technische universität
dortmund

fakultät für
informatik

TECHNICAL REPORTS IN COMPUTER SCIENCE

Technische Universität Dortmund

Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds (Extended Version)

Timon Kelter, Heiko Falk, Peter Marwedel

Chair XII, TU Dortmund University, Germany

Sudipta Chattopadhyay, Abhik Roychoudhury

National University of Singapore

Number: 837
January 2011

Technische Universität Dortmund — Fakultät für Informatik
Otto-Hahn-Str. 14, 44227 Dortmund, Deutschland

Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay, Abhik Roychoudhury: *Bus-Aware Multicore WCET Analysis through TDMA Offset Bounds (Extended Version)*, Technical Report, Department of Computer Science, TU Dortmund University.

ABSTRACT

In the domain of real-time systems, the analysis of the timing behavior of programs is crucial for guaranteeing the schedulability and thus the safeness of a system. Static analyses of the *WCET* (Worst-Case Execution Time) have proven to be a key element for timing analysis, as they provide safe upper bounds on a program's execution time. For single-core systems, industrial-strength WCET analyzers are already available, but up to now, only first proposals have been made to analyze the WCET in multicore systems, where the different cores may interfere during the access to shared TDMA-arbitrated resources. An important example for this are shared buses which connect the cores to a shared main memory. The time to gain access to the shared bus may vary significantly, depending on the used bus arbitration protocol and the access timings. In this report, we propose a new technique for analyzing the duration of accesses to shared buses. We implemented a prototype tool which uses the new analysis and tested it on a set of realworld benchmarks. Results demonstrate that our analysis achieves the same precision as the best existing approach while drastically outperforming it in matters of analysis time.

ZUSAMMENFASSUNG

Im Bereich der Realzeitsysteme ist die Analyse des Zeitverhaltens von Systemen von besonderer Wichtigkeit, um die Planbarkeit und die Sicherheit eines Systems zu gewährleisten. Statische Analysen der *WCET* (Worst-Case Execution Time) haben sich als Schlüsselelement dieser Zeitverhaltensanalyse erwiesen, da sie sichere, von konkreten Eingaben unabhängige, obere Schranken für die Laufzeit von Programmen berechnen können. Für Einzelprozessorsysteme existieren bereits industriell eingesetzte WCET-Analysatoren. Für Multiprozessorsysteme wurden bisher einzelne Vorschläge bezüglich der WCET-Analyse gemacht, allerdings fehlen hier vollwertige Lösungen. Das grundlegende Problem für die Analyse von Multiprozessorsystemen ist, daß Zugriffe auf gemeinsam genutzte Ressourcen, wie z.B. Busse zwischen CPU und Hauptspeicher, miteinander kollidieren können. Diese Kollisionen müssen aufgelöst werden, wobei nur ein einzelner Zugriff direkt stattfinden kann. Die restlichen Zugriffe aus der Kollision müssen warten bis die Ressource wieder freigegeben wurde. Diese Wartezeit kann erheblich variieren, je nachdem welcher Zuteilungsalgorithmus für die Verwaltung der Ressource verwendet wird. In diesem Bericht präsentieren wir eine neuartige Analyse, die die Dauer der Wartezeiten von Zugriffen auf nach dem TDMA-Verfahren verwalteten Ressourcen sicher abschätzen kann. Das neue Analyseverfahren wurde in einem Prototypen implementiert und auf einem Satz von industriell eingesetzten Programmen getestet. Die Ergebnisse zeigen, daß unsere neuartige Analyse dieselbe Genauigkeit aufweist wie der beste bisher existierende Ansatz aber nur einen Bruchteil von dessen Analysezeit benötigt.

ACKNOWLEDGMENTS

This work was partially funded by the European Community's Artist-Design Network of Excellence and by the European Community's 7^{th} Framework Program FP7/2007-2013 under grant agreement n^{o} 216008.

CONTENTS

1 Introduction 1

2 Related Work 3

3 System and Application Model 5
- 3.1 Modeled Hardware 5
- 3.2 Input program model 6

4 Analysis Framework 9

5 Static Analysis of TDMA Offsets 11
- 5.1 Abstract interpretation in timing analysis 11
- 5.2 Abstract hardware states and contexts 16

6 Computing Loop Offset Bounds 17
- 6.1 Determination of offset results for single iterations 18
- 6.2 Deriving full loop WCETs 22
  - 6.2.1 Global Convergence Analysis 23
  - 6.2.2 Graph Tracking Analysis 24
- 6.3 Offset analysis in architectures without timing anomalies 31
- 6.4 Extensions for further micro-architectural analyses 31

7 Experimental Results 33
- 7.1 Precision gain 35
- 7.2 Analysis time 37

8 Conclusions 41

1 INTRODUCTION

With the rising importance of multicore 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 [21], resulting in a variety of commercially available tools. In contrast, for multicores only first proposals exist. One of the major difficulties in analyzing the WCET for multicore 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 delay which can be incurred by a single access. As we will show, this bound is only a rough overestimation. In this report, we present a new type of analysis which safely bounds the access time for TDMA-arbitrated resources with high precision and moderate analysis times thus enabling a tighter WCET estimation. The results can be used to avoid pessimistic hardware overdimensioning and to derive tighter system schedules.

The rest of this report is organized as follows: In Section 2, we will present related work, Section 3 introduces our system model used in the analyses and Section 4 and 5 introduce the overall analysis framework as well as the general analysis concepts respectively. Section 6 presents our new analyses which are evaluated in Section 7. Finally, we provide a summary of our results and give directions for future work in Section 8.

2 RELATED WORK

The first approaches to multicore WCET analysis only modeled the shared resources to some extent. Suhendra [20] and Zhang [22] analyzed the effects of a shared L2 cache without considering the interference on a shared bus that is used to access the shared cache. [22] provides a bound on the number of additional cache misses due to the inter-core interference, whereas [20] eliminates the interference altogether by exploring different scenarios of locking and partitioning the shared cache. A similar approach is pursued by Hardy [7], where cache bypassing is used to eliminate the cache conflicts between different cores.

Gustavsson [6] investigates a totally different approach, where the whole multicore 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 [9] 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 [16] compared the predominant arbitration methods and TDMA arbitration resulted as the most predictable method. Therefore, most of the works which include a bus analysis are restricted to TDMA bus arbitration. To provide a better access time estimation than the mentioned D^{max} cycles, Andrei [2] tries to determine the precise time at which every single memory access takes place. The bus 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 [3] 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 [3] is far more efficient but also less precise than [2]. The analysis which we propose in the following, will present a com-

promise between the two approaches, being almost as precise as [2] and only slightly less efficient than [3].

Finally, Pellizzoni [15] derives the worst-case bus delays in a multicore 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 static timing analysis is the adaption of multicore hardware to exhibit better predictability properties. Paolieri [14] proposed a multicore architecture in which the WCET of basic blocks is measurable, whereas Mische [11] developed a superscalar SMT processor, which provides built-in real-time capabilities. These approaches are orthogonal to ours since we focus on estimating the WCET of tasks on existing hardware platforms.

3

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.

3.1 MODELED HARDWARE

We assume a system architecture where $n_c \geq 2$ cores are present in a single processor. Each of the cores has an in-order pipeline and a private L1-Cache and 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 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 3.1 for 2 cores, where core 1 issues a request "ACC" which gets delayed by D^{max} cycles.

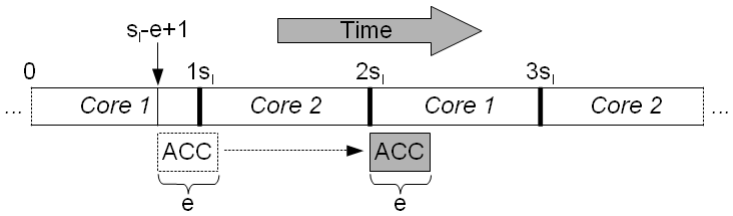


Figure 3.1: An example for a bus access which is maximally delayed.

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 3.1 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 explained in the introduction, this bound is not tight in general. Due to $T^{max} \leq s_l$ and $D^{max} \geq ((n_c - 1)s_l)$, the maximum bus delay will at least be $(n_c - 1)$ times as big as the maximum memory latency. 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 report 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 report.

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 [8] 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.

3.2 INPUT PROGRAM MODEL

While Section 3.1 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 5.

Definition 3. A *function* $f = (B_f, b^f_{\text{start}}, B^{\text{exit}}_f)$ 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 a set of exit blocks $B^{\text{exit}}_f \subseteq B_f$ such that there is no possible flow of control from any $b_1 \in B_f$ to a block $b_2 \in B^{\text{exit}}_f$.

For each loop L in the tasks, we require the minimum and maximum loop iteration counts B^{min}_L and B^{max}_L to be given.

Definition 4. The *intraprocedural control flow graph* $G_f = (B_f, E_f, v^{\text{source}}_{G_f})$ of a function f consists of all basic blocks B_f which constitute the function. The *source node* of the graph is $v^{\text{source}}_{G_f} = b^f_{\text{start}}$. For each possible flow of control from $b_1 \in V_f$ to $b_2 \in V_f$ there is an edge $(b_1, b_2) \in E_f$. This implies that:

$$| \left\{ e | e = (b_x, b_y) \wedge b_x \in B^{\text{exit}}_f \wedge e \in E_f \right\} | = 0 \tag{2}$$

Definition 5. A *task* $t = (F_t, f^t_{\text{start}})$ consists of a set of functions F_t and a start function f^t_{start} (usually called `main`) which is executed when the task starts.

Definition 6. The *interprocedural control flow graph* $G_t = (V_t, E_t, v^{\text{source}}_{G_t})$ of a task t has

$$V_t = \bigcup_{f \in F_t} B_f \tag{3}$$

$$E_t = \bigcup_{f \in F_t} E_f \cup E_{\text{call}} \tag{4}$$

where for every call from 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 $(b_{\text{call}}, b^{f_2}_{\text{start}}) \in E_{\text{call}}$. The source node of the graph is $v^{\text{source}}_{G_t} = b^{f^t_{\text{start}}}_{\text{start}}$.

Definition 7. A *path* through a control flow graph $G = \left(V, E, v^{\text{source}}_G\right)$ is a sequence of nodes $P = (v_0, v_1, \ldots, v_n)$ such that $(v_{i-1}, v_i) \in E$ for all $i \in \{1, \ldots, n\}$.

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 [12].

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 [12]. 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 which is out of the scope of this report.

4 ANALYSIS FRAMEWORK

We embed our new analyses into the CHRONOS timing analyzer framework from [3]. Figure 4.1 shows the analysis process. The 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 [3], 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 4.1) has been equipped with our new analysis technique, whereas all other modules have not been modified. After the tasks' BCETs and WCETs were computed, the overall system *worst-case response time* (WCRT) is determined. This process repeats as long as the task interference changes, e.g. due to altered task lifetimes. 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 of our analyses are applicable to the computation of BCETs as well.

1 Best-Case Execution Time

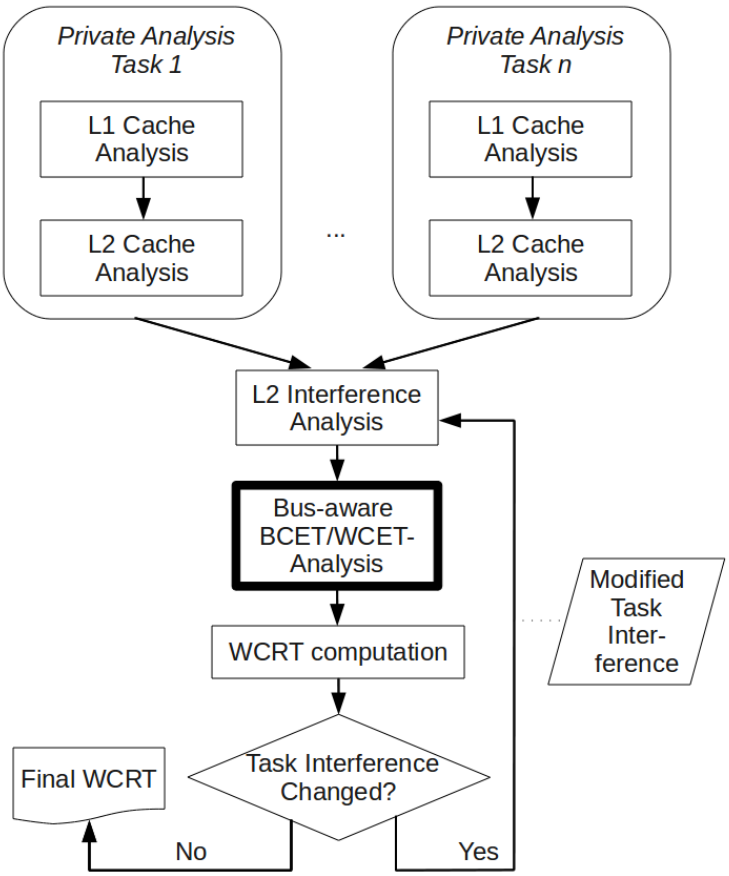


Figure 4.1: The analysis framework

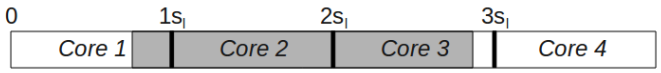
5 STATIC ANALYSIS OF TDMA OFFSETS

Our new analysis builds upon concepts which are heavily used in the analysis of other architectural features. To establish the link between those existing analyses and our new analysis, we first give a short overview of existing static analysis techniques. We also demonstrate why those techniques are not sufficient in our scenario.

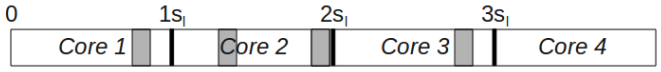
5.1 ABSTRACT INTERPRETATION IN TIMING ANALYSIS

A static timing analysis is usually composed of a microarchitectural analysis and a path analysis [21]. 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 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 3, 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. As a first step towards this, we will introduce a special notion of basic blocks. 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



(a) Offset range



(b) Offset set

Figure 5.1: Different abstract representations for possible start offsets of a basic block

concepts to out-of-order execution is possible, but is omitted for the sake of presentation clarity[1].

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 3.1).

Definition 10. The *absolute time* in the analyzed system is measured in processor cycles[2]. An *absolute point 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* $O = \{o_1, o_2, \ldots, o_n\}$

1 In case of out-of-order pipelines, the offset analysis would need to integrate with the pipeline analysis to obtain all orders in which the instructions can possibly be executed. The offset information must then be computed for every single one of these n orders like in Algorithm 2. The offset results from these n alternative execution scenarios must then be merged with the merge function m presented in this section.

2 We assume a processor with constant clock frequency.

The sets 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 5.1. While Figure 5.1(b) shows the offset set representation with the represented offsets marked in gray, Figure 5.1(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 representations.

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 data-flow analysis [1, 4]. The data-flow 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

$$u_b(O) = \begin{cases} \text{off}_{\text{execute}} & b \text{ never accesses bus} \\ \text{off}_{\text{execute}} \cup \text{off}_{\text{access}} & b \text{ may access bus} \\ \text{off}_{\text{access}} & b \text{ always accesses bus} \end{cases} \tag{1}$$

with

$$\text{off}_{\text{execute}} = \{(o+e) \mod s_l n_c | o \in O, e \in ET_b\} \tag{2}$$

$$\text{off}_{\text{access}} = \{(o+\Phi_p(o,e)) \mod s_l n_c | o \in O, e \in ET_b\} \tag{3}$$

The $\Phi_p(o,e)$ function returns the time needed to finish the bus access (including the bus delay), 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:

$$\Phi_p(o, e) = e + \begin{cases} s_l p - o & \text{if } o < s_l p \\ 0 & \text{if } s_l p \leq o \leq s_l(p+1) - e \\ s_l n_c - o + s_l p & \text{else} \end{cases} \tag{4}$$

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

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 1 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, after definition 2. Φ_p computes the offset of the first cycle t after the execution of the basic block for any starting offset $o \in O$ and runtime $e \in ET_b$. We show this by examining the three cases from Equation 4:

- 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 3.1).

By taking the union over all possible starting offsets $o \in O$ and execution times $e \in ET_b$ in Equation 3 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:

$$m(O_1, O_2) = O_1 \cup O_2 \tag{5}$$

We generalize m to take n arguments instead of just 2 by defining

$$\forall n > 2 : m\left(O_1, \ldots, O_n\right) = m\left(O_1, \ldots, O_{n-1}\right) \cup O_n \tag{6}$$

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

$$n = 1 : u_{(b_1)}\left(O\right) = u_{b_1}\left(O\right) \tag{7}$$

$$\forall n > 1 : u_{(b_1, b_2, \ldots, b_n)}\left(O\right) = u_{b_n}\left(u_{(b_1, b_2, \ldots, b_{n-1})}\left(O\right)\right) \tag{8}$$

Definition 12. A *b-trace* $q_b = (b_1, b_2, \ldots, b_n)$ for a basic block b in the interprocedural control flow graph of a task t is a sequence of basic blocks which starts at the entry block $b_0 = b_{\text{start}}^{f_{\text{start}}^t}$ of the entry function of the task and ends at block $b_n = b$ with $\forall i \in \{2, \ldots, n\} : (b_{i-1}, b_i) \in E_t$. 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

$$O_b^{\text{MOP}} = m\left(\left\{u_{q_b}\left(S\right) | q_b \in Q_b\right\}\right) \tag{9}$$

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 $u_{q_b}\left(S\right)$ 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 establish a standard data-flow analysis on the interprocedural control flow graph of each task. Since our transfer function u is monotonic and S^+ is a power set, this data flow analysis will terminate and the result will be equal to the MOP solution. This follows from the Kleene Fixpoint Theorem and the Coincidence Theorem [4]. We will not go into more detail here, since this is a purely technical application of classical data-flow theory and is of no importance for our own analyses.

Unfortunately, this data-flow 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.

5.2 ABSTRACT HARDWARE STATES AND CONTEXTS

Usually, the hardware states presented in Section 5.1 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 9, 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 [2]. 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, which each 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 from [3], 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.

In the next section, we will present a third, novel approach to context handling in bus access duration analysis, which will analyze $1 \leq x \leq B_L^{max}$ contexts to provide a compromise between analysis duration and analysis precision. Our approach is based on an analysis of TDMA offsets as presented above.

6 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 5. 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 single nodes in the surrounding DAG. Due to this, we required our input tasks to be reducible in Section 3.
- 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 6.1. Section 6.2 then introduces the core analysis which combines the single-iteration results into a complete loop WCET.

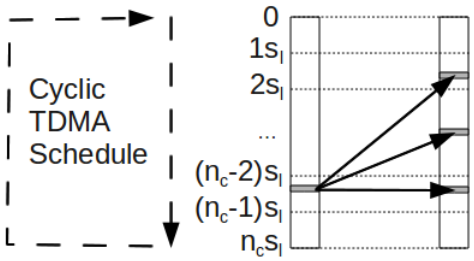


Figure 6.1: Mapping of a start offset to possible end offsets for a single loop iteration. The iteration takes $3s_l$ to $n_c s_l$ cycles in this example, depending on which path through the loop body is taken

6.1 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. Figure 6.1 shows a scenario where a loop iteration, starting from a single, given offset may end at various different offsets, e.g. due to different paths through the loop. These single-iteration offset results can be determined by iterating over the loop's basic block DAG in topological order, as sketched in the following.

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: $wcet = \max\left(wcet, \Phi_p\left(o, e\right)\right)$
8: **end for**
9: **else**
10: $wcet = \max\left(ET_b\right)$
11: **end if**
12: $result = \langle wcet, u_b\left(O_{in}\right)\rangle$
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**

In our analysis of a single loop iteration, each basic block is seen as a transformation function which maps input offsets O_{in} (either an offset set or an offset interval as explained in Section 5.1) 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 5.1, 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 5.2. 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 "AnalyzeLoopIteration". As stated, recursive calls must be converted to standard loops before our analysis can handle them.

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** $\langle wFinish[b_{sink}], oFinish[b_{sink}] \rangle$

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 here and add the missing parts later. We

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** $\langle wFinish[b_{sink}], oFinish[b_{sink}] \rangle$

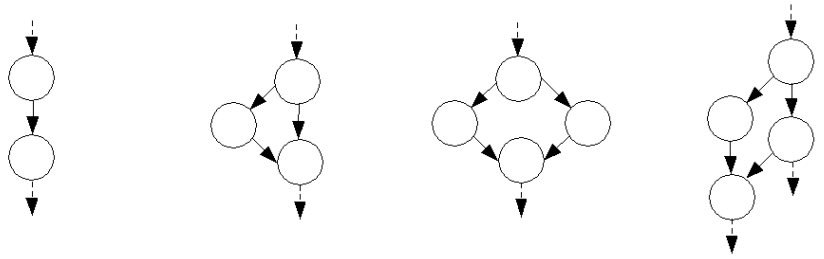
prove the proposition by structural induction over the interprocedural control flow graph.

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. After 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 Φ_p.
- 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.

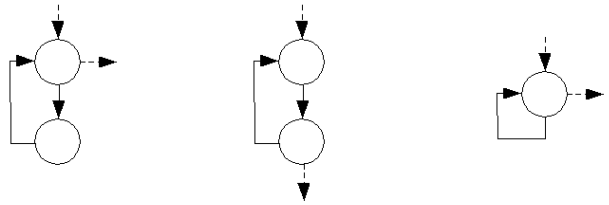
The correctness of the offset result follows from Lemma 1, since the result is computed through a single application of the transfer function u.

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 3. A reducible control flow graph can be inductively defined with the patterns shown in Figures 6.2 and 6.3. Every graph which adheres to Definition 9, which includes



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

Figure 6.2: Sequential structural patterns



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

Figure 6.3: Cyclic structural patterns

our control flow graphs, can be constructed using those inductive patterns [12]. 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
  By the induction hypothesis, the WCET and offset results for the subgraphs are valid overapproximations. For the sequential case shown in Figure 6.2a 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 6.2b and 6.2c 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 6.2d 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 6.3. 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.

6.2 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 [3]. Section 5.2 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 devise two new methods 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 6.3b). Self loops (see Figure 6.3c) are a special case of a natural loop, which consists of a single basic block. For while loops (see Figure 6.3a), 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.

6.2.1 *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 5.1 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 can't 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:

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

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

Observation 1. For two offset sets O_1 ad O_2 with $O_1 \subseteq O_2$ we observe that $wcet_l^{LB}\left(O_1\right) \leq wcet_l^{LB}\left(O_2\right)$ and $u_l^{LB}\left(O_1\right) \subseteq u_l^{LB}\left(O_2\right)$. 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. 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 outer induction hypothesis. 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. With Observation 1 it then follows 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 $k \geq j$. Therefore the offset and WCET results for the j-th iteration are safe overapproximations for all $B_l^{max} - j$ remaining iterations.

□

6.2.2 *Graph Tracking Analysis*

The global convergence analysis is superior to the static unrolling insofar, that it implicitly unrolls the loops selectively, as long as new information can be obtained. This is more suitable than a static unrolling, but it 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.

Example 3. 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 $O_{in}^j = \{x, y\}$ 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 $V = \{v^+, v^-\} \cup V_{off}$ with *source* v^+, *sink* v^-, *context nodes* $V_{off} = \{v_0, ..., v_{s_l n_c}\}$,

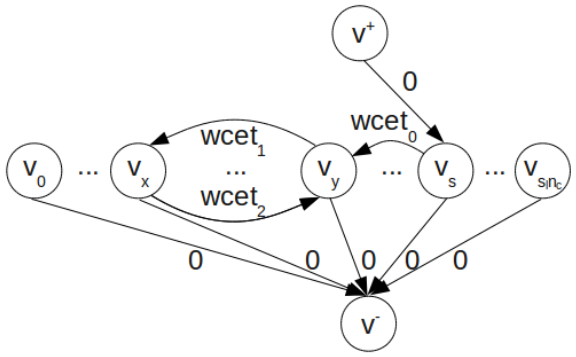


Figure 6.4: An example offset graph

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

$$E_{enter} = \left\{(v^+, v_x) \mid x \in O_{in,l}\right\} \tag{2}$$

$$E_{exit} = \left\{(v_x, v^-) \mid x \in [0, s_l n_c]\right\} \tag{3}$$

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 $wcet_i = wcet_l^{LB}\left(O_{in}^i\right)$ and $O_{out}^i = u_l^{LB}\left(O_{in}^i\right)$. O_{in}^1 is set to $O_{in,l}$ and for the other iterations $O_{in}^i = O_{out}^{i-1}$ applies. After the analysis of each iteration we extend $E_{transition}$ by all edges $e = (v_i, v_o)$ with $i \in O_{in}^i, o \in O_{out}^i$ 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 4. An example for such a graph is given in Figure 6.4 where the first iteration starts with offset s and the succeeding iterations alternate between starting offset x and y as sketched in Example 3.

The offset graph can then be used to obtain the final loop WCET by solving a *dynamic flow problem* [19]. 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 $T = \left\{0, ..., B_l^{max}\right\}$. 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):

$$\forall t \in T : \forall v \in V_{off} : \sum_{e \in \delta^-(v)} x(e, t - \tau(e)) = \sum_{e \in \delta^+(v)} x(e, t) \tag{4}$$

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} (i.e. we can model F full loop executions in a single flow problem). Therefore we have:

$$\sum_{e \in \delta^+(v^+)} x(e, 0) = F \tag{5}$$

$$\forall_{e \in \delta^+(v^+)} : \forall t \in T \setminus \{0\} : x(e, t) = 0 \tag{6}$$

$$\sum_{e \in \delta^-(v^-)} x(e, B_l^{max}) = F \tag{7}$$

$$\forall_{e \in \delta^-(v^-)} : \forall t \in T \setminus \{B_l^{max}\} : x(e, t) = 0 \tag{8}$$

For the WCET analysis we only model the single worst-case loop execution scenario by setting $F = 1$ and by maximizing the objective function

$$\max \sum_{e \in E} \sum_{t \in T} c(e)\, x(e, t) \tag{9}$$

The loop WCET is then given by the value of the objective function.

For the offset analysis, we use $F = s_l n_c$ flow units which must arrive at the sink between time step B_L^{min} and B_L^{max}. We therefore need different sink flow constraints which replace Equations 7 and 8:

$$\sum_{e \in \delta^-(v^-)} \sum_{t \in T_{leave}} x(e, t) = F \tag{10}$$

$$\forall_{e \in \delta^-(v^-)} : \forall t \in T \setminus T_{leave} : x(e, t) = 0 \tag{11}$$

$$\text{with } T_{leave} = \left\{ t \mid B_l^{min} \leq t \leq B_l^{max} \right\} \tag{12}$$

The flow of each of the flow units through the graph models a possible loop execution scenario. 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

$$\max |\left\{z \mid \exists t \in T_{leave} : x\left(\left(v_z, v^-\right), t\right) > 0\right\}| \tag{13}$$

The offsets $O_{out,l}$ which result after the loop execution are then given as the elements of the set from Equation 13 with $K \subseteq O_{out,l}$.

In the following we will prove that the results of the graph problem are valid overapproximation 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 $x : E \times T \rightarrow \mathbb{N}$, subject to the constraints from Equations 4-8 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 2. 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 $reachable(i) \supseteq O_i^{\text{real}}$ 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: By 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 perform 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 $\left(v_{a_p}, v_{a_q}\right) \notin E$. By 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 $\left(v_{a_p}, v_{a_q}\right) \in E$ which is a contradiction. □

Theorem 4. Let us assume $O_{in,l}$ is the set of offsets reaching at the entry of loop l. Given that $O_{in,l}$ is an overapproximation on the set of offsets at the entry of the loop and the offset graph is correctly constructed, 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: In this case, the objective function (Equation 9) 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) with TDMA offset in $src(e)$. Therefore, $\max_{src(e) \in O_{in,l}} c(e)$ precisely represents the WCET of the first loop iteration.

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

$$\max \sum_{e \in E} \sum_{t \in T} c(e) x(e, t) \tag{14}$$

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

$$\max \sum_{e \in E} \sum_{t \in T} c(e)x(e,t) = \max \sum_{e \in E} \sum_{t \in T'} c(e)x(e,t) + \max \sum_{e \in E} c(e)x(e,n+1) \tag{15}$$

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

$$\max \sum_{e \in E} \sum_{t \in T'} c(e)x(e,t) \geq WCET(l,n) \tag{16}$$

From Lemma 2 we know that $reachable(n+1) \supseteq O^{\text{real}}_{n+1}$. If an offset node is not reachable in iteration $n+1$, then he cannot contribute to Equation 15, therefore

$$\max \sum_{e \in E} c(e)x(e,n+1) = \max \sum_{e \in \{(v,w) \in E | v \in reachable(n+1)\}} c(e)x(e,n+1) \tag{17}$$

$$\geq \max \sum_{e \in \left\{(v,w) \in E | v \in O^{\text{real}}\right\}} c(e)x(e,n+1) \tag{18}$$

$$= WCET_{iter}(l,n+1) \tag{19}$$

Inserting Equation 19 and 16 into Equation 15 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 13 is the cardinality of the set of offsets with which the $s_l * n_c$ flow units exit the loop. By Lemma 2 the reachable offsets in the flow graph are an overapproximation of the reachable offsets in the real loop execution for all iterations $j \in \{1, ..., B^{max}_l\}$. Therefore, if the loop can be left in iteration $k \in \{B^{min}_l, ..., B^{max}_l\}$ with offset o_{left} during a real loop execution, then there is the possibility 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.

□

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. Theorem 4 and Theorem 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.

6.3 OFFSET ANALYSIS IN ARCHITECTURES WITHOUT TIMING ANOMALIES

Timing anomalies are a phenomenon which complicates WCET analysis. According to the definition from [17] a system shows timing anomalies whenever local worst-case behavior does not forcedly 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 [18]. In the static analysis of systems with timing anomalies it is not feasible to prune the search space of the analysis [17]. 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 we can safely assume the local worst-case (AM) to increase the analysis performance and precision.

In our offset analysis we did not prune the search space (the set of reachable offsets) at any point up to now. To increase the analysis precision for timing-anomaly-free architectures we can thus reduce the offset result of any merge or update operation to the offset o which is reached by the local worst-case path. Therefore, the differentiation between offset sets and offset intervals is of no importance for the analysis any longer, because we are only tracking single offsets after this reduction. The graph-based analysis is then ideally suited to track the development of the worst-case offsets inside of loops using the known ILPs from Section 6.2 to compute the total loop WCET. This reduction to the local worst-case makes the analysis highly precise, because the main source of imprecision, the divergence of offset information, is eliminated.

6.4 EXTENSIONS FOR FURTHER MICRO-ARCHITECTURAL ANALYSES

In an analysis that includes the analysis of more microarchitectural features like pipeline and branch prediction, 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

2. The hardware states on all nodes have converged

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

7 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 [3] 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 yet, so all instruction latencies are set to 1 cycle. Section 6.4 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 5.2. The number of contexts in turn has the biggest influence on the analysis precision and duration. All experiments were run on an Intel Xeon 2.13GHz machine with 4GB 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.

The experiments were performed on a subset of the `MRTC` test bench [10] where the tasks are independent from one another. Thus we map each `MRTC` test case $i \in [0, 23]$ from Table 7.1 to core $(i \mod n_c)$ with priority i, where 0 is the highest priority. We also tested the presented algorithms with the publicly available `PapaBench` [13] and `Debie` [5] benchmarks which are an unmanned aerial vehicle control software and a space debris monitoring software, respectively. The mapping of tasks to cores was done manually for these two benchmarks. The default system configuration is a 2-core system with 1KB L1 cache (direct-mapped, block size 32 byte) and 2KB L2 cache (4-way associative, block size 64 byte). Only for `Debie`, the cache configuration was changed to 2KB L1 cache (2-way associative) and 8KB L2 cache to account for the bigger program sizes of `Debie`. In any case, the L1 hit 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. The default TDMA schedule assigns a slot of 80 cycles to each core. A more detailed overview of the used benchmarks is provided in Table 7.1, including the byte size s_{byte} of the "text" section of the executable (excluding startup code), the lines of code LOC (excluding comments and empty lines), the number of loops L, the maximum loop nesting level D and the average loop bound $\varnothing_B$. The `Debie`

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 7.1: Benchmark properties

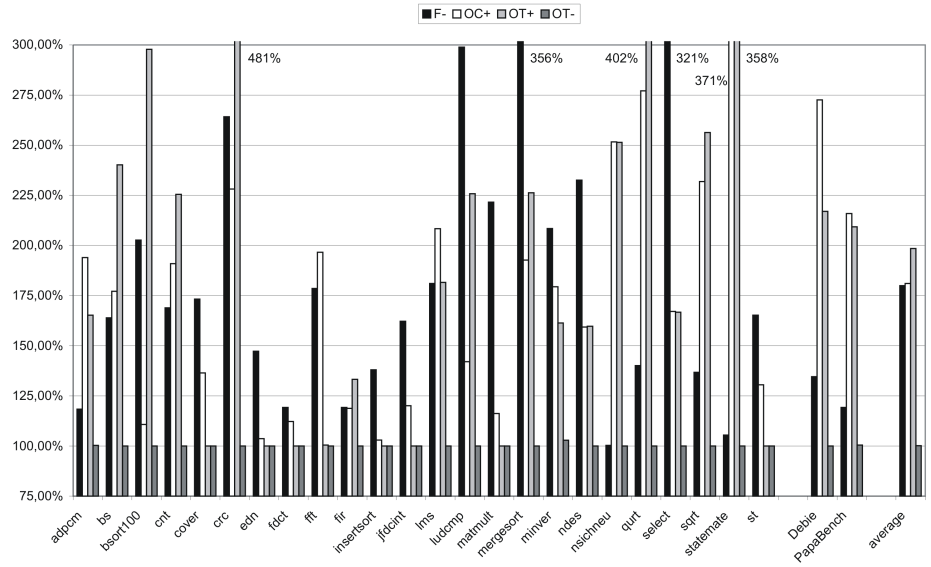


Figure 7.1: WCET results per benchmark and average WCET results for $\langle n_c, s_l \rangle = \langle 2, 80 \rangle$

and `PapaBench` benchmarks consist of 35 resp. 32 individual tasks which have a relatively simple structure, especially since they have no nested loops.

7.1 PRECISION GAIN

In this section, we will distinguish between the approaches that assume no timing anomalies on the target hardware and those which do not make such an assumption. The fully unrolling and fixed-alignment analyses that are built into CHRONOS do make this assumption. Therefore, strictly speaking, only the comparison to our approaches with the extension from Section 6.3 is feasible. In Figure 7.1, we have listed the WCET results for the different approaches on the `MRTC` test bench subset with the mentioned default machine configuration. In Figures 7.1 and 7.2 as well as in the following, all WCET results are relative to the WCET result of the fully unrolling analysis which does *not* consider timing anomalies. We use the following shorthands for the different approaches:

W Assume worst-case bus delay of D^{max} cycles for each bus access

F- Fixed alignment (No timing anomalies) [3]

OC+ Offset analysis (Global convergence, allow timing anomalies)

OT+ Offset analysis (Graph-tracking, allow timing anomalies)

OT- Offset analysis (Graph-tracking, no timing anomalies - with extensions from Sec. 6.3)

U- Full virtual unrolling (No timing anomalies)

The results for **OC-** are not displayed here, because the graph-tracking is the most suitable method for the case without timing anomalies. As can be seen in Figure 7.1 **OT-** almost always (except for `minver`) reaches the same precision as **U-** (100% = **U-**). It also outperforms **F-** which does not analyze cyclic contexts (compare Section 6.2), but instead analyzes all the loop iterations with a fixed alignment and finally adds a penalty term to the result which accounts for the ignored actual alignment of the loop iterations. This leads to imprecision because the actual blocking time due to bus accesses may be much lower than the blocking time for the fixed-alignment situation plus the penalty.

In contrast to **OT-**, our general analyses **OC+** and **OT+** are less precise, which was expected, but still they outperform **F-** on benchmarks which show deeply nested loops or loops with high loop bounds, like for example `mergesort`, `edn`, `ludcmp` or `select`. On benchmarks which have a flat structure with many branches, like `statemate`, **OC+** and **OT+** are outperformed by **F-**, because they lose track of the offsets and must revert to worst-case assumptions. Nevertheless, even in those cases, they are still much more precise than the pessimistic assumption (**W**) that all bus accesses incur maximum delay, which results in an average WCET ratio of 414%. A surprising result is, that **OT+** is worse than **OC+** on average for the `MRTC` test bench subset. This is possible, because the global convergence analysis implicitly unrolls the first iterations as discussed in Section 6.2, whereas the graph-tracking analysis summarizes the iteration behavior in the offset graph. Therefore, once the offset information gets highly imprecise, the graph will be imprecise for all iterations, whereas the global convergence may achieve a better precision during its implicit unrolling. For loops with few iterations, this can have a strong impact on the precision of the WCET estimations. The graph tracking only shows its strength on the rather sparse graphs of **OT-**.

On `Debie` and `PapaBench` **F-** performs much better than on the `MRTC` test bench, because there are no nested loops at all and the loop bounds are rather small. Nevertheless, **F-** is still outperformed by **OT-**, and

	F-	OC+	OT+	OT-	U-
`MRTC`	0.4*s*	15.5*s*	927.3*s*	52.4*s*	1770.6*s*
`Debie`	0.6*s*	5.8*s*	400.6*s*	198.5*s*	1458.7*s*
`PapaBench`	0.05*s*	1.7*s*	1.7*s*	0.4*s*	0.05*s*
Sum	1.05*s*	23.0*s*	1329.6*s*	251.3*s*	3229.4*s*

Table 7.2: Analysis time comparison

also **OT+** performs consistently better than **OC+** which emphasizes its applicability for realworld programs.

All presented results of the offset analyses use the offset interval representation, from Section 5.1. Using the offset set representation the WCET estimation is further reduced by a maximum of 89% for `bsort100` (avg. 1.3%) when combined with graph-tracking, or by a maximum of 0.3% for `bs` (avg. 0.0%) when combined with the global convergence. This underlines the suitability of the combination of offset sets with the graph tracking analysis.

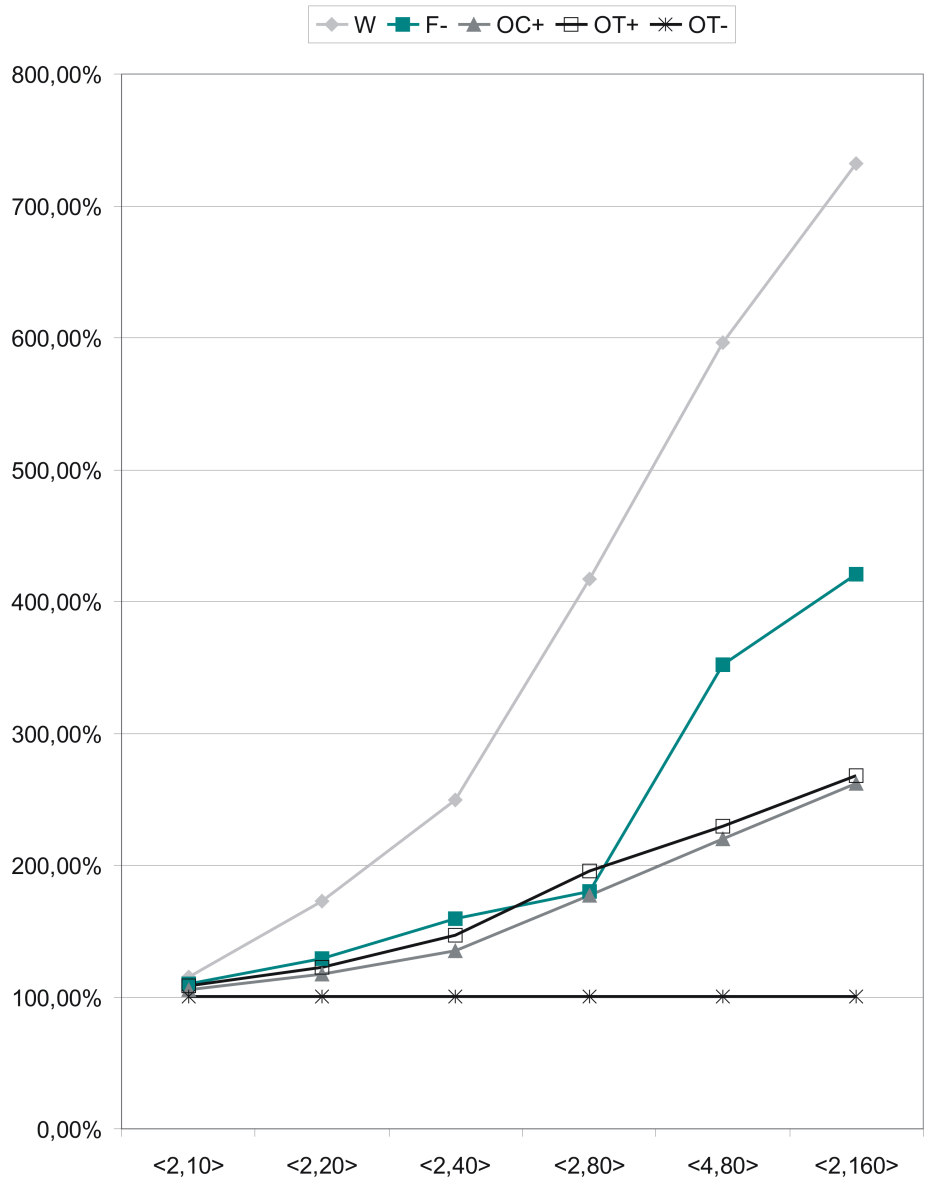
To evaluate the impact of different TDMA slot sizes or processor configurations on the precision of the WCET estimations, the analyses were performed for a varied number of cores (with manually adapted task mapping) and varied TDMA slot lengths. The average WCET results of these experiments are shown in Figure 7.2 where each configuration is described as a tuple $\langle n_c, s_l \rangle$. The experiment shows that **OT-** is able to compute results which are almost equal to those of **U-**, whereas the other analyses suffer from the increased maximum bus delay, **F-** even more so than **OC+** and **OT+**.

7.2 ANALYSIS TIME

Table 7.2 summarizes the analysis duration in seconds for the WCET analyses that generated Figure 7.1. Here it becomes visible that all analyses are much faster in total than **U-**, which takes 53.8 minutes. **OT-** only requires 7.7% of that time and delivers WCET results which deviate by less than 1% from those of **U-**. Therefore **OT-** is the best choice when high analysis precision with moderate runtimes is required. For applications where an extremely short analysis time is required, **F-** can be better suited. It delivers results with 79% overestimation compared to **U-** in only 0.4% of the analysis time of **OT-**.

The unrolling is quick for benchmarks with few loops and low loop bounds like e.g. `PapaBench`. Since its analysis time is directly dependent on the loop structures and the loop bound values, it performs much

Figure 7.2: Average WCET results for varying number of cores and TDMA slot sizes $\langle n_c, s_l \rangle$



worse for `Debie` and `MRTC` where nested loops and higher loop bounds are found (see Table 7.1). This indicates that the unrolling is unsuitable for bigger realworld applications.

8 CONCLUSIONS

We have presented a new approach to the WCET analysis of TDMA-arbitrated shared resources, and applied it to a multicore system with shared bus. Our new analysis type is based on a static analysis of the TDMA offsets with which basic blocks may be entered and uses the key concept of cyclic contexts to improve the analysis precision. Concerning precision and analysis time, our solutions provides a good compromise between the fastest and the most precise approaches. The best variant (**OT-**) reduces the WCET overestimation by 79% compared to the quickest preexisting approach (**F-**) and achieves a speedup of 12.9 compared to the most precise preexisting approach (**U-**). Possible improvements to our methods are

- The integration of further microarchitectural analyses
- Specialized algorithms for the graph-based approach, possibly based on the polynomially solvable *Maximum Dynamic Flow* problem [19]
- A tailored graph clustering or graph expansion to fine-tune the precision of the graph-tracking analysis
- Heuristics which combine the presented analysis techniques to optimize runtime and precision

BIBLIOGRAPHY

[1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2nd edition, 2006.

[2] Alexandru Andrei, Petru Eles, Zebo Peng, and Jakob Rosen. Predictable Implementation of Real-Time Applications on Multiprocessor Systems-on-Chip. In *Proceedings of the 21st International Conference on VLSI Design*, VLSID '08, pages 103–110, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Sudipta Chattopadhyay, Abhik Roychoudhury, and Tulika Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th International Workshop on Software & Compilers for Embedded Systems*, SCOPES '10, pages 6:1–6:10, New York, NY, USA, 2010. ACM.

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

[5] European Space Agency. DEBIE – First Standard Space Debris Monitoring Instrument. `http://gate.etamax.de/edid/publicaccess/debie1.php`, 2008.

[6] Andreas Gustavsson, Andreas Ermedahl, Björn Lisper, and Paul Pettersson. Towards WCET Analysis of Multicore Architectures Using UPPAAL. In *10th International Workshop on Worst-Case Execution Time Analysis*, WCET '10, pages 101–112. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, July 2010.

[7] Damien Hardy, Thomas Piquet, and Isabelle Puaut. 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*, RTSS '09, pages 68–77, Washington, DC, USA, 2009. IEEE Computer Society.

[8] Damien Hardy and Isabelle Puaut. WCET Analysis of Multi-level Non-inclusive Set-Associative Instruction Caches. In *Proceedings of*

the 2008 Real-Time Systems Symposium, pages 456–466, Washington, DC, USA, 2008. IEEE Computer Society.

[9] Mingsong Lv, Nan Guan, Wang Yi, and Ge Yu. Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In *31st IEEE Real-Time Systems Symposium (RTSS)*, 2010.

[10] Mälardalen WCET Research Group. Mälardalen WCET Benchmark Suite. `http://www.mrtc.mdh.se/projects/wcet`, February 2010.

[11] Jörg Mische, Irakli Guliashvili, Sascha Uhrig, and Theo Ungerer. How to Enhance a Superscalar Processor to Provide Hard Real-Time Capable In-Order SMT. pages 2–14, February 2010.

[12] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann, 1997.

[13] Fadia Nemer, Hugues Cassé, Pascal Sainrat, Jean-Paul Bahsoun, and Marianne De Michiel. PapaBench: a Free Real-Time Benchmark. In Frank Mueller, editor, *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum f"ur Informatik (IBFI), Schloss Dagstuhl, Germany.

[14] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Guillem Bernat, and Mateo Valero. Hardware support for WCET analysis of hard real-time multicore systems. In *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, pages 57–68, New York, NY, USA, 2009. ACM.

[15] Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 741–746, 2010.

[16] Christof Pitter and Martin Schoeberl. A real-time Java chip-multiprocessor. *ACM Transactions on Embedded Computing Systems*, 10:9:1–9:34, August 2010.

[17] Jan Reineke and Rathijit Sen. Sound and Efficient WCET Analysis in the Presence of Timing Anomalies. In Niklas Holsti, editor, *9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Germany.

[18] Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A Definition and Classification of Timing Anomalies. In *6th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, WCET '06, 2006.

[19] Martin Skutella. An Introduction to Network Flows Over Time. *Research Trends in Combinatorial Optimization*.

[20] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 300–303, New York, NY, USA, 2008. ACM.

[21] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem Overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7:36:1–36:53, May 2008.

[22] Wei Zhang and Jun Yan. Accurately Estimating Worst-Case Execution Time for Multi-core Processors with Shared Direct-Mapped Instruction Caches. volume 0, pages 455–463, Los Alamitos, CA, USA, 2009. IEEE Computer Society.