use 7 slots at maximum, then $l_{\vec{o}} = 7$.

The mutation is also only applied for parameters within the effective lengths and mutates each segment's values with probability 0.3. To restrict the step size, we use δ-mutation, where a new value v_{new} is randomly chosen from $[v_{old} - \delta, v_{old} + \delta]$. For the number of slots s the value of δ is 5, for the slot lengths $l \in \vec{l}$ we chose $\delta = 30$.

Finally, we perform a randomized "genome repair" step, which mutates the individual until each core has a unique priority and each core is the owner of at least one slot. The first is a requirement of our platform, whereas the latter is needed to avoid core starvation and thus infinite WCET values. The intention behind all of these design decisions is to increase the chances that we find good solutions early, since the objective evaluation and thus each new generation is costly in our scenario.

Experimental Results

We implemented the evolutionary optimization with the PISA framework [15], using the SPEA2 selector [16], which is used to determine individuals for mutation, recombination and generation survival. SPEA2 tries to keep individuals in the population that are not *pareto-dominated* by others, i.e. for which no other individual exists which is better in all objective values. In addition SPEA2 maintains a solution density to increase the diversity of the generated solutions.

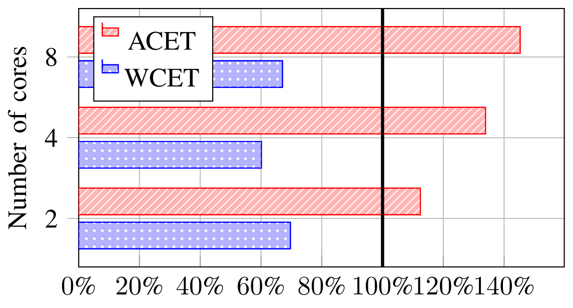
For the WCET analysis we had access to the analyzer from [2] and the ACET/utilization values were determined with the CoMET cycle-true virtual platform simulator [14].

The tasks we used for the tests come from the publicly available benchmark suites UTDSP, MRTC, MiBench and MediaBench, covering application domains such as signal and image processing, mathematical and control applications. In total we used 110 applications each consisting of 2 to 8 tasks depending on the analyzed system. The tasks are single-threaded but their input and output is read from and written to the shared memory. Thus only the I/O operations issued by the tasks are subject to bus arbitration, the tasks' code and local data are stored in the scratchpads of the cores, denoted *I-SPM* and *D-SPM* in Figure 1.

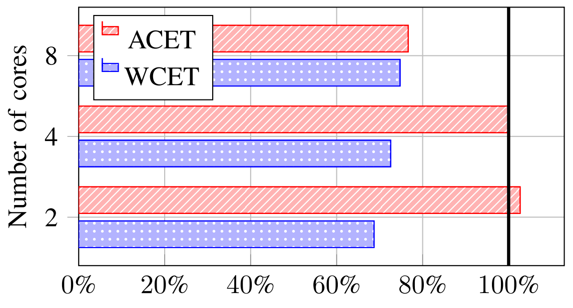
We used a generation size of 20 individuals and a minimum number of 20 generations. After the 20th generation, optimization is continued if the current generation is at least 0.05% better in any objective than the previous one. This was added to provide confidence that we do not abort the optimization prematurely, but we encountered no cases where the 21st generation was actually reached.

We present relative results in the following, where the first meaningful baseline is the FAIR individual as this represents the current practice in many real-world systems. Figure 4a shows the geometrical mean of the relative WCET (ACET) of the final-generation individuals with the best WCET, relative to the WCET (ACET) value of the FAIR individual from the first generation. Since the FAIR individual has no parameters and thus never evolves, it does not matter from which generation it is taken. It can be seen that the reduction in WCET of up to 39% in the case with 4 cores is accompanied by an increase in ACET. This is plausible, because most of the best-WCET individuals are using TDMA or PD (see Figure 6), which have better WCET, but worse ACET performance.

As the second baseline, we chose the uniform TDMA schedule with minimum slot length, which usually produces



(a) Baseline (100%): FAIR individual



(b) Baseline (100%): Uniform TDMA individual

Fig. 4: Average results for the best-WCET individuals.

good WCET values. Here, the question is whether the optimization can still improve upon this baseline. As can be seen in Figure 4b we can still observe WCET improvements of 31% (2 cores) to 25% (8 cores) without significant loss of ACET performance. Also note that Figure 4 contains the results for the individual with the *best WCET*. Thus, if we want to balance ACET and WCET, the evolutionary approach also delivers matching solutions, some of which are presented in the following.

Figure 4 also indicates that average-case and worst-case performance are not necessarily correlated, which motivates our approach of explicitly considering multi-objective and WCET-oriented optimizations.

To show the distribution of the results among the benchmarks, Figure 5 shows the detailed WCET results for all benchmarks in the 2-core configuration. Each segment in the figure represents the best-WCET individual for one benchmark, which is identified by its benchmark ID, shown on the x axis. All WCETs are relative to the WCET of the uniform-TDMA individual, which was also taken as the comparison base in Figure 4b. It is visible, that the average WCET reduction is achieved by a very even distribution of WCET reductions. The few benchmarks which experience WCET reductions larger than 50% are unbalanced examples, where one task with the biggest WCET needs much more bus bandwidth than the others, and thus its runtime can be drastically decreased by assigning more or longer slots to it.

The best-WCET individuals constitute of TDMA schedules with adapted slot lengths, of even more customized PD schedules and of some FAIR schedules. The distribution of the schedule types is depicted in Figure 6. Note that in the worst-case a FAIR access may have to wait for at most one access from all other cores. A TDMA access that is issued too late in the issuer's slot to finish inside that slot may have

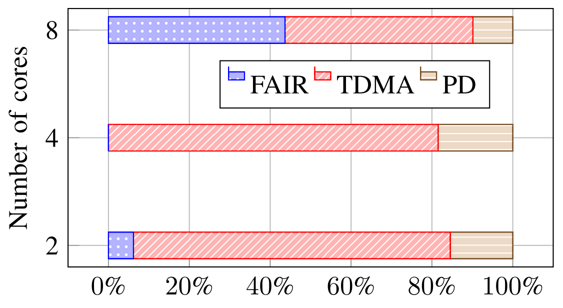


Fig. 6: Distribution of different schedule types among best-WCET individuals.

to wait for the rest of the slot *plus* the slots of all other cores. Due to this, for tasks on which the TDMA WCET analysis fails to produce precise results, FAIR can be better than TDMA. Apparently, this mostly happens for the platform with 8 cores. This platform requires a longer TDMA schedule, to still provide at least one slot per core and it seems that the WCET analysis gets more imprecise with growing schedule length. Nevertheless it works well for most examples, making TDMA the predominant schedule type among the best-WCET individuals.

Concerning the precision of the WCET results, for the example of the best-WCET individuals, our analyzer produces WCET estimates that are 27% (88%, 181%) higher than the measured ACET in the configuration with 2 (4, 8) cores, and thus can be assumed to be reasonably precise.

The fact that the optimization performs better for systems with fewer cores can also be explained when examining the baseline utilization of the shared bus. For 2 cores, the applications have an average bus load of 21%, which rises to 41% for 4 cores and 64% for 8 cores, measured under FAIR scheduling. Thus, all attempts to increase the utilization are ultimately limited by the amount of unused bus time, which is decreasing as the number of cores increases.

The development of the individuals during a single optimization run is illustrated in Figure 7 which shows the WCET, ACET and utilization for *all* individuals that were evaluated in the course of the optimization of an 8-core benchmark containing mixed multimedia and control tasks (ADPCM en-/decoder, Huffman encoder, FFT, FIR filter, edge detection, sorting algorithms). WCET and ACET are shown on the x and y axis, whereas the color of the marks indicates their utilization, as shown in the color bar under the Figure. The axes are scaled logarithmically, to accommodate the spread of the results.

The PD individuals all show a good ACET performance, but vary in their WCET by more than one order of magnitude depending on the configuration. In contrast, the TDMA individuals stringently have a worse ACET performance which is compensated by a bigger span of WCET values - they provide both the best and the worst WCET values. The utilization is directly proportional to the ACET, which confirms our expectations that higher utilization implies lower average bus access delays.

The pareto-optimal points are represented by the blank symbols on the left side of the figure and are also listed in

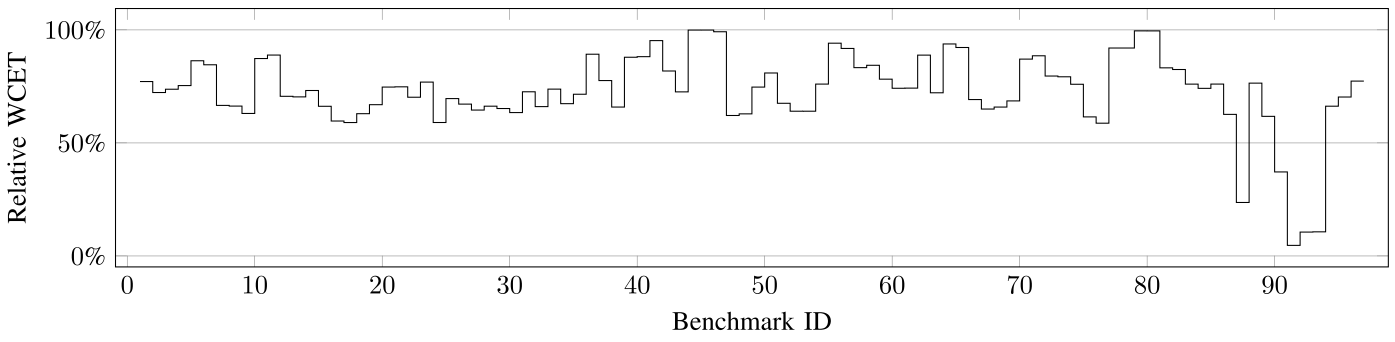


Fig. 5: Detailed results for the best-WCET individuals on the 2-core platform (Baseline: Uniform TDMA individual).

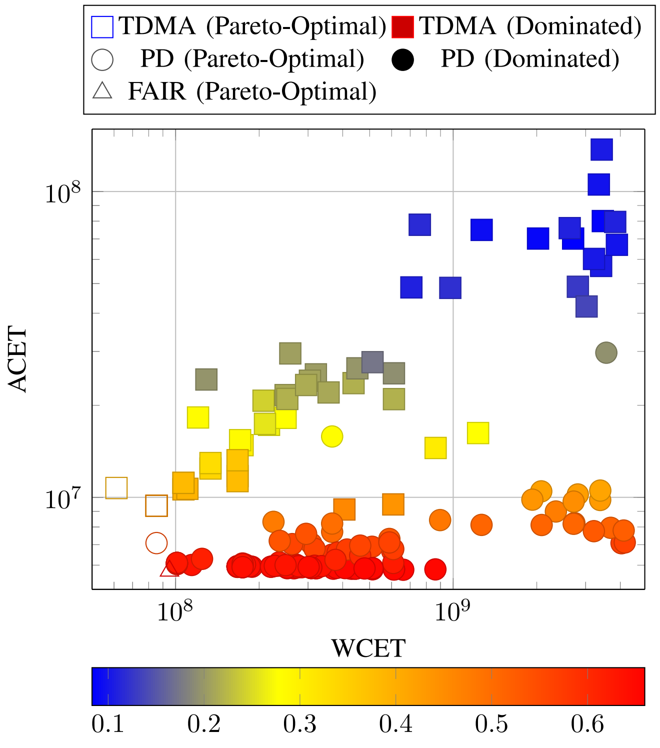


Fig. 7: Exemplary population with marked Pareto-Front for a benchmark with 8 cores.

detail in Table I together with their slot length, owner and priority vectors $\vec{l}$, $\vec{o}$ and $\vec{p}$. As can be seen, FAIR produces the best utilization and ACET values (the triangle in Figure 7) and TDMA has the best WCET value (the blank squares in Figure 7). In between we find TDMA and PD configurations, where, in this case, PD provides a significantly enhanced ACET without loss of precision at the WCET side (blank circle in Figure 7). This distribution of results is typical and could be observed for most benchmarks.

The evolutionary optimization of a single application takes 3 to 4 hours on average, depending on the number of analyzed cores. Taking into account that even the evaluation of a single configuration takes minutes on average, and that almost all of the time is spent on the WCET analysis (59%) and the CoMET simulation (36%), this is still reasonable for e.g. nightly builds of a software. Also, the optimization itself is trivially parallelizable, which we have not done here. The WCET determination is time-consuming since we use an abstract interpretation based analysis [2] which is also used in commercial WCET analyzers like aiT [3]. Usage of analysis techniques of lower runtime complexity [13] would result in a reduced precision. The WCET analysis and simulation runtime will scale linearly with the number of cores, but the total runtime of the optimization until "good" solutions are found might grow faster than linear since more parameters will have to be explored.

All in all, we have seen that FAIR arbitration is strong on producing good ACET values, and that it can even outperform more predictable arbitration schemes especially when the minimum schedule length increases as e.g. for systems with rising core numbers. TDMA has proven to be the best choice for WCET. Still, for TDMA as well as for PD an optimization of the schedule parameters is highly desirable and may lead to WCET improvements of more than 30%. PD can be used to balance ACET and WCET which again is easier to do in an automated way.

V. WCET-aware Instruction Scheduling

In Section IV we have examined the possibilities of adjusting the bus schedule parameters to the given task set. Of course, in practice, we have another degree of freedom, namely to reorder the instructions inside the tasks to match the bus schedule. Since both optimizations are interdependent we perform the instruction reordering for every single individual that is generated during the algorithm from Section IV. Thus we will also find solutions in which the bus schedule only excels when combined with a custom instruction schedule. The instruction reordering can also be invoked separately, for any user-defined schedule.

We build upon a classical *list scheduler* [17], which divides the task into sequential *regions* and schedules each of those separately. The simplest choice for such regions are *basic blocks*, i.e. maximal sequences of instructions which can only be entered at the first and only be exited at the last one. The instructions of these regions are re-ordered by our scheduler, which maintains a list of dependencies and a set of instructions which are *ready* for execution, i.e. whose dependencies have been fulfilled. Our task is to assign a *priority* to them, and the scheduler will then select one of the instructions with highest priority and append it to the result order. This process continues in the same manner until all instructions of the region have been scheduled. Note that this is a compile-time optimization, there is no runtime scheduling involved.

Bus Mode	s	WCET	ACET	Utilization	$\vec{l}$	$\vec{o}$	$\vec{p}$
FAIR	-	95076900	5752270	0.659636	-	-	-
PD	8	85379400	7085960	0.54958	$(3, 3, 3, 3, 3, 3, 3, 3)$	$(2, 1, 7, 5, 4, 0, 6, 3)$	$(7, 3, 1, 4, 5, 6, 8, 2)$
TDMA	8	61227900	10725600	0.387001	$(10, 3, 3, 3, 3, 3, 3, 3)$	$(2, 3, 1, 0, 4, 5, 6, 7)$	-
TDMA	8	85379400	9383380	0.442147	$(3, 3, 3, 3, 3, 3, 3, 3)$	$(0, 1, 2, 3, 4, 5, 6, 7)$	-

TABLE I: Details on the pareto-optimal individuals from Figure 7.

Scheduling Heuristics

In the following we present two novel priority assignment heuristics that are tailored towards the optimization of the WCET of tasks running on time-predictable multi-cores.

For the optimization of task t running on core c, the *slot length heuristic* (SL) first determines the length l_c^{max} of the longest slot which is assigned to c. During the scheduling of a region, it maintains a counter l_c^{cur} which is set to 0 at the start of the region. With the help of the maximum bus access duration t^{max} the priority[1] of instruction $i \in I$ is given by the priority function p_{SL} as

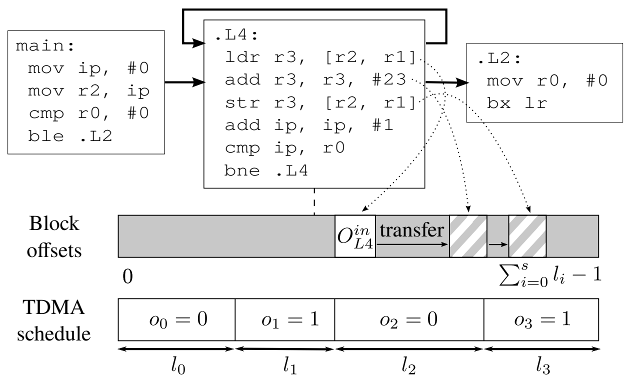
$$p_{\text{SL}}(i) = \begin{cases} 1 & \text{if } bac(i) \wedge (l_c^{cur} + t^{max}) \leq l_c^{max} \\ 0 & \text{else} \end{cases} \tag{1}$$

where $bac : I \rightarrow \{true, false\}$ determines whether an instruction will possibly access the shared bus. After an instruction i with $bac(i) = true$ was scheduled, l_c^{cur} is incremented by t^{max}, otherwise $l_c^{cur} = 0$. The intention is to bundle bus-accesses to packages which fit into the slots of the core.

The second heuristic, called *offset heuristic* (OF), is based on the same idea, but uses detailed information that is extracted directly from the WCET analysis results. For each basic block b the WCET analysis computes an incoming set of *offsets* O_b^{in}, which specify that whenever b is entered during the execution of the task, the position within the periodic TDMA schedule is *guaranteed* to be contained in O_b^{in} [4].

This is illustrated in Figure 8 which shows an example of a task's control-flow graph in the upper half. For this example, we assume a system with 2 cores, where the presented task is executed on core 0. The TDMA bus schedule consists of 4 slots, whose length and owner cores are depicted in the bottom of the Figure. Below the graph, the set O_b^{in} for block `L4` is shown, which is a subset of the full TDMA offset span, marked in gray. Thus, in this example the analysis has determined that the load instruction `ldr` at the head of block `L4`, which accesses the bus, will always start its execution from one of the offsets contained in the white rectangle marked in the schedule. Considering the schedule, we know that this area is contained in slot 2, which is owned by core 0 and thus the access will be granted immediately. After the `ldr` instruction was analyzed, the analysis will compute new offsets which reflect the positions in the schedule at which the execution of the next instruction will start. This step is called the *transfer* step of the analysis and will also be used by our optimization to dynamically update the current offsets. The striped areas in Figure 8 represent the results of two applications of the transfer function. The dotted arrows indicate which offset information belongs to which instruction. In this case, the optimization can decide that after the execution of the first `add`, the following `str` has to wait for the bus in slot 3 and thus can prefer to schedule the second `add` and `cmp` first.

[1]Higher values indicate higher priority.



```
main:
  mov ip, #0
  mov r2, ip
  cmp r0, #0
  ble .L2
```

```
.L4:
  ldr r3, [r2, r1]
  add r3, r3, #23
  str r3, [r2, r1]
  add ip, ip, #1
  cmp ip, r0
  bne .L4
```

```
.L2:
  mov r0, #0
  bx lr
```

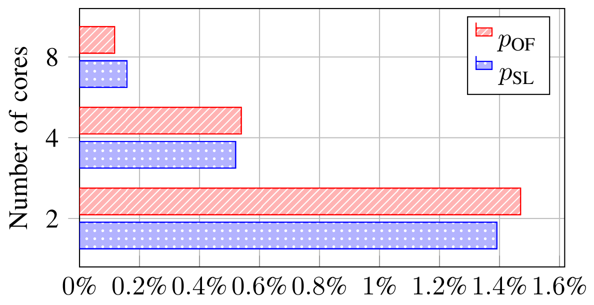
Fig. 8: An example for offset results as computed during the multi-core WCET analysis.

In the worst case the analysis can not infer any useful information which means $O_b^{in} = \{0, \ldots, \sum_{i=0}^{s} l_i - 1\}$. The computation of the O_b^{in} results is interwoven with the modeling of the pipeline and the rest of the memory hierarchy, since all of these components influence the timing behavior of the task and the offsets are just a compressed representation of time after all. The construction of the analysis framework is out of the scope of this paper, we merely use the results here, but further details can be found in [2].

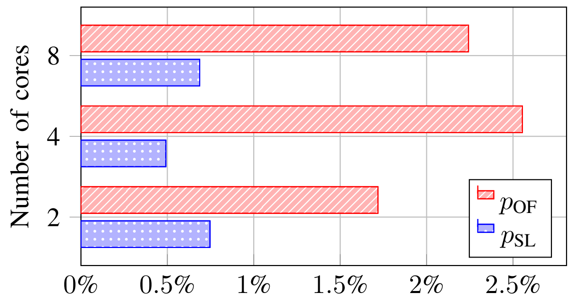
Our optimization has direct access to the sets O_b^{in} for all blocks b and uses those to position the instructions in the region to schedule. For this purpose, it creates a copy O_r^{copy} of the offsets $O_{b_{head}}^{in}$ where b_{head} is the head block of the current scheduling region r. Each time that a new instruction i is scheduled, O_r^{copy} is updated to reflect the time that passes until the processing of i has finished. For this purpose we use the transfer function from the WCET analysis itself, which includes a detailed pipeline and value analysis and thus maintains a high degree of precision. Therefore, at each instant O_r^{copy} contains exactly those offsets at which the next instruction i will be executed. With this information, we can determine whether i is guaranteed to be granted the bus or not, and define the priority function p_{OF} as

$$p_{\text{OF}}(i) = \begin{cases} 2 & \text{if } bac(i) \wedge O_r^{copy} \subseteq \omega_c \\ 1 & \text{if } \neg bac(i) \\ 0 & \text{else} \end{cases} \tag{2}$$

ω_c is the set of offsets at which core c owns the bus, e.g. in Figure 8 for core 0 these would be the offsets in slots 0 and 2. The idea is to force the immediate scheduling of a bus-accessing instruction when we know for sure that the access will be granted (case 1), and to delay them if possible when the access will not be granted (case 3). All instructions which do not require the bus use a default priority (case 2).



(a) Average WCET reductions for uniform TDMA with slot length of 3 cycles.



(b) Average WCET reductions for uniform TDMA with slot length of 12 cycles.

Fig. 9: Average results per platform for scheduling with the slot length heuristic (p_{SL}) and offset heuristic (p_{OF}).

Experimental Results

To evaluate the effectiveness of the heuristics, we tested the scheduler on the same benchmarks already presented in Section IV. We first evaluated scheduling at the basic block level, thus "regions" are "basic blocks" in the following. The basic blocks consisted of 1 to 661 instructions (average: 5.25), and 11.42% of those were accessing the bus on average. In total, the benchmarks contained 82,133 instructions.

To give an impression on the distribution of task sizes used in the evaluation, Table II lists the the lines of code (LOC) counted without comments, the number of loops and the average loop iteration bound. Some of the tasks are available for different input formats and the multimedia tasks have a separated encoder / decoder part, which are not contained in Table II. Taking these variations into account we used 110 different tasks in total. To form parallel applications, these tasks were grouped together in bundles with similar task runtimes.

In Figure 9a the average results for the scheduling are shown for a uniform TDMA schedule, where each core has one slot of length 3 cycles. Both heuristics perform equally well in this setting. This may be due to the fact that in this setting, only one bus access fits into each TDMA slot since the maximum bus access duration is also 3 cycles. Therefore, the schedule is short and it is sufficient to keep the bus accesses isolated, which both heuristics are capable of.

To test the sensitivity of the optimization w.r.t. different schedule configurations we also tested it on a uniform TDMA schedule with a slot length of 12 cycles for which the results are shown in Figure 9b. Here, the WCET reduction achieved by p_{OF} is up to 4 times higher than the reduction for p_{SL}.

Benchmark	LOC	Loops	∅ LB
adpcm	890	14	305
adpcm_g721	1336	22	9
anagram	440	23	341
basicmath_small	1001	11	164
binarysearch	35	1	4
bitcount	202	4	14
bsort100	54	3	99
cjpeg_jpeg6b_transupp	1571	56	9
cjpeg_jpeg6b_wrbmp	1246	5	262
codecs_codrle1	110	4	69
codecs_dcodhuff	221	11	164
compress	603	12	11
convolution	27	2	16
countnegative	73	4	20
crc	68	3	102
dijkstra	119	5	292
dot_product	22	1	2
duff	44	1	100
edge_detect	110	10	77
edn	204	12	53
epic	994	44	296
fdct	143	2	8
fft	311	9	686
fir	225	2	18
fir2dim	81	13	5
g721.marcuslee	106	1	2407
g721_encode	899	9	33
g723_encode	897	9	33
gsm	2394	58	66
gsm_encode	1951	46	53
h263	926	7	205
h264dec_ldecode_block	1567	28	8
hamming_window	62	3	153
histogram	36	6	128
iir	73	3	44
insertsort	61	2	9
jfdctint	218	3	26
latnrm	73	3	42
lcdnum	62	1	10
lms	292	16	50
lpc	328	23	80
ludcmp	86	11	5
matmult	57	5	30
minver	158	17	2
mult	32	3	10
n_updates	38	2	16
ndes	407	12	19
petrinet	500	1	2
pm	558	28	55
qmf	72	2	2005
qurt	88	1	19
real_update	24	0	0
rijndael	1999	30	3126
searchmultiarray	484	4	5
select	62	4	8
selection_sort	67	2	299
spectral	623	12	38
sqrt	45	2	12
st	106	5	803
startup_fixed	99	6	28
statemate	1047	1	100
test3	3985	121	4
v32.modem	1469	16	127

TABLE II: Task details.

The drawback is, that the absolute WCET values for the 12-cycle configuration are 25% (2 cores) to 158% (8 cores) worse than those for the 3-cycle one. It is a general observation in our experiments, that bigger TDMA slots lead to worse WCET and utilization values, which defeats the value of the increased optimization potential in these configurations.

Since the scheduler works on the micro-architectural level, it cannot be expected to have as much impact as the macroscopic schedule parameter optimization, presented in Section IV. To illustrate the results for the individual benchmarks, Figure 10 lists the 20 highest WCET reductions from the results shown in Figure 9a. In this range we observe an

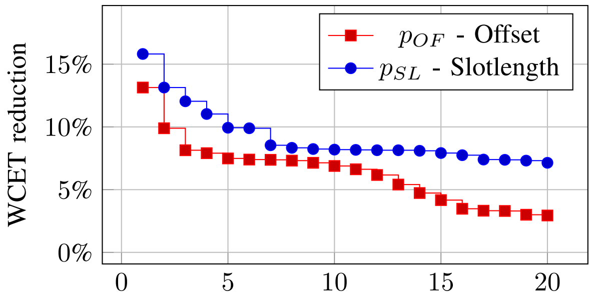


Fig. 10: Relative WCET results for the best 20 benchmarks from Figure 9a per scheduling method.

increased average bus utilization of 14% and 4.4 instructions per region. The WCET reductions for the individual benchmarks range from 13.2% to 2.5% (p_{OF}) or 15.8% to 7.2% (p_{SL}), respectively. Therefore, though the results are lower on average, we still have many benchmarks for which the scheduler achieves significant gains with both heuristics.

The compilation times have tripled compared to the compilation without the WCET-aware scheduling, but again, this is mostly due to the runtime of the WCET analyses.

We also extended the optimization to work on trace and superblock regions. Both are well-known methods to increase the scheduling flexibility, but they come at the cost of inserting compensation code which can adversely affect the WCET. In our experiments, the average gains obtained with both trace and superblock scheduling were lower than those with pure basic block scheduling. This also follows from the observation that the I/O operations which access the shared bus often have data dependencies to their neighbors and thus can rarely make use of the increased trace and superblock region size.

Finally, we also tested the combination of the evolutionary bus optimization from Section IV and the instruction scheduler. As stated, the instruction scheduler was invoked for every generated individual to also find solutions which are only accessible through a combination of bus and instruction scheduling. Apparently, such cases are very rare, since the additional WCET gain over the best-WCET results from Figure 4 is almost identical to the results shown in Figure 9a. This suggests the conclusion, that optimizing the bus schedule first and performing the instruction scheduling afterwards is sufficient in practice, leading to a combined *average* WCET reduction of to up to 33%.

VI. Conclusions

We have presented the first WCET-aware multi-core schedule optimization which takes into account fair, TDMA and priority-division schedules. Our results on a state-of-the-art multi-core WCET analyzer show, that we can reduce the WCET of real-world benchmarks by more than 30% on average. We have shown how ACET, WCET and bus utilization evolve under different parameterizations of the three schedule types. In addition, we have seen that TDMA is *not* always the best choice for minimizing the WCET, which was a basic assumption in previous publications. We have complemented this macroscopic approach with a new type of instruction scheduling heuristic tailored towards multi-core WCET reduction, which can further reduce the WCET by up to 15.8%. In summary, both optimizations significantly increase the precision of the estimated WCETs and thus the usability of multi-core WCET analysis.

Acknowledgments

This work was partially supported by EU COST Action IC1202: Timing Analysis On Code-Level (TACLe). The authors would also like to thank Synopsys for the provision of the virtual prototyping IDE CoMET.

References

[1] B. C. Ward, J. L. Herman, C. J. Kenna, and J. H. Anderson, "Making Shared Caches More Predictable on Multicore Platforms," in *Euromicro Conference on Real-Time Systems*, 2013.

[2] T. Kelter, T. Harde, P. Marwedel, and H. Falk, "Evaluation of Resource Arbitration Methods for Multi-Core Real-Time Systems," in *International Workshop on Worst-Case Execution Time Analysis*, July 2013.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, 2008.

[4] T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury, "Static Analysis of Multi-Core TDMA Resource Arbitration Delays," *Real-Time Systems*, 2013.

[5] S. Chattopadhyay, C. Kee, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A Unified WCET Analysis Framework for Multi-Core Platforms," in *Real-Time and Embedded Technology and Applications Symposium*, 2012.

[6] J. Rosen, C.-F. Neikter, P. Eles, Z. Peng, P. Burgio, and L. Benini, "Bus Access Design for Combined Worst and Average Case Execution Time Optimization of Predictable Real-Time Applications on Multiprocessor Systems-on-Chip." in *Real-Time and Embedded Technology and Applications Symposium*, 2011.

[7] A. Andrei, P. Eles, Z. Peng, and J. Rosen, "Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip," in *International Conference on VLSI Design*, January 2008.

[8] E. Wandeler and L. Thiele, "Optimal TDMA Time Slot and Cycle Length Allocation for Hard Real-Time Systems," in *Asia and South Pacific Design Automation Conference*, 2006.

[9] A. Hamann and R. Ernst, "TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques," in *In Proc. of the IEEE/ACM Design, Automation and Test in Europe Conference*, 2005.

[10] M.-K. Yoon, J.-E. Kim, and L. Sha, "Optimizing Tunable Wcet with Shared Resource Allocation and Arbitration in Hard Real-Time Multicore Systems," in *Real-Time Systems Symposium*, 2011.

[11] W. Zhao, W. Kreahling, D. Whalley, C. Healy, and F. Mueller, "Improving WCET by Applying Worst-Case Path Optimizations," *Real-Time Systems*, vol. 34, no. 2, 2006.

[12] Y. Huang, M. Zhao, and C. J. Xue, "WCET-aware Re-Scheduling Register Allocation for Real-Time Embedded Systems with Clustered VLIW Architecture," in *International Conference on Languages, Compilers, Tools and Theory for Embedded Systems*, 2012.

[13] A. Schranzhofer, R. Pellizzoni, J.-J. Chen, L. Thiele, and M. Caccamo, "Worst-Case Response Time Analysis of Resource Access Models in Multi-Core Systems," in *Design Automation Conference*, 2010.

[14] Synopsys Inc., "CoMET System Engineering IDE," http://www.synopsys.com.

[15] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA — A Platform and Programming Language Independent Interface for Search Algorithms," in *Evolutionary Multi-Criterion Optimization*. Springer, 2003.

[16] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization," in *EUROGEN2001 Conference*, 2002.

[17] A. Aho, M. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques and Tools*, 2nd ed. Addison Wesley, 2006.