Real-Time Systems manuscript No.
(will be inserted by the editor)

Static Analysis of Multi-Core TDMA Resource Arbitration Delays

Timon Kelter · Heiko Falk · Peter Marwedel · Sudipta Chattopadhyay · Abhik Roychoudhury

Received: date / Accepted: date

Abstract In the development of hard real-time systems, knowledge of the *Worst-Case Execution Time* (WCET) is needed to guarantee the safety of a system. For single-core systems, static analyses have been developed which are able to derive guaranteed bounds on a program's WCET. Unfortunately, these analyses cannot directly be applied to multi-core scenarios, where the different cores may interfere with each other during the access to shared resources like for example shared buses or memories. For the arbitration of such resources, *TDMA arbitration* has been shown to exhibit favorable timing predictability properties. In this article, we review and extend a methodology for analyzing access delays for TDMA-arbitrated resources. Formal proofs of the correctness of these methods are given and a thorough experimental evaluation is carried out, where the presented techniques are compared to preexisting ones on an extensive set of real-world benchmarks for different classes of analyzed systems.

Keywords WCET · TDMA arbitration · Multi-Core · Shared Resources · Worst-Case Analysis · Static Program Analysis

Timon Kelter, Peter Marwedel
TU Dortmund, Otto-Hahn-Straße 16, 44227 Dortmund, Germany
Tel.: +49-231-755-6133
Fax: +49-231-755-6116
E-mail: {timon.kelter,peter.marwedel}@tu-dortmund.de

Heiko Falk
Ulm University, Albert-Einstein-Allee 11, 89081 Ulm, Germany
Tel.: +49-731-50-24183
Fax: +49-731-50-24182
E-mail: heiko.falk@uni-ulm.de

Sudipta Chattopadhyay, Abhik Roychoudhury
National University of Singapore, 13 Computing Drive, Singapore 117417, Republic of Singapore
Tel.: +65-6516-8939
Fax: +65-6779-4580
E-mail: {sudiptac,abhik}@comp.nus.edu.sg

1 Introduction

With the rising importance of multi-core systems in the processor market, including the embedded systems or cyber-physical domain, there is a growing need for tools to verify the timing behavior of such systems, and as such the WCET. For embedded systems, this may be the most important metric, because they often must work under real-time conditions where a response must be delivered in a predefined time. Therefore, fine-grained WCET analyses have been developed for single-core systems in the last decade [Wilhelm et al (2008)], resulting in a variety of commercially available tools. In contrast, only first proposals exist for multi-cores. One of the major difficulties in analyzing the WCET for multi-core platforms is that programs running on different cores may interfere with each other, for example during accesses to a common shared bus which connects the cores to a shared main memory. A possible approach to resolve these interferences is to implement a *Time Division Multiple Access* (TDMA) bus arbitration protocol which assigns a fixed-length time slot to each core in round robin fashion. The TDMA arbitration scheme allows to derive a simple upper bound for the bus access delay which can be incurred by a single access. As we will show, this bound is only a rough overestimation. In [Kelter et al (2011)], a new type of analysis was presented which safely bounds the access time for TDMA-arbitrated resources with high precision and moderate analysis times. In this article we provide

- A detailed review of the method from [Kelter et al (2011)]
- Formal proofs of its correctness
- Introduction of *TDMA-composability* which is a property of a system that will allow us to analyze systems with short schedules and high variabilities which can be hardly analyzed using the unmodified method from [Kelter et al (2011)]
- A detailed evaluation on real-world benchmarks

The first approaches to multi-core WCET analysis only modeled the shared resources to some extent. [Suhendra and Mitra (2008)] and [Zhang and Yan (2009)] analyzed the effects of a shared L2 cache without considering the interference on a shared bus that is used to access the shared cache. [Zhang and Yan (2009)] provides a bound on the number of additional cache misses due to the inter-core interference, whereas [Suhendra and Mitra (2008)] eliminates the interference altogether by exploring different scenarios of locking and partitioning the shared cache. A similar approach is pursued by [Hardy et al (2009)], where cache bypassing is used to eliminate the cache conflicts between different cores.

[Gustavsson et al (2010)] investigates a totally different approach, where the whole multi-core system is modeled as a set of timed automata. The WCET is obtained by proving special predicates through model checking. This approach allows for a detailed system modeling, but does not scale very well as all system states have to be explored in the course of the WCET analysis, leading to a state explosion. [Lv et al (2010)] enhances this approach by combining model checking with abstract interpretation of cache states to increase the performance of the proposed analysis.

For analyses that include the shared bus, the choice of the bus arbitration method is crucial. [Pitter and Schoeberl (2010)] compared the predominant arbitration meth-

ods and TDMA arbitration resulted as the most predictable method, because it allows to derive an upper bound on the delay that any access to the shared resource may incur until it is granted. Unfortunately this bound is not tight in general. To provide a better access time estimation, [Andrei et al (2008)] tries to determine the precise time at which every single memory access takes place. The bus access delay estimation is then performed separately for each access. The main problem is, that accesses in loops with an iteration count of i can potentially have i different access times associated to the same memory access. Therefore, the analysis has to unroll all loops virtually to determine the access times for each access individually, which makes the analysis runtime dependent on the loop iteration counts.

[Chattopadhyay et al (2010)] circumvents this costly unrolling by aligning each loop head execution to the first TDMA slot during the analysis. However, this artificial alignment of each loop iteration results in an additional penalty term to be added in WCET estimation. Therefore, the analysis proposed in [Chattopadhyay et al (2010)] is far more efficient but also less precise than [Andrei et al (2008)]. We have introduced a new type of analysis in [Kelter et al (2011)] which is a compromise between precision ([Andrei et al (2008)]) and analysis speed ([Chattopadhyay et al (2010)]). In this article we will review and extend this new analysis and show that it is even able to outperform [Andrei et al (2008)] in terms of precision in some cases.

Finally, [Pellizzoni et al (2010)] derives the worst-case bus delays in a multi-core system analytically with the help of memory traffic arrival curves. This approach is different from ours since we do not require such curves.

A different direction in timing analysis is the adaption of multi-core hardware to exhibit better predictability properties. [Paolieri et al (2009)] proposed a multi-core architecture in which safe measurements of the WCET are possible in a dedicated processor mode, and [Mische et al (2010)] developed a superscalar SMT processor, in which a single hard real-time task is able to execute with no interference, e.g. with exactly the same timing as in a single-core processor. The rest of the CPU time is then assigned to the remaining (soft real-time) tasks. These approaches are orthogonal to ours since we focus on estimating the WCET of tasks on existing hardware platforms.

The rest of this article is organized as follows: Section 2 introduces the system model used in the analyses and Sections 3 and 4 introduce the overall analysis framework as well as the general analysis concepts, respectively. Section 5 presents the analyses and their correctness proofs and introduces TDMA-composability as a new composability class to support the timing analysis of multi-cores. The evaluation of the analyses for different composability classes is given in Section 6. Finally, we provide a summary of our results and give directions for future work in Section 7.

2 System and Application Model

In the following, we present the model of the hardware and software of the system that we want to analyze. We state which prerequisites are needed for the analyses and which application scenarios are covered.

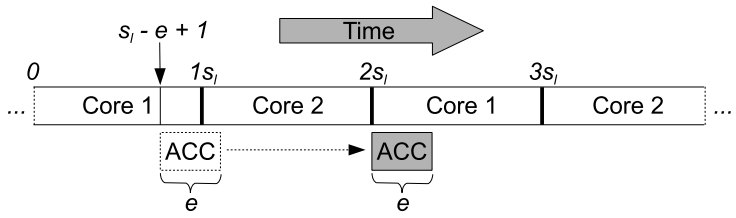


Fig. 1 An example for a bus access which is maximally delayed.

2.1 Modeled Hardware

We assume a system architecture where $n_c \geq 2$ cores are present in a single processor. Each of the cores is working on a constant frequency and has an in-order pipeline and a private L1-Cache. All the cores are connected to a shared TDMA-arbitrated memory bus with a uniform TDMA slot size of s_l cycles per core. The bus is used to access a shared L2-Cache, which itself is linked to the main memory. The bus, the L2 cache and the main memory may be located on-chip or off-chip.

Definition 1 In a scenario with n_c cores each having a TDMA time slot of length s_l cycles, a single bus access can incur a *maximum bus access delay* of

$$D^{max} = ((n_c - 1)s_l) + (e - 1) \tag{1}$$

cycles for a bus access which occupies the bus for e cycles.

Example 1 This maximum delay is encountered when the access request is issued $e-1$ cycles before the end of the executing core's slot. The bus cannot be granted then, since the access would span into the slot of the next core. An example for this scenario is shown in Figure 1 for 2 cores, where core 1 issues a request "ACC" which gets delayed by D^{max} cycles.

On the other hand, the bus access is granted instantly, if the access request is issued when the bus is assigned to the executing core for at least e remaining cycles, i.e. if access "ACC" from the example of Figure 1 is issued during the time steps 0 to $s_l - e$. Thus, an important problem is to determine tighter bounds on the durations of bus accesses. D^{max} cycles, as mentioned, is a valid but highly overestimated bound.

We do not allow split transactions on the bus, therefore, for the maximum duration T^{max} of a bus transaction, $T^{max} \leq s_l$ must hold. An access to the TDMA bus may incur a variable delay, depending on when the access is performed, but the delay cannot exceed D^{max} cycles. As mentioned in the introduction, this bound is not tight in general. Due to $T^{max} \leq s_l$ and $D^{max} \geq ((n_c - 1)s_l)$, D^{max} will at least be $(n_c - 1)$ times as big as T^{max}. Thus, the bus access delay is the factor with the greatest variability and also with the greatest potential for overestimations during WCET analysis. This underlines the need for precise analyses of the bus access delays. In this article, we will provide such an analysis using a fixed TDMA schedule. The optimization of the TDMA schedule itself is out of the scope of the article.

All the caches in the considered system are non-inclusive and use the *least-recently-used* (LRU) replacement policy. The cache hierarchy can be easily extended e.g. with more private cache levels, because we apply the generic framework from [Hardy and Puaut (2008)] to determine which accesses from cache level $i-1$ hit cache level i. We only model instruction caches and thus assume that data accesses occur via a different bus and do not interfere with the instruction accesses in any other way. The integration of a data cache analysis into our analysis would remove these restrictions. We do not allow self-modifying code hereby removing the need to deal with cache-coherency in our model. Also, all shared libraries are duplicated for each core that uses them.

2.2 Input program model

While Section 2.1 presented our assumed hardware model, this section will introduce the model of the input programs / tasks. We start with a definition of the basic unit of execution, the basic block.

Definition 2 A *basic block* $b = (i_1, \ldots, i_k)$ is a sequence of instructions which may only be entered at i_1 and only be exited at i_k. In addition, b must also either not contain any instruction which potentially accesses the shared bus, or the block contains only a single instruction.

This definition does not conform with the usual definition of basic blocks, but this is motivated by the needs of our analysis as we will see in Section 4. Note that we consider calls as "exiting the block" and thus a call terminates a block.

Definition 3 A *function* $f = (B_f, b_f^{\text{start}}, b_f^{\text{exit}})$ is a user-defined set of basic blocks B_f together with a starting block b_f^{start} at which the execution of the function starts and an exit block b_f^{exit} at which the function is left. For each loop L in a function, we require the minimum and maximum loop iteration counts B_L^{min} and B_L^{max} to be given. Each function f has an *intraprocedural control flow graph* $G_f = (B_f, E_f)$, which for all $b_1, b_2 \in B_f$ contains an edge $(b_1, b_2) \in E_f$ if immediate flow of control from $b_1 \in V_f$ to $b_2 \in V_f$ is possible.

Definition 4 A *task* $t = (F_t, f_{\text{start}}^t)$ consists of a set of functions F_t and a start function f_{start}^t (usually called `main`) which is executed when the task starts.

Definition 5 The *interprocedural control flow graph* (IPCFG) $G_t = (V_t, E_t)$ of a task t has

$$V_t = \bigcup_{f \in F_t} B_f \tag{2}$$

$$E_t = \bigcup_{f \in F_t} E_f \cup E_{\text{call}} \tag{3}$$

where for every call from a basic block $b_{\text{call}} \in B_{f_1}$ from function $f_1 \in F_t$ to $f_2 \in F_t$ there is an edge $(b_{\text{call}}, b_{\text{start}}^{f_2}) \in E_{\text{call}}$ and a respective return edge $(b_{\text{exit}}, b_{\text{succ}}) \in E_{\text{call}}$

with $b_{\text{exit}} \in B_{f_2}^{\text{exit}}$. $b_{\text{succ}} \in B_{f_1}$ is the unique successor block of the call block in function f_1. The source node of the graph is $v_{G_t}^{\text{source}} = b_{\text{start}}^{f_{\text{start}}^t}$. We require functions to be non-recursive, therefore G_t will be loop-free.

Definition 6 A *path* through a control flow graph $G = (V,E)$ is a sequence of nodes $P = (v_0, v_1, \ldots, v_n)$ such that $(v_{i-1}, v_i) \in E$ for all $i \in \{1, \ldots, n\}$. The notation $v \rightarrow w$ is used to describe an arbitrary path from v to w. The time that is takes the given platform to execute a path P is denoted as $et(P)$.

Definition 7 The *Worst-Case Execution Path* (WCEP_t) of a task t is a path $b_{\text{start}}^{f_{\text{start}}^t} \rightarrow b_{\text{exit}}^{f_{\text{start}}^t}$ through G_t with maximum $et(P)$ among all such paths P. The *Worst-Case Execution Time* (WCET_t) of task t is equal to $et(WCEP_t)$.

We require our input tasks to be *well-structured*, which we will detail in the following. To formally define this term, we first need the notion of dominance and back-edges:

Definition 8 For a given control flow graph $G = \left(V, E, v_G^{\text{source}}\right)$, a node $u \in V$ *dominates* a node $v \in V$ iff $u \in P$ holds for every path $P = \left(v_G^{\text{source}}, \ldots, v\right)$. A *back-edge* $(u,v) \in E$ is an edge where v dominates u.

With these terms, we can define the reducibility of a control flow graph. Our definition follows the one in [Muchnick (1997)].

Definition 9 A control flow graph $G = \left(V, E, v_G^{\text{source}}\right)$ is *reducible* iff E can be partitioned into the disjoint sets E_F (forward edge set) and E_B (backward edge set), such that (V, E_F) forms a directed acyclic graph in which each node can be reached from the source node, and the edges in E_B are all back-edges.

In the following, we will use *well-structured* as a synonym for *reducible*. This is motivated by the fact that in a reducible control flow graph, loops can be unambiguously identified and back-edges can be unambiguously mapped to their corresponding loops [Muchnick (1997)]. Since we will need to identify loops in our analyses, we require the interprocedural control flow graphs of our input tasks to be reducible.

The whole input is then given as an acyclic task graph with a fixed mapping of tasks to cores. Each edge (x,y) in the task graph denotes that task y can start execution only after task x has finished. We use fixed-priority, non-preemptive scheduling. A preemptive scheduling would require the integration of a cache-related preemption-delay (*CRPD*) analysis [Altmeyer et al (2010)] which is out of the scope of this article.

3 Analysis Framework

The analyses are implemented inside the CHRONOS timing analyzer framework from [Chattopadhyay et al (2010)]. Figure 2 shows the analysis process. The framework

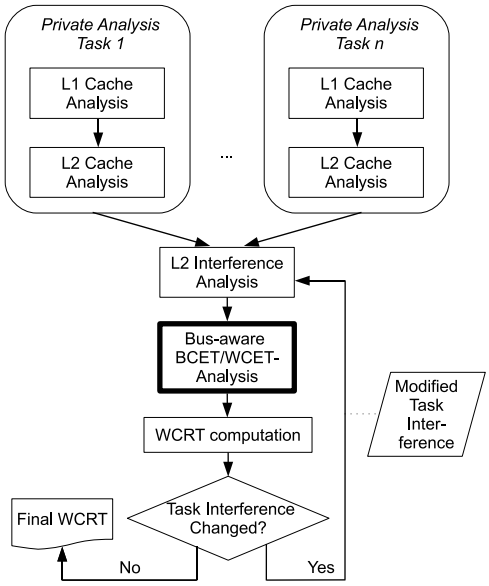


Fig. 2 The analysis framework

first analyzes the cache behavior of each task in isolation and then computes the maximum possible cache interference in the shared L2 cache. This interference information is used to update the worst-case cache states of the individual tasks. The cache analysis assigns to each single access one of the following categories for each cache level:

AH	"Always Hit"
AM	"Always Miss"
PS	"First Miss" / "Persistent"
UNKNOWN	"Unknown Behavior"

PS means that the first execution of the instruction suffers a cache miss, but every following execution hits the cache, which is most useful for instructions inside of loops. For details on the cache analysis, the interested reader is referred to [Chattopadhyay et al (2010)], since we are only using its results here. In the next analysis step, the cache information is used to compute BCET[1] and WCET values per task. This module (marked in bold in Figure 2) contains WCET analyses which we examine in this article. After the tasks' BCETs and WCETs were computed, the system's *worst-case response time* (WCRT) is determined. This process repeats as long as the task interference changes, e.g. due to altered task lifetimes, which in turn may lead to more precise WCETs. In the following, we will focus on the determination of single task WCETs with given cache states as this is our main contribution. Nevertheless, all presented analyses are applicable to the computation of BCETs as well.

[1] Best-Case Execution Time

4 Static Analysis of TDMA Offsets

The analyses presented in this article build upon concepts which are heavily used in the analysis of other architectural features. To establish the link to those existing analyses, we first give a short overview of existing static analysis techniques. We also demonstrate why those techniques are not sufficient for TDMA access delay analysis.

4.1 Abstract interpretation in timing analysis

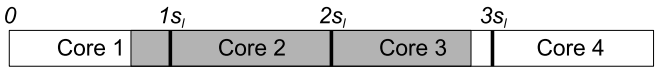
A static timing analysis is usually composed of a microarchitectural analysis and a path analysis [Wilhelm et al (2008)]. The microarchitectural analysis is responsible for determining abstract hardware states which describe the possible concrete hardware states at every basic block entry. This microarchitectural analysis is normally based on *abstract interpretation*, a technique for static program analysis, which can provide safe approximations of program or, in this case, hardware states. In the past, it was successfully employed to analyze cache, branch prediction and pipeline behavior. With these hardware states, a basic block's WCET can be computed, which in turn can be fed into the path analysis to compute the longest path through the program.

As we have seen in Section 2, the execution time of a bus access heavily depends on the time at which the access is made. Thus, we will try to determine or at least approximate that time in the following analyses, to be able to more precisely predict how long it will take for the bus accesses to execute. After our definition of basic blocks (see Definition 2), a basic block can either consist of multiple non-bus-accessing instructions or of a single potentially bus-accessing instruction. The information whether an instruction potentially accesses the shared bus can be extracted from the cache information. In our case, it may access the bus when it may access the L2 cache. This means that the WCET of basic blocks following our definition is either fixed (no bus access) or variable (bus access) which will simplify the analysis. The basic blocks execute in-order, since we required an in-order pipeline. A generalization of our concepts to out-of-order execution is possible, but is omitted for the sake of presentation clarity[2].

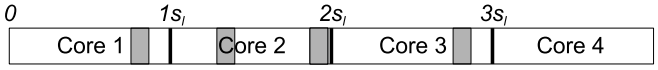
Since every possible bus access forms a basic block of its own, it is now sufficient to be able to approximate the starting time of all basic blocks, to obtain bounds on the times at which the bus is accessed. The abstract hardware states which are used by our analyses thus must model a set of time instants at which the execution of a basic block may start. Note that the bus access delay does not depend on the absolute time at which the access is performed, but only on the position of the access inside the cyclic TDMA schedule (compare Figure 1).

Definition 10 The *absolute time* in the analyzed system is measured in processor cycles, since we assume a processor with constant clock frequency. An *absolute point*

[2] In case of out-of-order pipelines, the analysis which we will present in the following would need to consider all orders in which the instructions of basic blocks can possibly be executed in separation and merge this information afterwards to get a valid overapproximation.



(a) Offset interval



(b) Offset set

Fig. 3 Different abstract representations for possible start offsets of a basic block

in time in an execution is given as $t \in \mathbb{N}_0$ which means the t-th clock cycle after the start of the system. An *offset* o can be computed from an absolute point in time t as $o = (t \mod n_c s_l)$.

To approximate the bus access delay it is therefore sufficient to approximate the *offsets* instead of absolute times. To be able to model the fact that a block can be entered with more than one offset we devise two offset representations:

- An *offset interval* $I = [o_{min}, o_{max}]$
- An *offset set* $O = \{o_1, o_2, \ldots, o_n\}$

The set of all possible offset intervals (offset sets) is denoted as I^+ (O^+). These offset representations are the abstract hardware states that will be used in the analyses.

Example 2 An example for the different representations can be found in Figure 3. While Figure 3(b) shows the offset set representation with the represented offsets marked in gray, Figure 3(a) presents the same offset information, again marked in gray, for the offset interval representation.

Obviously, the set representation is more precise, but it also requires greater effort to maintain the sets during the analysis, thus leading to a typical tradeoff between analysis precision and analysis duration. In the following, we will only use the set representation to keep the presentation clear, but all of our algorithms can also be applied based on the offset interval representation.

With our notion of basic blocks and the results from the other microarchitectural analyses which yield WCET values for the blocks without bus accesses, we can formulate the offset analysis as a classical dataflow analysis [Aho et al (2006), Cousot and Cousot (1979)]. The dataflow analysis requires a transfer function $u_b : O^+ \to O^+$ which returns the offsets which result after the execution of a given basic block b and a join function $m : O^+ \times O^+ \to O^+$ which merges the states at control flow joins in the control flow graph. Given the set $ET_b \subseteq \mathbb{N}$ of possible execution times of b and either an offset set S or an offset interval I, we have

$$u_b(O) = \begin{cases} \text{off}_{\text{execute}} & \text{if } b \text{ never accesses the bus} \\ \text{off}_{\text{execute}} \cup \text{off}_{\text{access}} & \text{if } b \text{ may access the bus} \\ \text{off}_{\text{access}} & \text{if } b \text{ always accesses the bus} \end{cases} \quad (4)$$

with

$$\text{off}_{\text{execute}} = \{(o+e) \mod s_l n_c | o \in O, e \in ET_b\} \tag{5}$$

$$\text{off}_{\text{access}} = \left\{(o+\Phi_p(o,e)+e) \mod s_l n_c | o \in O, e \in ET_b\right\} \tag{6}$$

where $\text{off}_{\text{execute}}$ (Equation 5) denotes the offsets resulting from the execution of the block's instruction in the pipeline and $\text{off}_{\text{access}}$ (Equation 6) denotes those which result from the instruction execution and a succeeding bus access. Since we have required all instructions which potentially access the bus to form a basic block of their own, any block for which we have to compute $\text{off}_{\text{access}}$ will only consist of a single instruction. The $\Phi_p(o,e)$ function returns the time for which a bus request has to wait until it is granted, when the bus request is issued by core $p \in \{0,\ldots,n_c-1\}$, begins at offset $o \in \{0,\ldots,n_c s_l - 1\}$ and needs $e \in \{1,\ldots,T^{max}\}$ cycles to complete after the bus access was granted. In the TDMA arbitration we can define $\Phi_p(o,e)$ as:

$$\Phi_p(o,e) = \begin{cases} s_l p - o & \text{if } o < s_l p \\ 0 & \text{if } s_l p \leq o \leq s_l(p+1) - e \\ s_l n_c - o + s_l p & \text{else} \end{cases} \tag{7}$$

In Equation 7, the first case corresponds to an access *before* the current core's slot, the second case is an access *inside* the current core's slot and the third case handles an access *after* the current core's slot.

To show the correctness of the offset update function u_b we define a *TMDA hyperperiod* as follows:

Definition 11 A *TDMA hyperperiod* is an absolute time interval $[t_a, t_e)$ with $t_a \bmod n_c s_l$ and $t_e = t_a + n_c s_l$

The following lemma shows, that our update function u_b correctly models the offset progression for a single execution of the basic block b.

Lemma 1 *For any $O \in O^+$, $u_b(O)$ contains the offsets of all absolute time instants t such that t is the first cycle after the execution of basic block b, starting at an offset $o \in O$.*

Proof If a particular execution of the basic block does not access the bus, $\text{off}_{\text{execute}}$ from Equation 4 contains all possible resulting offsets, since ET_b is the set of all possible running times then. If the particular execution of the block does access the bus, the block only consists of a single instruction, according to Definition 2. $\text{off}_{\text{access}}$ contains the possible offsets of the first cycle t after the execution of the basic block for any starting offset $o \in O$ and runtime $e \in ET_b$. Since the only difference between $\text{off}_{\text{execute}}$ and $\text{off}_{\text{access}}$ is the application of the Φ_p function, we show this by examining the three cases from Equation 7:

- In the first case, the access has to be delayed until the start of core p's slot in the current TDMA hyperperiod.
- In the second case, the access can be granted immediately, since the bus is allocated to core p and will be allocated to p for at least e cycles.

– In case three, the access cannot be served in the current TDMA hyperperiod and thus must be delayed to the next TDMA hyperperiod (as shown in Figure 1).

By taking the union over all possible starting offsets $o \in O$ and execution times $e \in ET_b$ in Equation 6, the Lemma follows for this case, too. □

Note that ET_b may for example model the fact that we have a block with variable-latency instructions or a block whose L2 instruction memory access was classified as UNKNOWN. The join function m is defined as:

$$m(O_1, O_2) = O_1 \cup O_2 \tag{8}$$

We generalize m to take n arguments instead of just two by defining

$$\forall n > 2 : m(O_1, \ldots, O_n) = m(O_1, \ldots, O_{n-1}) \cup O_n \tag{9}$$

Analogously, we generalize u_b to sequences $q = (b_1, b_2, \ldots, b_n)$ of basic blocks by setting

$$n = 1 : u_{(b_1)}(O) = u_{b_1}(O) \tag{10}$$

$$\forall n > 1 : u_{(b_1, b_2, \ldots, b_n)}(O) = u_{b_n}\left(u_{(b_1, b_2, \ldots, b_{n-1})}(O)\right) \tag{11}$$

Definition 12 A *b-trace* for a basic block b in the interprocedural control flow graph of a task t is a path from $b^{f^t_{\text{start}}}_{\text{start}}$ to b. The set of all b-traces is called Q_b.

Definition 13 The *Meet-Over-All-Paths* (MOP) solution to the problem of determining the possible offsets with which a basic block b may be entered when the surrounding task is started with offsets S, is given as

$$O^{\text{MOP}}_b(S) = m\left(\left\{u_{q_b}(S) \,|\, q_b \in Q_b\right\}\right) \tag{12}$$

Theorem 1 *The MOP solution provides a valid overapproximation of all offsets with which block b can be entered.*

Proof m joins the offsets resulting from the single b-traces which represent all execution paths leading to b. We must thus only prove that $u_{q_b}(S)$ is an overapproximation of the offsets which result from the execution of b-trace q_b starting with an offset $o \in S$. This can be proven via induction over the length of q_b where the induction step is made by applying Lemma 1. □

Instead of directly computing the MOP solution, which would be computationally expensive, we can conduct a standard dataflow analysis on the interprocedural control flow graph of each task. Since our transfer function u is monotonic and S^+ is a power set, this dataflow analysis will terminate and the result will be equal to the MOP solution. This follows from the Kleene Fixpoint Theorem and the Coincidence Theorem [Cousot and Cousot (1979)]. We will not go into more detail here, since this is a purely technical application of classical dataflow theory and is of no importance for our own analyses.

Unfortunately, this dataflow analysis will not be very precise, because *branches* and *loops* in the control flow force us to repeatedly merge the offset information,

which quickly leads to results where a block can be reached with arbitrary offsets. In this situation, we cannot provide a better estimation than the pessimistic assumption that each bus access is delayed by D^{max} cycles. The imprecision that stems from branches can be reduced through the offset set representation which allows to track the offset development in more detail. Loops pose a bigger problem. They can only be handled effectively with the concept of *contexts* in the analysis.

4.2 Abstract hardware states and contexts

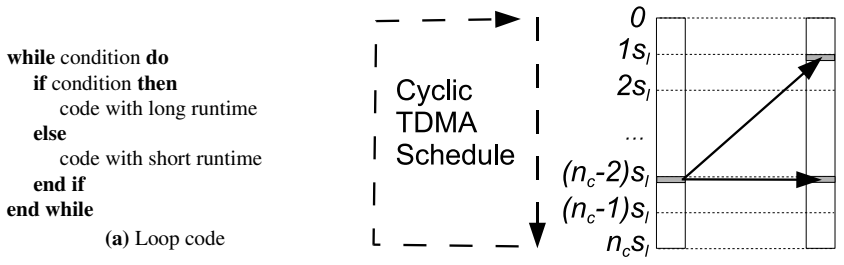
Usually, the hardware states presented in Section 4.1 are computed in a context-insensitive way, meaning that the abstract interpretation computes states which are valid for all *execution contexts* of a basic block, where an execution context denotes a certain loop iteration or calling context. This can be observed e.g. in Equation 12, where we merge the offset information over *all* b-traces. This behavior is insufficient for some analyses like e.g. the cache analysis, where the first loop iteration may have a significantly different cache behavior than the following ones. For this purpose, *analysis contexts* were introduced, which describe the hardware states for a certain execution context. The known methods for dealing with contexts during bus access duration analysis are the following:

- The loop is virtually unrolled by a factor equal to its loop bound and thus, each loop iteration is explicitly analyzed [Andrei et al (2008)]. This method, called *full virtual unrolling* is very precise but also very inefficient for larger loop bounds. It results in the analysis of B_L^{max} analysis contexts, each of which represents exactly one execution context.
- The analysis is performed for a fixed offset o, and a delay is added that represents the maximum additional delay that can occur due to execution with offsets $s \neq o$. This is the approach pursued by [Chattopadhyay et al (2010)], and we will refer to it under the name *fixed-alignment approach*. It results in a single analysis context which represents all B_L^{max} execution contexts.
- We analyze $1 \leq x \leq B_L^{max}$ contexts to provide a compromise between analysis duration and analysis precision. This is the approach from [Kelter et al (2011)] which is based on the analysis of TDMA offsets as presented above, and which we will review and extend in the next sections.

5 Computing Loop Offset Bounds

Our approach is based upon the observation that for each loop iteration which starts from a given set of offsets, we can compute the set of offsets in which the iteration may terminate. Therefore, our goal is to track the development of the TDMA offsets of the loop header block and thus to provide more precise offset bounds than by using the data flow analysis from Section 4. This requires:

- A structural analysis to find loops in the CFG, and to build a *directed acyclic graph* (DAG) from each loop or function body. Nested loops are represented as



(a) Loop code

(b) TDMA offset progression for a single iteration

Fig. 4 Mapping of a start offset to possible end offsets for a single loop iteration

single nodes in the surrounding DAG. Due to this, we required our input tasks to be reducible in Section 2.

– An analysis that computes the set of offsets that may be reached when a loop body is executed once with given starting offsets.

The overall analysis will then proceed in a hierarchical way, starting at the beginning of the task entry function and descending into called functions or loops only when they are discovered in the CFG. The structural analysis is already present in the CHRONOS framework, whereas a single-iteration offset analysis is presented in Section 5.1. Section 5.2 then introduces the core analysis which combines the single-iteration results into a complete loop WCET.

5.1 Determination of offset results for single iterations

As mentioned, we are interested in determining the offsets that can be reached after a single execution of the loop body finishes. This will be called a *loop iteration* in the following, in contrast to a *loop execution* which denotes the (possibly) repeated execution of the loop body until the loop condition is false. It is important to note, that a loop iteration, starting from a single, given offset may end at different offsets, e.g. due to variability in the runtime of instructions, due to UNKNOWN cache accesses or due to different paths through the loop.

Example 3 The case of different paths through a loop is exemplified in Figure 4. The loop is shown in Figure 4a, and Figure 4b shows two TDMA hyperperiods. The start offset is shown on the left hand side and the possible result offsets are shown on the right hand side of Figure 4b. The top arrow represents a loop iteration of length $3s_l$ cycles whereas the bottom one represents a loop iteration of length $n_c s_l$ cycles which corresponds to the two branches of the `if`-statement in the loop code shown in Figure 4a. Thus Figure 4b shows that a single iteration of the given loop with start offset set $\{(n_c-2)s_l\}$ may only end with one of the offsets in $\{1s_l,(n_c-2)s_l\}$.

In the following, an analysis which is able to compute such offset results for single loop iterations is presented. In this analysis, each basic block is seen as a

Algorithm 1 AnalyzeBlock

Require: block b, offsets O_{in}
1: **if** b is head of inner loop l_{inner} **then**
2: **return** AnalyzeLoop (l_{inner}, O_{in})
3: **else**
4: $wcet = 0$
5: **if** b consists of bus access instruction **then**
6: **for all** offset $o \in O_{in}, e \in ET_b$ **do**
7: $wcet = \max(wcet, \Phi_p(o,e) + e)$
8: **end for**
9: **else**
10: $wcet = \max(ET_b)$
11: **end if**
12: $result = \langle wcet, u_b(O_{in}) \rangle$
13: **if** b is terminated by call to function f **then**
14: tmp = AnalyzeFunction $(f, result.offsets)$
15: $result = \langle tmp.wcet + result.wcet, tmp.offsets \rangle$
16: **end if**
17: **return** $result$
18: **end if**

transformation function which maps input offsets O_{in} (either an offset set or an offset interval as explained in Section 4.1) to resulting offsets O_{out} and produces WCET values which are valid for the given O_{in}. Algorithm 1 shows the analysis of single basic blocks. Function calls (lines 13 - 16) or blocks which represent inner loops (line 2) are handled by specialized analysis functions. Note that function calls terminate basic blocks in our model. The WCET and offsets which result from bus accesses (lines 4 - 12) or simple instructions (line 10) are computed with the known ET_b values and Φ_p and u_b functions from Section 4.1, where p is the core which executes the currently analyzed task.

Each DAG analysis, on either a function or a loop body, then composes the single-block results in topological order and forms its own WCET and offset result out of them. Algorithm 2 shows this for the case of a single loop iteration, where b_{sink} and b_{header} are the sink and header node of loop l, respectively and $pred(b_i)$ returns the set of predecessor blocks for block b_i. By supplying the starting offsets to the loop iteration analysis (lines 3 - 4), this information becomes part of the analysis context, as explained in Section 4.2. The iteration analysis then analyzes the behavior of each single block (lines 9 - 11) and propagates the results to the successor blocks (lines 6 - 7). Finally the results per loop iteration are summarized (line 13). The analysis of functions in "AnalyzeFunction" (Algorithm 3) works analogously as "AnalyzeLoop-Iteration". As stated, recursive calls must be converted to standard loops before our analysis can handle them.

Theorem 2 *For a given interprocedural control flow graph of a task t and given starting offsets O_{in}, the results $w \in \mathbb{N}$ and $O \in O^+$ as computed by Algorithm 3 for function f_t^{start} are overapproximations of the WCET and the resulting offsets of any execution of t which starts with an offset $o \in O_{in}$.*

Proof We cannot present the full proof at this point, since we have not yet presented the "AnalyzeLoop" function. Nevertheless we will introduce the structure of the proof

Algorithm 2 AnalyzeLoopIteration

Require: loop l, offsets O_{in}
1: **for all** blocks b_i of loop l in topological order **do**
2: **if** $b_i = b_{header}$ **then**
3: $wStart = 0$
4: $oStart = O_{in}$
5: **else**
6: $wStart = \max_{b_d \in pred(b_i)} (wFinish[b_d])$
7: $oStart = m(\{oFinish[b_d] \mid b_d \in pred(b_i)\})$
8: **end if**
9: $\langle wcet_{b_i}, O_{b_i} \rangle$ = AnalyzeBlock $(b_i, oStart)$
10: $wFinish[b_i] = wStart + wcet_{b_i}$
11: $oFinish[b_i] = O_{b_i}$
12: **end for**
13: **return** $\langle wFinish[b_{sink}], oFinish[b_{sink}] \rangle$

Algorithm 3 AnalyzeFunction

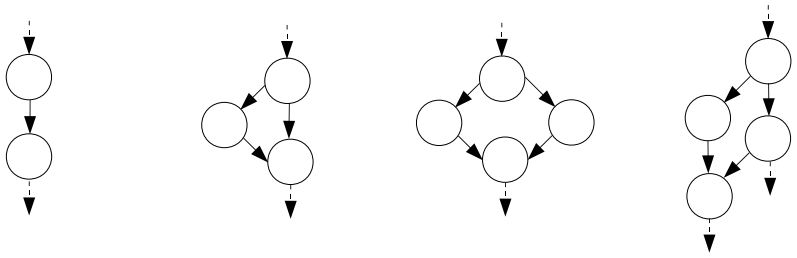
Require: function r, offsets O_{in}
1: **for all** blocks b_i of function r in topological order **do**
2: **if** $b_i = b_r^{\text{start}}$ **then**
3: $wStart = 0$
4: $oStart = O_{in}$
5: **else**
6: $wStart = \max_{b_d \in pred(b_i)} (wFinish[b_d])$
7: $oStart = m(\{oFinish[b_d] \mid b_d \in pred(b_i)\})$
8: **end if**
9: $\langle wcet_{b_i}, O_{b_i} \rangle$ = AnalyzeBlock $(b_i, oStart)$
10: $wFinish[b_i] = wStart + wcet_{b_i}$
11: $oFinish[b_i] = O_{b_i}$
12: **end for**
13: **return** $\langle wFinish[b_{sink}], oFinish[b_{sink}] \rangle$

here and add the missing parts later. We prove the proposition by structural induction over the interprocedural control flow graph.

Base case: The smallest possible graph is a single basic block. Therefore, we have to prove the proposition for a single basic block to give the induction base case. According to Definition 2, the basic block either consists of a single instruction which accesses the bus, or of multiple instructions which do not access the bus.

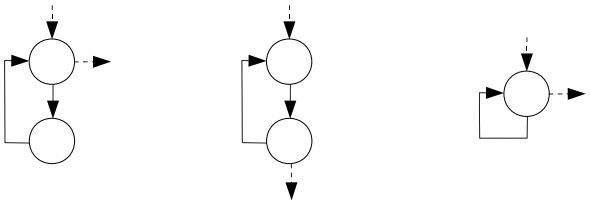
- A basic block with a bus access
  In this case, the returned WCET is a valid overapproximation since we compute the maximum over all possible completion times as returned by $\Phi_p + e$.
- A basic block without a bus access
  In this case, the returned WCET is a valid overapproximation since we maximize over the given ET_b values.

The correctness of the offset result follows from Lemma 1, since the result is computed through a single application of the transfer function u.



(a) Sequence **(b)** If-Then **(c)** If-Then-Else **(d)** Proper

Fig. 5 Sequential structural patterns



(a) While loop **(b)** Natural loop **(c)** Self loop

Fig. 6 Cyclic structural patterns

Induction step: The induction step must consider the possible structures which can appear in the CFG. We required our interprocedural control flow graphs to be reducible in Section 2. A reducible control flow graph can be inductively defined with the patterns shown in Figures 5 and 6. Every graph which adheres to Definition 9, which includes our control flow graphs, can be constructed using those inductive patterns [Muchnick (1997)]. In the patterns, the circles indicate reducible subgraphs. For the induction step, we can assume that the proposition was already shown for the subgraphs. We then must prove that the proposition is also true for the depicted graphs as a whole. This is done by looking at the different cases:

- Sequential patterns
  According to the induction hypothesis, the WCET and offset results for the subgraphs are valid overapproximations. For the sequential case shown in Figure 5a we add up the WCETs and combine the offset results in lines 9 to 11 of Algorithm 3. This obviously yields overapproximations for the whole sequence.
  For the case of branches as shown in Figure 5b and 5c, we compute safe overapproximations, since we take the maximum WCET of any path leading to the end block in line 6 of Algorithm 3. Similarly, we merge the result offsets of all paths reaching the end block in line 7 of the same algorithm.

The last sequential case as shown in Figure 5d is a combination of an if-then with a sequence. Therefore, the correctness for this case follows from the same arguments as in those cases.

– Cyclic patterns
The possible cyclic patterns are shown in Figure 6. We omitted patterns for loops which contain `break` or `continue` statements, since the generalization to these cases is a pure technicality. The induction step for the case of loops has to be supplied when the analysis function "AnalyzeLoop" was presented. □

For the analysis of complete loop executions (all iterations) in "AnalyzeLoop", we need to combine the context-sensitive single iteration results to form an overall loop WCET and offset result. This will be discussed in the next section.

5.2 Deriving full loop WCETs

To implement "AnalyzeLoop" for a given loop l and starting offsets $O_{in,l}$, full unrolling could be performed by analyzing all iterations and supplying the offset results from one iteration as inputs to the next one. Alternatively, only a single iteration can be analyzed, with a forced alignment at the TDMA schedule border and an added alignment penalty as suggested in [Chattopadhyay et al (2010)]. Section 4.2 already mentioned that our goal is to avoid these two approaches, because they are computationally too expensive or lose precision, respectively. In this section we present and review two methods from [Kelter et al (2011)] which present a compromise between those two extremes.

In the following sections we will use $wcet_l^{LB}(O)$ and $u_l^{LB}(O)$ to denote the (safe) WCET and offset results of a single loop iteration starting at an offset $o \in O$ as computed by Algorithm 2. In the proofs of correctness of the proposed "AnalyzeLoop" functions, we can use the induction hypothesis from Theorem 2, that $wcet_l^{LB}(O)$ and $u_l^{LB}(O)$ compute valid overapproximations.

Our analyses will concentrate on the case of natural loops (see Figure 6b). Self loops (see Figure 6c) are a special case of a natural loop which consists of a single basic block. For while loops (see Figure 6a), the loop condition is evaluated one more time before the loop is exited, which must be accounted for in the analysis. Since this a purely technical issue, we will omit this in the following.

5.2.1 Global Convergence Analysis

Starting with the initial offset information $O_{in}^1 = O_{in,l}$ we iteratively analyze single loop iterations $i \in \{1,2,\dots\}$ and record the WCET $wcet_i = wcet_l^{LB}\left(O_{in}^i\right)$ and offset result $O_{out}^i = u_l^{LB}\left(O_{in}^i\right)$. With the merge function m from Section 4.1 the offset inputs O_{in}^i for iteration i are then computed as $m\left(O_{in}^{i-1}, O_{out}^{i-1}\right)$. The analysis stops after iteration j when either $j = B_l^{max}$ or $O_{in}^j = O_{in}^{j+1}$ is true. In the first case, we have hit the loop bound and thus have performed full unrolling implicitly, therefore this

is the undesired case. In the second case, we have reached a fixpoint of the starting offsets and thus the result from iteration j stays valid for all following iterations. In total there cannot be more than $\min\left(B_l^{max}, n_c s_l\right)$ iterations, which is the number of possible offset values. The final loop WCET can then be easily computed as:

$$wcet_l\left(O_{in,l}\right) = \left(\sum_{i=1}^{j} wcet_i\right) + \left(B_l^{max} - j\right) \cdot wcet_j \tag{13}$$

The offset result for the loop is equal to the offset result from iteration j, because this result stays valid for all following iterations.

Lemma 2 *For two offset sets O_1 and O_2 with $O_1 \subseteq O_2$ we observe that $wcet_l^{LB}(O_1) \leq wcet_l^{LB}(O_2)$ and $u_l^{LB}(O_1) \subseteq u_l^{LB}(O_2)$. For $wcet_l^{LB}$ this can be derived from the monotony of Φ_p and for u_l^{LB} it can be derived from the monotony of the m and u_b functions. Thus, $wcet_l^{LB}$ and u_l^{LB} are monotone.*

Theorem 3 *For given starting offsets $O_{in,l}$, the global convergence analysis computes safe overapproximations of the loop WCET and result offsets.*

Proof This proof handles the case of cyclic patterns in the proof of Theorem 2 and thus is a plug-in for this proof. If we would set $O_{in}^i = O_{out}^{i-1}$ in the analysis, then we would perform a fully unrolling analysis, which would be unlikely to converge at any time step before the loop bound. The safeness of this fully unrolling analysis then follows from the safeness of the single-iteration analysis which we can assume since this is the induction hypothesis from Theorem 2. We use $O_{in}^i = m\left(O_{in}^{i-1}, O_{out}^{i-1}\right)$, therefore in our algorithm $O_{in}^i \supseteq O_{out}^{i-1}$ holds. Lemma 2 implies that the WCET and offset results which we compute per iteration are overapproximations of the real WCET and offsets. This proves the correctness of the algorithm for the first j loop iterations. Then we have two cases:

- $j = B_l^{max}$
  In this case, all loop iterations were analyzed and thus the correctness of the analysis was shown for all loop iterations.
- $O_{in}^j = O_{in}^{j+1}$
  In this case, since O_{in}^j is a safe overapproximation of the offsets in loop iteration j and $O_{in}^{j+1} = u_l^{LB}\left(O_{in}^j\right)$ is a safe overapproximation of the offsets in loop iteration $j+1$, the loop can never be entered with offsets $o \notin O_{in}^j$ in any succeeding iteration $k > j$. Therefore the offset and WCET results for the j-th iteration are safe overapproximations for all $B_l^{max} - j$ remaining iterations. □

5.2.2 Graph Tracking Analysis

The global convergence analysis is superior to a static unrolling insofar, that it implicitly unrolls the loops selectively, as long as new information can be obtained. Nevertheless, this still relies on the idea of unrolling the first j iterations and handling the rest of the iterations under a single analysis context. The drawback is that cyclic progressions of offsets cannot be captured by the analysis.

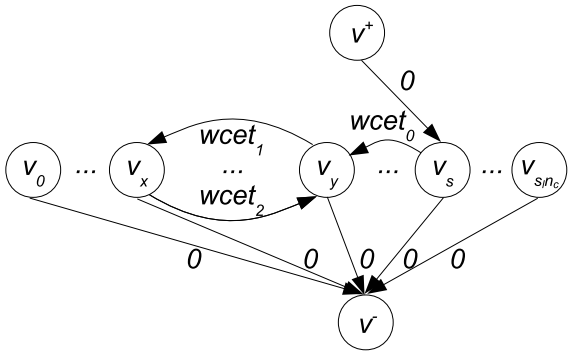


Fig. 7 An example offset graph

Example 4 Consider e.g. a loop in which all even iterations start with offset x and all odd iterations start with offset $y \geq x$, because only the even iterations have to wait for the TDMA bus access, whereas the odd iterations can then proceed with direct bus access. The global convergence analysis will analyze the first two iterations ($j = 2$), compute $O^j_{in} = \{x, y\}$ and use this offset information for all following iterations. This is clearly valid, but still imprecise.

The example shows the need to handle *cyclic contexts* which do not distinguish the first j execution contexts from the remaining ones, but which distinguish *groups* of execution contexts which repeat cyclically. In our case, a cyclic context consists of all iterations starting with offset o, which leads to $s_l n_c$ contexts. Thus, we can identify a cyclic context via the offset which it represents.

To obtain the final timing results using cyclic contexts, we construct a weighted, directed graph from the contexts and compute the loop WCET by solving a flow problem on that graph.

Definition 14 An *offset graph* $G = (V, E, c)$ consists of a set of nodes $V = \{v^+, v^-\} \cup V_{\text{off}}$ with *source* v^+, *sink* v^-, *context nodes* $V_{\text{off}} = \{v_0, ..., v_{s_l n_c}\}$, of a set of edges $E = E_{enter} \cup E_{exit} \cup E_{transition}$ and of a weight function $c : E \to \mathbb{N}$. We have

$$E_{enter} = \{(v^+, v_x) \mid x \in O_{in,l}\} \tag{14}$$

$$E_{exit} = \{(v_x, v^-) \mid x \in [0, s_l n_c]\} \tag{15}$$

For all edges $e \in (E_{enter} \cup E_{exit})$ we set the weight $c(e)$ to 0. $E_{transition}$ is then constructed by iteratively analyzing single iterations. For each iteration i, we compute $wcet_i = wcet_l^{LB}(O^i_{in})$ and $O^i_{out} = u_l^{LB}(O^i_{in})$. O^1_{in} is set to $O_{in,l}$ and for the other iterations $O^i_{in} = O^{i-1}_{out}$ applies. After the analysis of each iteration we extend $E_{transition}$ by all edges $e = (v_i, v_o)$ with $i \in O^i_{in}, o \in O^i_{out}$ and $c(e) = wcet_i$. We stop the iteration analyses when we reach an iteration where no edge is added or when $i = B_l^{max}$.

Example 5 An example for such a graph is given in Figure 7 where the first iteration starts with offset s and the succeeding iterations alternate between starting offset x and y as sketched in Example 4.

The offset graph can then be used to obtain the final loop WCET by solving a *dynamic flow problem* [Skutella (2009)]. In contrast to standard flow problems, dynamic flow problems have an explicit notion of time built into the problem formulation. Based on the offset graph, we can derive two different dynamic flow problems: one for determining the WCET and one for the resulting offsets. The basis of the problem formulation is a flow function $x : E \times T \to \mathbb{N}$, which specifies for each edge $e = (u,v)$ the amount of flow $x(e,t)$ which leaves u at the discrete time instant t. This flow arrives at v at time $t + \tau(e)$ where $\tau(e)$ is the constant runtime of the edge. Conceptually, in our graph, a single time step of the flow problem corresponds to a single iteration of the loop, which implies $T = \{0,...,B_l^{max}\}$. Thus, a flow of $x(e,t) = 1$ through an edge $e = (v,w) \in E_{transition}$ represents the loop iteration t which starts at offset v and ends at offset w and has a maximum runtime of $c(e)$. Therefore, we set $\tau(e) = 1$ for all $e \in E_{transition}$, since these edges model single loop iterations, and we set $\tau(e) = 0$ for all $e \in E_{enter} \cup E_{exit}$, modeling entry into and exit from the loop. Both dynamic flow problems share a common constraint that ensures that all flow which enters a node at a time step must leave it in the same step (i.e. there must be one loop iteration per time step):

$$\forall t \in T : \forall v \in V_{\text{off}} : \sum_{e \in \delta^-(v)} x(e, t - \tau(e)) = \sum_{e \in \delta^+(v)} x(e,t) \tag{16}$$

Here, $\delta^-(v)$ and $\delta^+(v)$ denote the sets of incoming and outgoing edges at node $v \in V$. For the start node v^+ and the sink node v^- we need to provide explicit bounds on the flow. We want F units of flow to leave v^+ at time 0 and to arrive at v^- at time B_l^{max}. The flow of each of the flow units through the graph models a possible loop execution scenario. F will be set to a specific value in the final ILPs, but since we will be solving two ILPs we keep the formulation generic first. Therefore, we have:

$$\sum_{e \in \delta^+(v^+)} x(e,0) = F \tag{17}$$

$$\forall_{e \in \delta^+(v^+)} : \forall t \in T \setminus \{0\} : x(e,t) = 0 \tag{18}$$

$$\sum_{e \in \delta^-(v^-)} \sum_{t \in T_{leave}} x(e,t) = F \tag{19}$$

$$\forall_{e \in \delta^-(v^-)} : \forall t \in T \setminus T_{leave} : x(e,t) = 0 \tag{20}$$

$$\text{with } T_{leave} = \{t \mid B_l^{min} \leq t \leq B_l^{max}\} \tag{21}$$

Equations 17 and 18 specify that the F flow units must leave v^+ exactly at time step 0 and Equations 19 and 20 ensure that they must arrive at v^- in the time interval $[B_L^{min}, B_L^{max}]$. In the time steps in between they must travel along the edges in $E_{transition}$.

Thus, to model the worst-case loop execution scenario we set $F = 1$ and maximize the objective function

$$\max \sum_{e \in E} \sum_{t \in T} c(e)\, x(e,t) \tag{22}$$

The loop WCET is then given by the value of the objective function. Effectively the one flow unit that is enforced by $F = 1$ sums up all single-iteration WCETs and the maximization ensures that the worst-case iteration path is chosen.

For the offset analysis, we use $F = s_l n_c$ flow units. If K is the (unknown) set of offsets with which the loop can be left, then we have $|K| \leq s_l n_c$ since this is the total number of possible offsets. With $F = s_l n_c$ flow units we can thus model at least one loop execution scenario which terminates with offset k for each offset $k \in K$. Therefore, we can compute an overapproximation of K by maximizing the objective function

$$\max |\left\{ z \mid \exists t \in T_{leave} : x\left(\left(v_z, v^-\right), t\right) > 0 \right\}| \tag{23}$$

The offsets $O_{out,l}$ which result after the loop execution are then given as the elements of the set from Equation 23 with $K \subseteq O_{out,l}$.

In the following, we will prove that the results of the graph problem are valid overapproximations of the WCET and offset results of the loop execution. For an offset graph $G = (V, E, c)$ we assume the following notations:

- $\forall E_x \subseteq E : src(E_x) = \{v | (v,w) \in E_x\}$
- $\forall E_x \subseteq E : dest(E_x) = \{w | (v,w) \in E_x\}$
- O_t^{real} is the set of offsets with which the loop header may be entered in the t-th iteration in any real execution of the loop.

Definition 15 An offset node v_n is *reachable at time t* iff there exists a flow function $x : E \times T \to \mathbb{N}$, subject to the constraints from Equations 16-20 with $F = 1$ and $\exists v_m \in V : \exists e = (v_m, v_n) \in E : x(e, t - \tau(e)) \geq 0$.

We define $reachable(t) = \{o | v_o \text{ is reachable at time } t\}$.

Lemma 3 *For a loop l, assume $O_{in,l}$ is an overapproximation on the set of offsets at the entry of the loop before the first iteration. We claim that $reachable(i) \supseteq O_i^{real}$ is true for all iterations of the loop.*

Proof Let us assume that the construction of the offset graph terminates at iteration m (thus, m is the last iteration of the construction) and the loop bound is i. We prove the proposition by induction over the loop bound.

Base case: We can use the outer induction hypothesis, that the offset results computed by the single-iteration analysis are valid overapproximations. With $O_{in,l}$ being an overapproximation of the input offsets and $i = 1$, this already proves the proposition since only a single loop iteration is modeled then.

Induction step: Due to the induction hypothesis we know that $reachable(i) \supseteq O_i^{\text{real}}$. We must show that $reachable(i+1) \supseteq O_{i+1}^{\text{real}}$ holds. To accomplish this, we assume that there is an offset $o_{err} \in O_{i+1}^{\text{real}}$ with $o_{err} \notin reachable(i+1)$. We will show that this leads to a contradiction.

If such an offset o_{err} exists, then by definition of O_{i+1}^{real} there must be a possible execution scenario A in which the $(i+1)$-th loop iteration is entered with offset o_{err}. Let $(a_1, a_2, \ldots, a_{i+1})$ be the offsets with which the first $i+1$ iterations of the loop are entered in scenario A. Note that this implies $a_{i+1} = o_{err}$. Since we assume that $o_{err} \notin reachable(i+1)$, there must be at least two such offsets a_p and a_q for which $\left(v_{a_p}, v_{a_q}\right) \notin E$. Using the induction hypothesis it follows that $a_p \in reachable(i)$ and thus that $p = i$ and $q = i+1$.

Since a_p is reachable in the graph, there must have been a construction iteration $j < \min(m,i)$ with $a_p \in O_{out}^j$ and $a_p \notin O_{in}^j$ where offset a_p was reached for the first time. In construction iteration $j+1$ we add all edges $E_{j+1} = O_{out}^j \times O_{out}^{j+1}$ to the graph. Since $O_{out}^{j+1} = u_l^{LB}\left(O_{out}^j\right)$ and $a_p \in O_{out}^j$, it follows that $a_q \in O_{out}^{j+1}$ since u_l^{LB} yields a safe overapproximation of the offsets and offset a_p is followed by offset a_q in scenario A. Therefore, we have $\left(v_{a_p}, v_{a_q}\right) \in E$ which is a contradiction. □

Theorem 4 *Let us assume $O_{in,l}^{real}$ is the set of offsets with which loop l may be entered in the first iteration. Given that $O_{in,l} \supseteq O_{in,l}^{real}$, the graph tracking analysis always computes an overapproximation of the total execution time of the loop.*

Proof We prove this by induction on the loop bound B_l^{max}.

Base case ($B_l^{max} = 1$: In this case, the objective function (Equation 22) simply takes the maximum of $c(e)$ where $e \in E_{transition}$ and $src(e) \in O_{in,l}$. Note that for any $e \in E_{transition}$, $c(e)$ represents the worst-case execution time of one loop iteration (computed by Algorithm 2) starting at offset $src(e)$. Therefore, $\max_{src(e) \in O_{in,l}} c(e)$ precisely represents the WCET of the first loop iteration. For $B_l^{max} = 1$ the ILP target function (Equation 22) is equal to this maximization, which proves the base case.

Induction step: We assume that the WCET computation is sound for loop bound $B_l^{max} = n$. We shall show that the computation is also sound for loop bound $B_l^{max} = n+1$. Let us assume that the actual WCET of the entire loop l with n iterations is denoted by $WCET(l,n)$. On the other hand, the actual WCET of the n-th iteration of the loop is denoted by $WCET_{iter}(l,n)$. According to the graph tracking analysis, we compute the WCET of the loop with $n+1$ iterations as

$$\max \sum_{e \in E} \sum_{t \in T} c(e) x(e,t) \tag{24}$$

where E is the set of all edges in the offset graph and $T = \{0, \ldots, n+1\}$. However,

$$\max \sum_{e \in E} \sum_{t \in T} c(e) x(e,t) = \max \sum_{e \in E} \sum_{t \in T'} c(e) x(e,t) + \max \sum_{e \in E} c(e) x(e, n+1) \tag{25}$$

where $T' = \{0,\ldots,n\}$. By induction hypothesis, we have

$$\max \sum_{e\in E}\sum_{t\in T'} c(e)x(e,t) \geq WCET(l,n) \tag{26}$$

From Lemma 3 we know that $reachable(n+1) \supseteq O^{\text{real}}_{n+1}$. If an offset node is not reachable in iteration $n+1$, then it cannot contribute to Equation 25, therefore

$$\max \sum_{e\in E} c(e)x(e,n+1) = \max \sum_{e\in\{(v,w)\in E|v\in reachable(n+1)\}} c(e)x(e,n+1) \tag{27}$$

$$\geq \max \sum_{e\in\left\{(v,w)\in E|v\in O^{\text{real}}{}_{n+1}\right\}} c(e)x(e,n+1) \tag{28}$$

$$= WCET_{iter}(l,n+1) \tag{29}$$

Inserting Equation 29 and 26 into Equation 25 provides the induction step. Thus, the proposition is proven. □

Theorem 5 *Computation of $O_{out,l}$ is sound. More precisely, $O_{out,l}$ predicted by the graph tracking analysis always overapproximates the set of offsets with which a loop may be left.*

Proof We are sending $s_l n_c$ flow units through the graph. Each one of these units models an independent execution of the loop. Each of these modeled executions (say they are numbered with $i \in \{1,...,n_c s_l\}$) will exit the loop with some offset $o_{end,i}$. The unknown set of all possible exit offsets is K. What we must show, is that $K \subseteq \{o_{end,i} | i \in \{1,...,n_c s_l\}\}$.

What we maximize in Equation 23 is the cardinality of the set of offsets with which the $s_l n_c$ flow units exit the loop. By Lemma 3 the reachable offsets in the flow graph are an overapproximation of the reachable offsets in the real loop execution for all iterations $j \in \{1,...,B_l^{max}\}$. Therefore, if the loop can be left in iteration $k \in \{B_l^{min},...,B_l^{max}\}$ with offset o_{left} during a real loop execution, then it is possible to construct a flow with one flow unit i which starts at v^+ at time 0 and takes the edge $e = (v_{o_{left}}, v^-)$ at time step k, thus $o_{end,i} = o_{left}$ for a given o_{left}.

Up to this point we have then shown, that for each exit offset $o_{left} \in K$ we can construct a flow with one flow unit that exits the loop with this offset. It is also possible that we get flows which end with offsets $o_{err} \notin K$, but that is no problem since we only require an overapproximation of the offsets. If we now assume that we compute a solution $O_{out,l}$ with an offset $k \in K$ and $k \notin O_{out,l}$, then we can easily show that this is a contradiction:

1. $|O_{out,l}| = s_l n_c$
   In this case, the set $O_{out,l}$ represents all possible offsets, therefore an offset $k \notin O_{out,l}$ cannot exist.
2. $|O_{out,l}| < s_l n_c$
   In this case, there must be at least two flow units i and j with $o_{end,i} = o_{end,j}$, since we used $F = n_c s_l$ flow units in total. Since $k \in K$ holds, there exists a valid flow f through the graph which exits the loop with offset k (as shown in paragraph 2). If we let one of the flow units, say i, follow that flow f instead of the flow which it

followed in the original solution, then we get a new solution to the flow problem in which $|O_{out,l}|$ is increased by 1, compared to the previous solution. Since the original solution to the flow problem must have been maximal with respect to $|O_{out,l}|$, this is a contradiction. □

Like for the global convergence approach, we can now use the graph tracking analysis as a plug-in for the offset analysis framework from Section 4.

Corollary 1 *The analysis framework, using the graph tracking analysis, provides overapproximations of the WCET of any task t executed with starting offset $O_{in,t}$ on our assumed platform.*

Proof Theorems 4 and 5 provide the missing induction step case for the proof of Theorem 2. The WCET for the f_t^{start} function is the WCET of the task. Following Theorem 2, our analysis framework together with the graph tracking analysis produces valid WCET overapproximations for this function and thus also for the task. □

Using either the global convergence or the graph tracking analysis, the analysis of tasks as a whole now only requires the offset information at the entry point of the task, which is provided by the overall analysis framework through the known processor mapping and task dependencies. All internal offset information, and with this, the WCET of the task, can then be computed through the presented framework.

5.3 Recovering from diverged offset information

In all of the presented analyses the quality of the results depends on the precision of the computed offset information. If e.g. the information is, that a block is entered with an single offset o, then uncertainties in the basic block analysis, such as variable execution times and cache accesses categorized as UNKNOWN may lead to multiple offsets which which the block may be left. Thus the information about the current offset is then less precise at the block exit. Repetitive merges of offset information, like in Algorithm 2 and 3 and in the global convergence analysis lead to further losses in offset information (still they are needed to obtain valid overapproximations). Once the analysis has reached a state, where a block b may be reached with any offset $o \in O_{max} = [0, n_c s_l - 1]$, then this will propagate to the successor blocks since $u_b(O_{max})$ (Equation 4) is equal to O_{max} then. These effects are able to slowly degenerate the offset information - we will also call this process *offset divergence*. In the following, we will provide a first step towards an improvement of the analysis which will allow us to regain knowledge about the offsets even after the offset information has diverged. To be able to recover from divergence we need to make some assumptions about the analyzed machine model.

Definition 16 A multi-core hardware platform is said to be *TDMA-composable* if the arbitration delays that occur during the accesses to a shared resource do influence neither

- the execution time of instructions which do not access the shared resource, nor
- the time that a single access to the shared resource takes (excluding the arbitration delay).

TDMA-composability in the sense given is a weaker form of *timing composability* [Wilhelm et al (2009)] which in general denotes the partial or full absence of timing anomalies. The following are examples for systems falling into the three categories:

- **Fully timing-composable** *In-order pipeline, no branch prediction, no caches*
  In such a reduced setup no timing anomalies can occur, since each instruction is executed in-order without any disturbance from other hardware units.
- **TDMA-composable** *In-order pipeline, branch prediction, private caches*
  The combination of branch-prediction / speculation and caches can cause timing anomalies as shown in [Reineke et al (2006)] but the duration of a TDMA bus access has no influence on the branch predictor or cache state of the individual cores as long as there are no shared caches. Therefore this system is TDMA-composable, but not fully timing-composable.
- **Non-composable** *Out-of-order pipeline, branch prediction, shared caches*
  In this system the execution time of basic blocks may shrink when TDMA access duration increases, since other instructions may be executed before the waiting TDMA-access instruction (out-of-order). Therefore this system is neither TDMA-composable nor fully timing-composable.

It should be noted that the fixed-alignment approach [Chattopadhyay et al (2010)] implicitly requires a TDMA-composable system in its proof of correctness. In the following, we will generalize the mechanism behind the fixed-alignment approach and show that it can be used to enhance the precision of the offset bound analyses for TDMA-composable systems.

Lemma 4 ***(Offset Relocation Lemma)*** *Given a control flow graph $G = (V,E)$ and a path P through that CFG, two executions of the path P, one starting at TDMA offset o_1 and the other starting at TDMA offset o_2 with all other states of hardware components being identical between the two executions, will lead to a divergence in execution time of at most $n_c s_l$ cycles between the two execution scenarios if a TDMA-composable platform is used.*

Proof We can view the execution of the path P as a sequence $S = (s_0, \ldots, s_i)$ where each $s_j \in P$ is either an accesses to the shared resource or a block of local computations. We therefore define a set of accesses A and a set of "processing" blocks B such that $\forall s \in S : s \in A \oplus s \in B$, where $\oplus$ is the logical *exclusive OR*. For each access $s_j \in A$ we define the time from the access request to the access grant as ω_i and the time that it takes to perform the access as γ_i. Similarly we define the runtime of each block $s_i \in B$ as α_i. Note that with our current assumptions, we know that α_i and γ_i are constant among the two execution scenarios since the system is TDMA-composable, thus the possible change in the ω_i values does not influence the execution times of the local computations or the resource access itself.

To simplify our computations below, we allow blocks in B to have length 0, that is $\alpha_i = 0$. In this way, we can ensure that each pair of accesses is separated by a block

of computation, though this block may have length 0 ($\forall j \in 0,\dots,i-1 : s_j \in A \Leftrightarrow s_{j+1} \in B$). Without loss of generality, we assume $s_0 \in A$.

The arbitration delay ω_j that an access $s_j \in A$ incurs, will change among the two execution scenarios. The arbitration delay that is incurred by access s_j in the execution scenario starting at offset o_1 (o_2) will be denoted by ω_j^1 (ω_j^2), respectively. In the same way, we will refer to the point in time at which the access request is issued in the two scenarios as β_j^1 and β_j^2. What we would like to prove now, is:

$$\forall j \in \{0,2,4,\dots\} : |\beta_j^1 - \beta_j^2| \leq n_c s_l \tag{30}$$

Note that this implies the Lemma no matter whether $s_i \in A$ or $s_i \in B$. For $s_i \in A$ we can apply Equation 30 with $j = i$ immediately, and for $s_i \in B$ the Lemma follows from Equation 30 with $j = i-1$ and the constant runtime of s_i.

To prove Equation 30 we first define the dependencies among the β and ω values, which are as follows:

$$\forall j \in \{0,2,4,\dots\} : \beta_j = \beta_{j-2} + \omega_{j-2} + \gamma_{j-2} + \alpha_{j-1} \tag{31}$$

$$\forall j \in \{0,2,4,\dots\} : \omega_j = \Phi_p(\beta_j \mod n_c s_l, \gamma_j) \tag{32}$$

In Equation 31 the access request time for s_j is computed as the sum of

- β_{j-2} - the request time of the last access (in s_{j-2})
- ω_{j-2} - the arbitration delay that this request incurs
- γ_{j-2} - the access duration itself
- and α_{j-1}, the duration of the consecutive block of computation s_{j-1}

The value for ω_j is then easily derived in Equation 32 from β_j with the usual TDMA arbitration computation via the known Φ_p function (see Equation 7). These dependencies are valid for both execution scenarios since they model the execution of the path P. The only variability comes from the accesses to the shared resource, for which the waiting time ω_j is computed using the known Φ_p function (see Equation 7). To initialize the computation, we have $\beta_j^1 = o_1$ and $\beta_j^2 = o_2$. We prove Equation 30 by induction over the sequence S. In the proof, Δ_j is used as a shorthand for $|\beta_j^1 - \beta_j^2|$.

Base case ($j = 0$): In this case, we have $\Delta_j = |o_1 - o_2| \leq n_c s_l$ by definition of β_j^1 and β_j^2.

Induction step ($j-2 \rightarrow j$): By inserting the definitions of β_j into Δ_j we obtain:

$$\Delta_j = |\left(\beta_{j-2}^1 + \omega_{j-2}^1 + \gamma_{j-2} + \alpha_{j-1}\right) - \left(\beta_{j-2}^2 + \omega_{j-2}^2 + \gamma_{j-2} + \alpha_{j-1}\right)| \tag{33}$$

$$= |\left(\beta_{j-2}^1 + \omega_{j-2}^1\right) - \left(\beta_{j-2}^2 + \omega_{j-2}^2\right)| \tag{34}$$

$$= |\left(\xi_{j-2}^1 - \xi_{j-2}^2\right)| \tag{35}$$

where ξ_j is a shorthand for $\beta_j + \omega_j$, i.e. the time when the access is granted. Obviously, Δ_j only depends on the access request times and waiting times of the preceding resource access. Via the induction hypothesis we know that $\Delta_{j-2} \leq n_c s_l$, that is, in both execution scenarios the preceding access is requested in a window of size $n_c s_l$

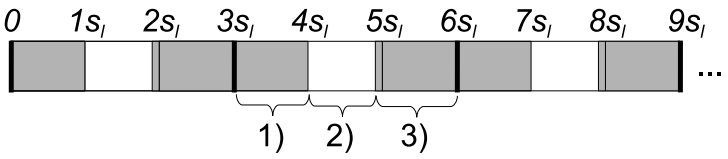


Fig. 8 Illustration of proof scenario for the Relocation Lemma.

cycles. What we then have to show is, that both accesses are granted in a new window of size $n_c s_l$ cycles. Figure 8 illustrates a scenario from the perspective of the second core in a configuration with three cores in total. Three TDMA hyperperiods are shown in the Figure. The gray areas are those where an access request from core two will have to wait, and the white ones are the areas where an access request from core two will be granted immediately. Due to the cyclicity of the TDMA schedule it is sufficient to assume that $\beta^1_{j-2} \in [3s_l, 6s_l)$ and to place β^2_{j-2} at positions which are at most $n_c s_l$ cycles away from β^1_{j-2}. We then have to prove that $\Delta_j \leq n_c s_l$ holds. We will examine the individual cases with the help of the example, since it is more intuitive. The notation could nevertheless be generalized to describe the cases in a fully abstract way (in this case only the position and size of the white boxes will change in all hyperperiods).

- $\beta^1_{j-2} \in [3s_l, 4s_l)$: This case is marked as 1) in Figure 8 and obviously we then have to wait for the core's slot, thus $\xi^1_{j-2} = 4s_l$. If β^2_{j-2} is in another gray area, then ξ^2_{j-2} will be either $1s_l$, $4s_l$ or $7s_l$. If β^2_{j-2} is in a white area, then it follows that $\xi^2_{j-2} \in \big[1s_l, 2s_l - (\gamma_{j-2} - 1)\big) \cup \big[4s_l, 5s_l - (\gamma_{j-2} - 1)\big)$. In all cases, $|\xi^1_{j-2} - \xi^2_{j-2}| \leq n_c s_l$ holds.
- $\beta^1_{j-2} \in \big[4s_l, 5s_l - (\gamma_{j-2} - 1)\big)$: In Figure 8 this case is marked as 2) and since the access can be granted immediately here, we have $\xi^1_{j-2} \in \big[4s_l, 5s_l - (\gamma_{j-2} - 1)\big)$. If β^2_{j-2} is in a gray area, then ξ^2_{j-2} will be either $4s_l$ or $7s_l$. In this case, $|\xi^1_{j-2} - \xi^2_{j-2}| \leq n_c s_l$ directly holds. For the case that β^2_{j-2} is in a white area, the Lemma follows from the induction hypothesis that $|\beta^1_{j-2} - \beta^2_{j-2}|$ since in this case $\xi_{j-2} = \beta_{j-2}$.
- $\beta^1_{j-2} \in \big[5s_l + (\gamma_{j-2} - 1), 6s_l\big)$: This case (3) in Figure 8) is symmetrical to the first one.

Since in all these cases $|\xi^1_{j-2} - \xi^2_{j-2}| \leq n_c s_l$ holds and we known from Equation 35 that $\Delta_j = |\left(\xi^1_{j-2} - \xi^2_{j-2}\right)|$, the induction step is complete and the lemma is proven. □

During the analysis, in the case that we run into a situation where we no longer have useful offset information at a basic block, we can use the Offset Relocation Lemma. To do this, we add a code block after line 12 in Algorithm 1, which does the following:

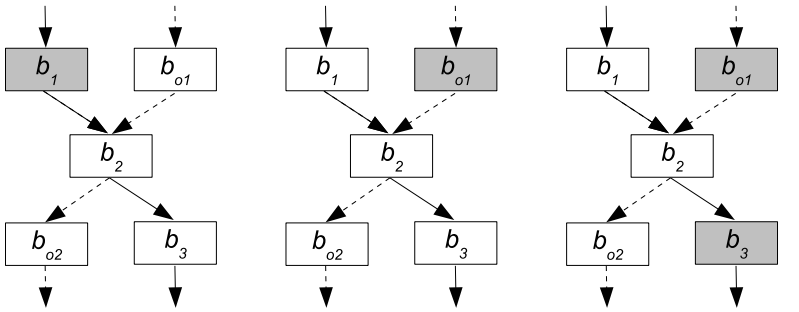
1. Check whether

  - b is a block which consists of a bus accessing instruction and
  - whether $\exists o \in O_{in}, e \in ET_b : \Phi_p(o,e) = D^{max}$ is true
2. In case that these checks succeed, set $result = \langle wcet + n_c s_l, \{s_l p\} \rangle$ (with $p \in [0,\ldots,n_c-1]$ being the index of the current core)

For the single access this heuristic will not yield an improvement in analysis precision, but successive accesses may profit from the increased precision of the offset information. This is especially useful in systems with short TDMA schedules or instructions with variable latencies etc. since these properties will lead to a quick divergence of the offset bounds. With the help of the Offset Relocation Lemma these diverged bounds can be restored with little cost. Note that this is only a heuristic, we could also apply the Lemma at arbitrary other program points. With the heuristic, we are no longer overapproximating the set of possible offsets. Nevertheless, we are still generating safe WCET overestimations and the analysis is still guaranteed to terminate, which we will both show in the following. Figure 9 shows the scenarios that we have to distinguish in the following. In each subfigure, we applied the offset relocation heuristic at the gray blocks, all white blocks have their offsets computed without offset relocation. The solid arrows mark the WCEP, whereas the dashed arrows represent control-flow edges which are not part of the WCEP $P = (b_1, b_2, \ldots, b_n)$. We consider a single application of the heuristic at a block b^A and distinguish three cases:

- The WCEP contains b^A: This case is illustrated in Figure 9a with $b^A = b_1$. Since we applied the heuristic, we added a penalty of $n_c s_l$ cycles to b^A's WCET. Thus, through Lemma 4 the correctness of the computed WCET follows.
- The WCEP does not contain b^A: This case is shown in Figures 9b and 9c with $b^A = b_{o1}$. Then, the application of the heuristic may only affect the WCET results due to merged-in offset information, when there is a path in the CFG from b^A to any node b_w on the WCEP $P = (b_1, b_2, \ldots, b_n)$, e.g. path (b^{o1}, b_2) in Figures 9b and 9c. This can only lead to additional offsets in the offset set of b_w. Again we have two cases:
  - If there is no further node $b_y \in P_{\text{tail}}$ with $P_{\text{tail}} = \{b_i | w \leq i \leq n\} \subseteq P$ where the heuristic was applied too (e.g. as in Figure 9b), then these additional offsets will only make the WCET results for the blocks in P_{tail} less precise, but they do not endanger their safeness. This is guaranteed since the conventional update function (see Equation 4), which is then applied for all blocks in P_{tail}, is linear and thus all original offsets, coming from blocks on the WCEP, are conserved.
  - If such a node b_y does exist (e.g. b_3 in Figure 9c), then the application of the heuristic renders the offset set with which b_y is reached irrelevant, since the offset result for b_y will be $\{s_l p\}$ anyways. Therefore the correctness of the WCET also follows in this case.

Note that the heuristic application of the Offset Relocation Lemma breaks the linearity of the update function. As an example for this, consider a basic block b with a guaranteed bus access of 2 cycles duration in a system with 2 cores and slot length 5. From the point of view of the first core, $u_b(\{3,5\}) = \{5,2\}$ whereas $u_b(\{0-9\}) = \{2\}$ with applied relocation to offset 0. This non-linearity can lead to effects, where a more detailed representation and analysis, e.g. offset-set-based,



(a) Relocation applied on WCEP

(b) Relocation applied outside of WCEP

(c) Relocation applied outside of and on WCEP

Fig. 9 Different scenarios for applying the offset relocation heuristic

can have worse results than a less detailed one, e.g. offset-interval-based. Example 6 shows this counter-intuitive behavior on a simplified real-world CFG from the Papabench benchmark.

Both the Global Convergence and the Graph Tracking analysis are still guaranteed to terminate when used with the offset relocation heuristic. The original termination arguments - the finiteness of the set of possible offsets and the finiteness of the offset graph, respectively - are not affected by the heuristic and thus still hold.

Example 6 Such a counter-intuitive result, triggered by the non-linearity, is shown in Figure 10. The code shown between the dashed horizontal lines could for example be a loop body that is analyzed repeatedly during the global convergence loop offset analysis. In this case, the upper half of the figure shows the analysis of the first iteration and the lower half of the figure shows the analysis of the second iteration of the loop. The analysis run A (shown in light gray on the left side) starts with the information that the top block may be entered with offsets $\{0,\ldots,9\}$ and obtains a WCET of $14+10(X-1)$ if the relocation heuristic is applied. X is the loop bound of the analyzed loop in this case. In contrast, the analysis run B (shown in dark gray on the right side) starts with the offset information $\{2\}$ which is obviously more precise than the information that run A has and obtains a WCET of $12X$ if the relocation heuristic is not applied. Obviously for all $X \geq 3$ analysis run A will produce a more precise WCET result than analysis run B, though it started with more imprecise offset information.

This shows that if we allow the application of the Offset Relocation Heuristic in the analysis of a TDMA-composable system, it may happen that we obtain worse results for more precise input offsets. Thus, in these analyses it may be not worthwhile to maintain a highly-precise offset representation as e.g. offset sets since they may even worsen the results. Note again that the correctness of the analysis is not endangered by this phenomenon. As we have shown above, this is a mere observation on the precision of the analysis.

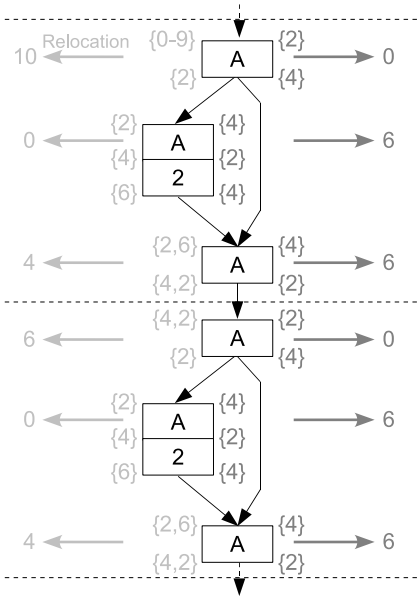


Fig. 10 An example of counter-intuitive analysis results due to the application of the relocation lemma heuristic in a 2-core system with sloth length 5.

5.4 Offset analysis in architectures without timing anomalies

Timing anomalies are a phenomenon which complicates WCET analysis. According to the definition from [Reineke and Sen (2009)] a system shows timing anomalies whenever local worst-case behavior does not necessarily lead to global worst-case behavior, thus for example whenever a cache hit instead of a cache miss does trigger the global worst-case behavior. This may be the case e.g. on systems with instruction prefetching and speculative execution [Reineke et al (2006)]. In the static analysis of systems with timing anomalies it is not feasible to prune the search space of the analysis [Reineke and Sen (2009)]. Therefore, in a cache analysis for a system exposing timing anomalies we may not assume an UNKNOWN access to be a cache miss (AM), but instead we must then consider both possibilities, a hit and a miss, in the analysis. On systems without timing anomalies (termed *fully timing-composable*) we can safely assume the local worst-case (AM) to increase the analysis performance and precision.

Timing composability can be seen as a stronger form of TDMA-composability, since it requires all hardware components to have local, bounded effects on the timing. We can exploit timing composability in a similar way as TDMA-composability. In a fully timing-composable system, we may reduce the offset result of any update operation to the offset o which is reached by the local worst-case path to increase the analysis precision, because this local worst-case path *must* be part of the global worst-case path. This does result in a non-linear update function just like in

the TDMA-composable case but counter-intuitive analysis effects (see end of Section 5.3) are not very likely to happen, since the non-linearity is not associated with an extra cost here.

5.5 Extensions for further micro-architectural analyses

In an analysis that includes more microarchitectural features like branch prediction and pipelining, the computed overapproximations of the hardware states must become part of the analysis context, in addition to the offset information. For the global convergence analysis, this means that a global overapproximation of the hardware states at the loop header is built and used in the analyses. For the offset graph, every context node must be annotated with an overapproximation of the hardware states with which the node may be entered, including cache, pipeline and branch prediction states. In such a scenario, the graph must be iteratively refined until

1. no more edges are added, and
2. the hardware states on all nodes have converged.

Alternatively, it is also possible to construct only a single, global overapproximation of the hardware states, depending on which degree of precision is required.

6 Experimental Results

In the following, the different approaches to bus-aware WCET analysis are compared. As mentioned, we have implemented our approaches based upon the code from [Chattopadhyay et al (2010)] which enables a precise comparison. The prototype tool analyzes executables compiled for the SIMPLESCALAR platform and includes a thorough cache analysis. Unfortunately, no pipeline or branch prediction analysis is integrated, so all instruction latencies are set to 1 cycle. Section 5.5 nevertheless introduced the general concept of how to perform such an integration. It can be expected that the classification of the approaches with respect to precision and analysis time stays the same even after additional microarchitectural analyses were integrated, since the number of analysis contexts is directly dependent on the analysis type as explained in Section 4.2. The number of contexts in turn has the biggest influence on the analysis precision and duration. All experiments were run on an Intel Xeon E5630 machine with 8 cores at 2.53GHz and 20GB of main memory under Debian Linux. Concerning the solution of the dynamic flow problems during the graph tracking analysis, we used the CPLEX ILP solver in the experiments.

6.1 Implementation tweaks

Some points in the implementation were adapted for best performance and precision. Since they do not contain any new concepts but nevertheless have an influence on the analysis time and precision, we just list them here briefly.

- **Graph tracking ILP timeout**
  For the graph tracking approach we specified an ILP timeout of 60 seconds. The offset ILPs are very hard to solve for those cases where all edges in the offset graph have nearly the same WCET, since the ILP branch and bound algorithm then cannot prune any part of the search tree. In addition, the obtained precision for those cases is not much better than the precision of the global convergence approach. Therefore, we revert to global convergence if the time limit is exceeded. In addition, if a timeout occurs for a loop L analyzed with incoming offsets O_{timeout}, we store $|O_{\text{timeout}|}$ and if L is analyzed again with incoming offsets O_{other} where $|O_{\text{other}}| \geq |O_{\text{timeout}}|$, then we use the global convergence immediately.
- **Detailed graph tracking**
  During the graph tracking analysis (compare Section 5.2) we stated that for each construction step we add all edges $e \in O_{in} \times O_{out}$ with $c(e) = wcet_l^{LB}(O_{in})$. To increase the precision, we construct the edges exiting from each input offset node in separation. For each offset $o_{in}^k \in O_{in}$ we add all edges $e \in \{o_{in}^k\} \times u_l^{LB}(o_{in}^k)$ with $c(e) = wcet_l^{LB}(o_{in}^k)$. This turned out to be especially useful in the case of the analysis for non-timing-composable systems, since it may give more precise WCET results for the individual edges, even when offsets have diverged.

In the following, we will compare against the fixed-alignment approach and the fully-unrolling approach as mentioned in Section 4.2. These were implemented in the same framework as the global convergence and graph tracking approaches. The fixed-alignment approach will use Algorithm 2 once per cache context and add the respective penalties, whereas the fully-unrolling approach will use Algorithm 2 repeatedly by feeding the offset results from iteration i as input offsets into the analysis of iteration $i+1$ until $i = B_l^{max}$. In this way, the resulting runtimes are comparable and are not influenced by different implementations of the analyses.

6.2 Benchmarks

The experiments were performed on multiple benchmark sets. On the one hand we used a subset of the `MRTC` test bench [Mälardalen WCET Research Group (2012)] which is a popular test bench for WCET analysis, but only has single-task, single-core benchmarks. To build a multi-core scenario with these benchmarks, we mapped each `MRTC` test case $i \in [0,23]$ from Table 1 to core $(i \bmod n_c)$ with priority i, where 0 is the highest priority. On the other hand, we also tested the presented algorithms with the `PapaBench` [Nemer et al (2006)] and `Debie` [European Space Agency (2012)] benchmarks which are an unmanned aerial vehicle control software and a space debris monitoring software, respectively. These are multi-task benchmarks for which we mapped the tasks to the individual cores manually. The default system configuration is a 2-core system with 1KB L1 cache (direct-mapped, block size 32 bytes) and 2KB L2 cache (4-way set-associative, block size 64 bytes). Only for `Debie`, the cache configuration was changed to 2KB L1 cache (2-way set-associative) and 8KB L2 cache to account for the bigger program sizes of `Debie`. In any case, the L1 hit

Benchmark	LOC	s_{byte}	L	D	$\varnothing_B$
adpcm	468	12480	18	0	69
bs	79	480	1	0	4
bsort100	74	1024	3	1	100
cnt	72	1552	4	1	40
cover	228	9312	3	0	60
crc	66	1936	3	0	102
edn	196	8000	11	2	61
fdct	148	5088	2	0	8
fft	97	8368	7	2	88
fir	188	1056	2	1	375
insertsort	20	752	2	1	10
jfdcint	165	5424	3	0	26
lms	146	4576	10	0	50
ludcmp	71	4544	11	2	4
matmult	81	1536	5	2	20
mergesort	266	9152	23	3	126
minver	135	6160	17	2	2
ndes	201	6256	12	0	19
nsichneu	2362	63632	1	0	2
qurt	87	1952	1	0	19
select	55	3120	4	2	8
sqrt	42	912	2	0	12
st	98	60528	1	0	1000
statemate	1128	11728	4	0	20
Debie	24528	1622912	39	0	157
PapaBench	4663	200256	10	0	3

Table 1 Benchmark properties

penalty is 0 cycles, the L2 hit penalty is 1 cycle and the main memory access time is 5 cycles modeling a Flash-based main memory. A more detailed overview of the used benchmarks is provided in Table 1, including the lines of code LOC (excluding comments and empty lines), the byte size s_{byte} of the "text" section of the executable (excluding startup code), the number of loops L, the maximum loop nesting level D and the average loop bound $\varnothing_B$. The `Debie` and `PapaBench` benchmarks consist of 35 resp. 32 individual tasks which have a relatively simple structure, especially since they have no nested loops.

In the following sections, we will have to distinguish 3 different analysis scenarios:

- *C-* Fully timing-composable system
- *C#* TDMA-composable system
- *C+* Non-composable system

For the *C-* (*C#*) case we may apply the reduction to the worst case (Section 5.4) or the relocation lemma heuristic (Section 5.3), respectively, which is expected to improve the precision of the results.

Additionally, we have to identify the different analysis methods which we will name in the following way:

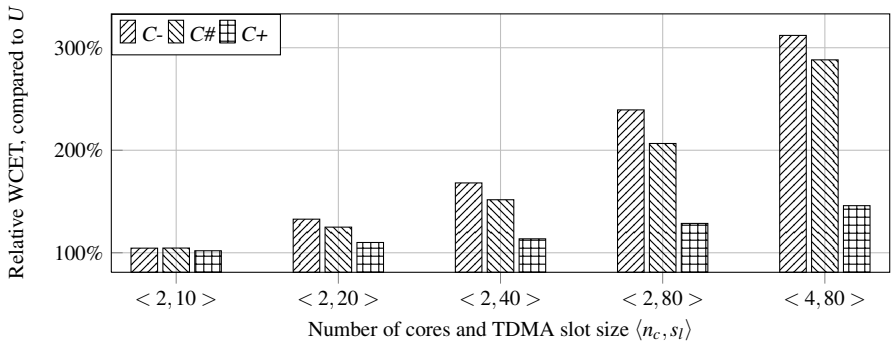


Fig. 11 Average WCET results when using the worst-case assumption (analysis *W*)

W	Assume worst-case bus access delay of D^{max} cycles for each bus access
F	Fixed alignment [Chattopadhyay et al (2010)]
OC	Offset analysis (Global convergence)
OT	Offset analysis (Graph tracking)
U	Full virtual unrolling

As a last step, for each analysis we have the choice to run it using offset intervals (*RI*) or offset sets (*RS*). With these 3 components we can form a naming scheme to identify experimentation scenarios like e.g. *C#/OC/RI* for an analysis in a TDMA-composable system with the global convergence analysis based on offset intervals. When presenting WCET results in the following, we will always present the *relative WCET*, i.e. the ratio of the WCET obtained with the currently examined analysis and of the WCET obtained with analysis *U*. In addition we will not present results for the individual benchmarks here, but average values obtained by building the average of the relative WCETs of all benchmarks from Section 6.2.

6.3 Maximum overestimation

To keep the following diagrams more readable, we first analyze how much impact the overestimation of the TDMA access duration can have. Therefore, we first present the relative WCETs obtained with analysis *W* in Figure 11. The y-axis shows the average relative WCETs, and the x-axis shows different configurations of core number and slot length. The results clearly indicate that an analysis of the bus access times is needed if there is any significant amount of sharing in the multi-core system since always reverting to worst-case assumptions may lead to overestimations of more than 200%. The fact that the overestimation with respect to *U* is decreased for less predictable composability classes (*C#*,*C+*) is due to the fact that the results for *C#/U* and *C+/U* are less precise themselves, whereas the results for *W* stay the same, irrespective of the composability class.

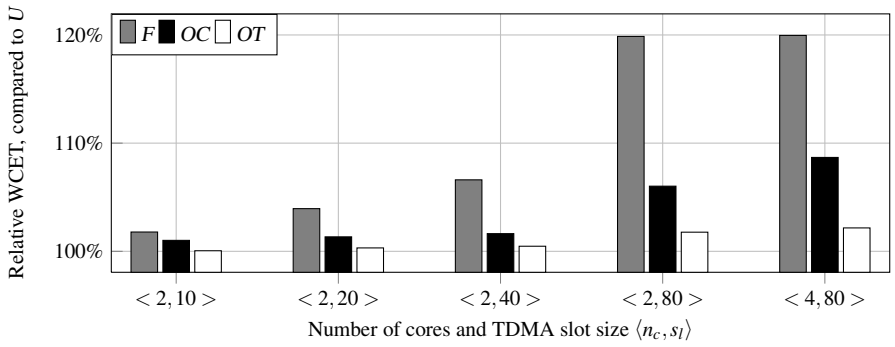


Fig. 12 Average WCET results for a fully timing-composable system (*C-/RI*)

6.4 Fully timing-composable system

Figure 12 shows the average relative WCETs analyzed for a fully timing-composable system, i.e. with the extension sketched in Section 5.4. These extensions allow a more precise analysis already, but still the advantage of *OC* and *OT* over *F* ranges between 1.5% (configuration $\langle 2,10 \rangle$) and 17.5% (configuration $\langle 2,80 \rangle$). Using *RS* instead of *RI* only results in a insignificant improvement of less than 1% here, due to the mentioned inherent precision of the extensions from Section 5.4. We therefore do not list results for *RS* in Figure 12 to increase the readability. The analysis times as measured on a single core (without exploiting parallelism) are given in Table 2a. Time results for *RS* are not given, since as indicated, it does not yield notable WCET improvements for this composability class. The presented time values only show the WCET analysis time excluding other analysis phases like CFG reconstruction and cache analysis. It becomes apparent that *OC* is just as fast as *F* while being far more precise. *U* and *OT* are slower, and up to configuration $\langle 2,40 \rangle$ *OT* outperforms *U* in matters of analysis time. For the bigger schedule lengths, the ILP solving times dominate the analysis runtime and make *OT* less attractive than *U*. The analysis time of all analyses increases with the TDMA schedule length since the time needed for the u_b function evaluations (see Equation 4) in "AnalyzeBlock" (Algorithm 1) is dependent on the number of offsets in the incoming offset set. If there is no clear local worst-case path, i.e. if all input offsets lead to the same local WCET, then we must track all possibilities even in the fully timing-composable case. For longer TDMA schedule lengths these sets are getting bigger then, which implies that more time is needed for the "AnalyzeBlock" stages. This is the reason why the analysis time grows even for the *U* and *F* cases.

6.5 TDMA-composable system

The *C#* case, as shown in Figure 13 and Table 2b, shows increased overestimation ratios and analysis times compared to the *C-* case. The analysis times for *C#/OT/RI* as shown in Table 2b are partially smaller than those for *C-/OT/RI*, e.g. for configura-

$< n_c, s_l >$	U	F	OC/RI	OT/RI
$< 2,10 >$	3.93	0.00	0.00	0.28
$< 2,20 >$	4.92	0.01	0.01	0.50
$< 2,40 >$	7.15	0.01	0.02	2.42
$< 2,80 >$	10.38	0.04	0.04	26.62
$< 4,80 >$	13.46	0.16	0.20	60.29

(a) Fully composable (*C-*)

$< n_c, s_l >$	U	F	OC/RI	OT/RI
$< 2,10 >$	4.35	0.00	0.00	0.53
$< 2,20 >$	4.32	0.01	0.01	0.57
$< 2,40 >$	6.25	0.01	0.02	3.95
$< 2,80 >$	10.99	0.05	0.05	25.38
$< 4,80 >$	17.00	0.19	0.20	54.99

(b) TDMA-composable (*C#*)

$< n_c, s_l >$	U	OC/RI	OC/RS	OT/RI	OT/RS
$< 2,10 >$	9.65	0.00	0.00	0.30	1.85
$< 2,20 >$	20.29	0.01	0.01	2.95	19.16
$< 2,40 >$	35.30	0.02	0.02	16.19	68.78
$< 2,80 >$	57.00	0.07	0.08	22.88	148.62
$< 4,80 >$	79.50	0.24	0.23	29.52	159.41

(c) Non-composable (*C+*)

Table 2 Aggregated sequential WCET analysis times (runtime on a single core) for all examined scenarios (in minutes)

tions $\langle 2,80 \rangle$ and $\langle 4,80 \rangle$, which may be surprising at the first sight since the generated ILPs certainly are not simpler for *C#*. The reason for this is the ILP solver timeout mentioned in Section 6.1 which occurs more often for the *C#* instances, such that the analysis has to revert to the faster global convergence method more often than in the *C-* case.

The expectation that was associated with TDMA-composability was, that for both precision and analysis time it will range in between *C-* and *C+*. Only in this case TDMA-composability can be useful, since then it will allow a better analysis for systems which cannot be classified as fully timing-composable. We cannot verify this expectation in Figure 13 since it only presents WCET values which are relative to *C#/U*, but the results for *C#/U* already use the offset relocation lemma and are thus different from *C-/U*. Therefore, Figure 14 shows the sum of the *absolute* WCET results for all examined benchmarks for analysis *U* in different composability classes which allows to compare the attainable estimation precision of the different classes against each other. Because of the big differences in the absolute values, the y-axis is scaled logarithmically. Obviously, the absolute WCET values which are determined for *C#* in Figure 14 are much closer to the results of *C-* than to those of *C+*. This shows the potential of TDMA-composability to improve the analysis results for systems which are not fully timing-composable.

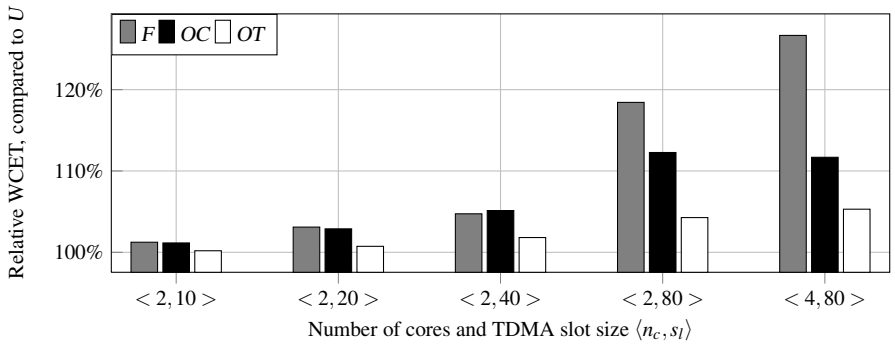


Fig. 13 Average WCET results for a TDMA-composable system (*C#/RI*)

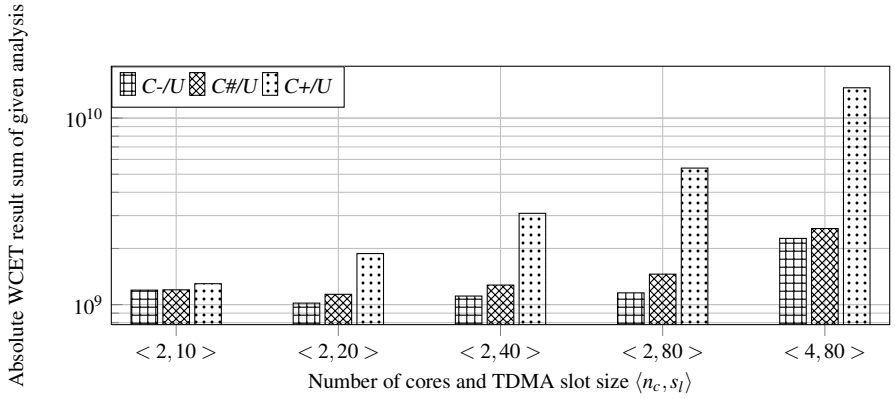


Fig. 14 Absolute WCET result sums for all composability classes

Surprisingly, using *RS* in the *C#* case is even slightly worse than *RI* which can be explained with the precision anomaly explained in Example 6. Therefore, we again do not list the results for *RS* in Figure 13 and Table 2b to keep them more readable.

6.6 Non-composable system

The WCET results for non-composable systems are given in Figure 15 and Table 2c. As stated in Section 5.3, analysis *F* is not applicable here, since it requires TDMA-composable or fully timing-composable systems. *OC/RI* delivers results with an overestimation of $0.8\% - 36.4\%$, which drops to $0.1\% - 7.9\%$ for *OT/RI*. Here, the set representation *RS* really improves the results such that *OT/RS* is 10.2% *more* precise than *U* in the case $\langle 2, 80 \rangle$ and 7.5% more precise than *U* on average. This is possible, since the unrolling has to merge the offset information after each unrolled loop iteration, whereas the graph with the extensions from Section 6.1 features a precise WCET for each single source offset and is able to track their development in more

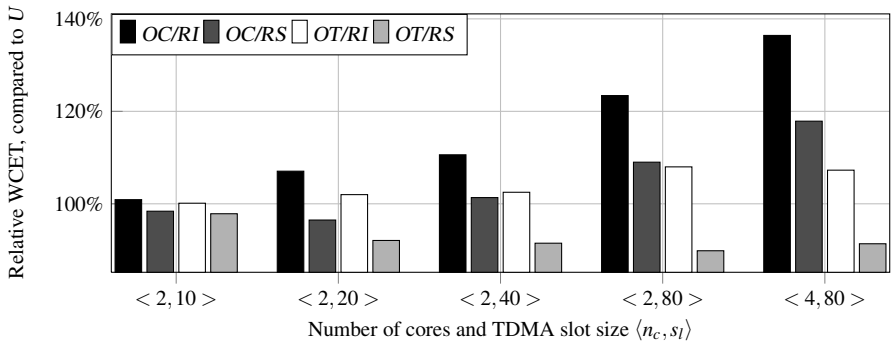


Fig. 15 Average WCET results for a non-composable system (*C+/RI*)

detail. Concerning the runtimes of *C+/U*, *C+/OT/RI* and *C+/OT/RS*, except for the $\langle 2, 10 \rangle$ configuration they are probably only feasible for nightly-build analysis or with parallel computations since the ILP solving can be easily parallelized and also different tasks can be analyzed in parallel. The fact that the runtimes for *C+/OT/RI* are less than those for *C#/OT/RI* in some configurations is again due to the ILP timeout as already mentioned in Section 6.5.

7 Conclusions

We have reviewed and extended an approach to the WCET analysis of TDMA-arbitrated shared resources which uses a static analysis of TDMA offsets and cyclic contexts to determine maximum access delays. We have formally proven the correctness of this analysis and have introduced a new system classification called *TDMA composability* which enables more precise analyses for systems which belong to this class, but are not fully timing-composable. The improved precision of this class, compared to non-composable systems, was empirically shown. We presented a detailed evaluation of the given analysis methods in different composability classes with a broad selection of real-world benchmarks and pointed out that different offset representations have a strong impact on the achievable precision and analysis time. For fully timing-composable and TDMA-composable systems, the interval representation showed superior results, whereas for non-composable systems the set representation achieves higher precision. The experiments show that the inspected TDMA access delay analysis methods are able to work as fast as the quickest known method, while delivering results which are 0.8% to 21.7% more precise on average, depending on the analyzed system. When precision is the main objective we are even able to outperform the most precise known analysis by up to 10.2% in the case of non-composable systems.

Future plans include the integration of further microarchitectural analyses, specialized flow algorithms for the graph tracking approach and experiments with different platforms.

Acknowledgements This work was partially funded by the European Community's ArtistDesign Network of Excellence, by the European Community's 7^{th} Framework Program FP7/2007-2013 under grant agreement n^{o} 216008, by the German Research Foundation DFG under reference number FA1017/1-1 and by Faculty Research Council grant T1 251RES0914 (R-252-000-416-112) at NUS.

References

[Aho et al (2006)] Aho AV, Lam MS, Sethi R, Ullman JD (2006) Compilers: Principles, Techniques, and Tools, 2nd edn. Addison-Wesley

[Altmeyer et al (2010)] Altmeyer S, Maiza C, Reineke J (2010) Resilience Analysis: Tightening the CRPD Bound for Set-Associative Caches. In: LCTES '10: Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems, ACM, New York, NY, USA, pp 153–162, DOI 10.1145/1755888.1755911, URL http://rw4.cs.uni-saarland.de/ reineke/publications/ResilienceAnalysisLCTES10.pdf

[Andrei et al (2008)] Andrei A, Eles P, Peng Z, Rosen J (2008) Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In: Proceedings of the 21st International Conference on VLSI Design, IEEE Computer Society, Washington, DC, USA, VLSID '08, pp 103–110

[Chattopadhyay et al (2010)] Chattopadhyay S, Roychoudhury A, Mitra T (2010) Modeling shared cache and bus in multi-cores for timing analysis. In: Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems, ACM, New York, NY, USA, SCOPES '10, pp 6:1–6:10

[Cousot and Cousot (1979)] Cousot P, Cousot R (1979) Systematic design of program analysis frameworks. In: Proceedings of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), ACM Press, New York, NY, San Antonio, Texas, pp 269–282

[European Space Agency (2012)] European Space Agency (2012) DEBIE – First Standard Space Debris Monitoring Instrument. https://gate.etamax.de/edid/publicaccess/debie1.php

[Gustavsson et al (2010)] Gustavsson A, Ermedahl A, Lisper B, Pettersson P (2010) Towards WCET Analysis of Multicore Architectures Using UPPAAL. In: 10th International Workshop on Worst-Case Execution Time Analysis, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, WCET '10, pp 101–112

[Hardy and Puaut (2008)] Hardy D, Puaut I (2008) WCET Analysis of Multi-level Non-inclusive Set-Associative Instruction Caches. In: Proceedings of the 2008 Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, pp 456–466

[Hardy et al (2009)] Hardy D, Piquet T, Puaut I (2009) Using Bypass to Tighten WCET Estimates for Multi-Core Processors with Shared Instruction Caches. In: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium, IEEE Computer Society, Washington, DC, USA, RTSS '09, pp 68–77

[Kelter et al (2011)] Kelter T, Falk H, Marwedel P, Chattopadhyay S, Roychoudhury A (2011) Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds. In: Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS), Porto / Portugal, pp 3–12

[Lv et al (2010)] Lv M, Guan N, Yi W, Yu G (2010) Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In: 31st IEEE Real-Time Systems Symposium (RTSS)

[Mälardalen WCET Research Group (2012)] Mälardalen WCET Research Group (2012) Mälardalen WCET Benchmark Suite. http://www.mrtc.mdh.se/projects/wcet

[Mische et al (2010)] Mische J, Guliashvili I, Uhrig S, Ungerer T (2010) How to Enhance a Superscalar Processor to Provide Hard Real-Time Capable In-Order SMT pp 2–14

[Muchnick (1997)] Muchnick SS (1997) Advanced Compiler Design and Implementation. Morgan Kaufmann

[Nemer et al (2006)] Nemer F, Cassé H, Sainrat P, Bahsoun JP, Michiel MD (2006) PapaBench: a Free Real-Time Benchmark. In: Mueller F (ed) 6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany

[Paolieri et al (2009)] Paolieri M, Quiñones E, Cazorla FJ, Bernat G, Valero M (2009) Hardware support for WCET analysis of hard real-time multicore systems. In: Proceedings of the 36th annual international symposium on Computer architecture, ACM, New York, NY, USA, ISCA '09, pp 57–68

[Pellizzoni et al (2010)] Pellizzoni R, Schranzhofer A, Chen JJ, Caccamo M, Thiele L (2010) Worst case delay analysis for memory interference in multicore systems. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE '10, pp 741–746

[Pitter and Schoeberl (2010)] Pitter C, Schoeberl M (2010) A real-time Java chip-multiprocessor. ACM Transactions on Embedded Computing Systems 10:9:1–9:34

[Reineke and Sen (2009)] Reineke J, Sen R (2009) Sound and Efficient WCET Analysis in the Presence of Timing Anomalies. In: Holsti N (ed) 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Germany, Dagstuhl, Germany

[Reineke et al (2006)] Reineke J, Wachter B, Thesing S, Wilhelm R, Polian I, Eisinger J, Becker B (2006) A definition and classification of timing anomalies. In: Proceedings of 6th International Workshop on Worst-Case Execution Time (WCET) Analysis

[Skutella (2009)] Skutella M (2009) An Introduction to Network Flows Over Time. Research Trends in Combinatorial Optimization pp 451–482

[Suhendra and Mitra (2008)] Suhendra V, Mitra T (2008) Exploring locking & partitioning for predictable shared caches on multi-cores. In: Proceedings of the 45th annual Design Automation Conference, ACM, New York, NY, USA, DAC '08, pp 300–303

[Wilhelm et al (2008)] Wilhelm R, Engblom J, Ermedahl A, Holsti N, Thesing S, Whalley D, Bernat G, Ferdinand C, Heckmann R, Mitra T, Mueller F, Puaut I, Puschner P, Staschulat J, Stenström P (2008) The worst-case execution-time problem — Overview of methods and survey of tools. ACM Transactions on Embedded Computing Systems 7:36:1–36:53

[Wilhelm et al (2009)] Wilhelm R, Grund D, Reineke J, Schlickling M, Pister M, Ferdinand C (2009) Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-Critical Embedded Systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28(7):966–978

[Zhang and Yan (2009)] Zhang W, Yan J (2009) Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches. IEEE Computer Society, Los Alamitos, CA, USA, vol 0, pp 455–463