A Framework for the Derivation of WCET Analyses for Multi-Core Processors

Michael Jacobs, Sebastian Hahn, Sebastian Hack
Saarland University, Saarbrücken, Germany
Email: jacobs@cs.uni-saarland.de, sebastian.hahn@cs.uni-saarland.de, hack@cs.uni-saarland.de

Abstract—Multi-core processors share common hardware resources between several processor cores. As a consequence, the performance of one processor core is influenced by the programs executed on the concurrent cores. We refer to this phenomenon as shared-resource interference. An explicit consideration of all such interference effects is in general combinatorially infeasible. This makes a precise worst-case execution time (WCET) analysis for multi-core processors challenging.

In order to reduce the complexity, WCET analyses for multi-core processors coarsely approximate the behavior of the considered applications. However, current approaches are only applicable to rather restricted classes of hardware platforms. We propose a framework for the derivation of WCET analyses for multi-core processors. It relaxes the restricting assumptions that existing approaches are based on.

The derivation starts from a WCET analysis that makes maximally pessimistic assumptions about the shared-resource interference. More precise interference bounds for the concrete system are subsequently lifted to the approximation of the analysis. The lifted bounds are finally incorporated in the analysis in order to model the interference in a more precise way.

I. INTRODUCTION

For a timing-critical application it is important that the time needed to deliver the results of its calculations does not exceed a deadline dictated by the physical environment. A timing-critical application may consist of several programs that interact. Knowledge about the worst-case execution time (WCET) [1] of each such program allows us to verify the timeliness of the overall application. It is safe to replace the WCET of a program by an upper bound on its execution times (a so-called WCET bound) in this verification step. However, the timeliness of an application can only be verified if the WCET bounds are relatively tight. WCET analyses are used for the calculation of WCET bounds.

The execution times of a program depend on the possible execution behaviors at the micro-architectural level of the processor that executes the program. Modern processors are too complex to exhaustively simulate or measure the execution times of all possible behaviors. WCET analyses for those processors need to approximate some of the micro-architectural details in order to reduce the inherent complexity [2], [3]. Approximation often comes at the cost of a less tight WCET bound.

The use of multi-core processors can reduce the weight, the energy consumption and the production costs of computer systems. Hence, they are likely to also be used for timing-critical applications in the long run.

However, multi-core processors consist of several processor cores, which share common resources such as buses or caches. The resource sharing has a significant impact on the overall performance of a system [4] because the cores compete for the shared resources. For example, an access request to a shared bus may be blocked for some cycles before it is granted because a concurrent core is granted access first. This effect is commonly referred to as shared-resource interference.

The WCET analysis of programs executed on multi-core processors needs to take into account all possible interference effects due to resource sharing. An exact consideration of all such effects requires in general an exhaustive enumeration of all possible interleavings of accesses to the shared resources by the different processor cores. Such an enumeration is combinatorially infeasible.

Most of the current approaches to WCET analysis for multi-core processors [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] try to find a level of approximation that avoids this complexity without sacrificing precision too much. Unfortunately, they are restricted to Time-Division-Multiple-Access (TDMA) bus arbitration or not sound in the presence of indirect interference effects, which most modern multi-core platforms exhibit.

Contributions

We propose a framework for the derivation of WCET analyses for multi-core processors. An instance of our framework—derived according to the criteria proposed in this paper—is guaranteed to be a sound WCET analysis. The derivation starts from a baseline WCET analysis that makes maximally pessimistic assumptions about the shared-resource interference. We can infer more precise interference bounds from the specification of the concrete system. Lifting these bounds to the approximation of the baseline analysis avoids overly pessimistic assumptions about the interference.

Our iterative overapproximation analyzes each processor core on its own and still incorporates cumulative information about the concurrent cores in the lifted interference bounds. In this way, it finds a trade-off between the performance of analyzing each core in isolation and the precision of simultaneously considering all processor cores.

Our framework has been successfully used in the development of a novel analysis [17] that avoids the restrictions of the existing approaches.

II. RELATED WORK

Most approaches [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] to WCET analysis or response time analysis for multi-core processors rely on compositionality [18] in the sense that they start with a timing analysis that ignores the shared-resource interference. Subsequently, they add bounds on the direct interference effects to their results. In modern micro-processors, however, the overall impact of the interference can exceed the direct interference effects [19]. Thus, these approaches are not applicable to current hardware platforms.

An approach by Chattopadhyay et al. [16] supports complex processor core pipelines. It is restricted to TDMA bus arbitration. Most multi-core processors on the market, however, implement event-driven bus arbitration protocols.

A recent approach by Kelter and Marwedel [20] supports complex multi-core processors equipped with event-driven bus arbitration. However, it relies on the enumeration of all interleavings of accesses to the shared bus by the different cores. Therefore, we expect it to not scale to realistic application scenarios.
A novel analysis developed by our group [17] overcomes the restrictions and simplifying assumptions of previous approaches. To the best of our knowledge, it is the first approach to the calculation of co-runner-sensitive WCET bounds that scales to multi-core processors with out-of-order execution and event-driven bus arbitration.

This paper presents the concepts we applied during the derivation of our novel analysis. They are embedded in a general and formally sound framework for the derivation of WCET analyses for multi-core processors.

### III. Motivation

We motivate the key principle of our framework by considering an example program executed on a hardware platform. Figure 1 shows all six possible execution behaviors of the program. Each sequence of boxes represents one execution behavior. White boxes stand for time units of non-interfered execution. The boxes colored in light blue represent direct interference effects like cycles blocked at a shared bus or needed to serve a miss in the shared cache. Dark boxes denote the prolonging effects of timing anomalies that are a consequence of earlier interference. A processor core pipeline might, for example, only speculate in a particular situation if it is blocked at the shared bus. If the prediction turns out as false, the execution time is prolonged by more than the blocked cycles. Such indirect interference effects can be observed in modern multi-core processors [19]. Sound WCET analyses for such platforms have to take these indirect effects into account.

The example program has a WCET of eleven time units as its longest execution behavior takes this long.

In this example, we assume that no execution behavior of the example program can exhibit more than five direct interference effects. Such assumptions can for instance be inferred from the specification of a bus arbiter if the actual set of execution behaviors is unknown.

#### A. Classical Compositional Analysis

Existing compositional analyses [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15] first calculate a basic timing bound that does not take into account behaviors exhibiting interference effects. Subsequently, they add an upper bound on the direct interference effects to the result. This principle is depicted in Figure 2. The longest behavior without interference takes five time units. The maximum of five direct interference effects is subsequently added.

This shows that the common way of decomposing analyses is efficient but unsound in the presence of indirect interference effects. In order to be sound, an upper bound on the indirect effects has to be additionally incorporated.

#### B. Key Principle of our Framework

The derivation of a WCET analysis in our framework starts from an overapproximation of all behaviors of the program. This overapproximation is maximally pessimistic with respect to the shared-resource interference. Each access request to a shared bus or cache can be a hit or a miss. Figure 4 shows such an overapproximation for our example program. It contains two infeasible behaviors that cannot be observed when actually executing the program (dashed box).

We exploit the upper bound on the direct interference effects to prune the two infeasible behaviors, which exhibit more than five direct interference effects. The remaining execution behaviors result in an exact WCET bound of eleven time units.

In this way, our framework supports the development of timing analyses that explicitly model the impact of the interference on the processor cores and, thus, precisely bound the indirect interference effects.

Note that the pessimistic overapproximation is only a conceptual starting point. The implementation of the analysis derived from our framework will not materialize this overapproximation. Instead, it can directly leave out many of the infeasible behaviors during analysis.

Our example program only exhibits six execution behaviors. Programs executed on real-world hardware platforms, however, exhibit too many execution behaviors to exhaustively enumerate them. Thus, it is common to approximate some of the micro-architectural details [2]. In the next section, we formally show how the principle presented above can be applied to such approximations.

### IV. Property Lifting

This section describes the concept of property lifting, which is the central part of our framework.
A. Concrete Execution Behavior and Time

We consider a multi-core processor consisting of the set $Cores$ of $n$ processor cores.

$$Cores = \{C_1, \ldots, C_n\}$$

For simplicity, we assume that each core only runs one program and that each program may at most be executed once per system run. This restriction is not inherent to our framework but only made to simplify the notation. For a more detailed discussion on how to overcome this restriction, we refer to [22]. In the following, we use the term system to refer to the combination of the hardware including the multi-core processor and the software executed on it.

The system may exhibit different execution behaviors depending on its initial state, external input parameters and clock drift effects. Let $Traces$ be the set of all execution behaviors of the system. Its superset $Universe$ additionally contains the spurious behaviors that might be described by imprecise analyses. Spurious behaviors can, for example, be sequences of concrete system states that cannot be observed during any execution of the concrete system.

$$Universe \supseteq Traces$$

The program executed on processor core $C_i$ can be assigned an execution time per concrete behavior. This time is given by the function $etc_i$,

$$etc_i : Universe \rightarrow \mathbb{N} \cup \{\infty\}$$

The WCET of the program executed on core $C_i$ is its maximal execution time over all execution behaviors of the considered system.

$$WCET_{C_i} = \max_{t \in Traces} etc_i(t)$$

B. Approximation by Abstract Traces

Modern processors usually exhibit too many execution behaviors to allow for an exhaustive consideration of all of them. The set $Traces$ is simply too large. Therefore, it is mandatory to introduce some kind of approximation. The goal is to not have to argue separately about each concrete execution behavior.

In our view, an abstract model of the considered system is given by the tuple $(Traces, \gamma_{\text{trace}})$. $Traces$ is the set of abstract traces of the model. Depending on the chosen way of approximation, an abstract trace might for example be a sequence of abstract states in an analysis based on abstract interpretation [23] or the combination of a sequence of superblocks [5] and a corresponding sequence of blocking cycle counts. The function $\gamma_{\text{trace}}$ maps those abstract traces to subsets of the universe of execution behaviors. Note that $P(Universe)$ denotes the power set of this universe of execution behaviors.

$$\gamma_{\text{trace}} : Traces \rightarrow P(Universe)$$

We say that an abstract model $(\widetilde{Traces}, \gamma_{\text{trace}})$ is an overapproximation of $Traces$ iff:

$$\bigcup_{t \in \text{Traces}} \gamma_{\text{trace}}(t) \supseteq Traces$$

We assume that for each core $C_i$ there is an upper bound on its execution times per abstract trace. This bound shall be given by $UB_{etc_i}$.

$$UB_{etc_i} : \widetilde{Traces} \rightarrow \mathbb{N} \cup \{\infty\}$$

$$\forall t \in \widetilde{Traces} : UB_{etc_i}(t) \geq \max_{t \in \gamma_{\text{trace}}(t)} etc_i(t)$$

From (2) and (3) it follows that the abstract model provides an upper bound to the WCET as defined in (1) by:

$$\max_{t \in Traces} UB_{etc_i}(t) \geq WCET_{C_i}$$

From now on we only consider abstract models that are overapproximations of $Traces$.

C. Infeasible Abstract Traces

The method used to obtain the set of abstract traces (e.g. a static analysis exploring abstract states) might introduce imprecision. Therefore, there may be abstract traces that do not describe any execution behavior of the considered system. We call them infeasible abstract traces.

$$Infeas = \{t \in Traces \mid \gamma_{\text{trace}}(t) \cap Traces = \emptyset\}$$

Correspondingly, we refer to $Traces \setminus Infeas$ as the set of feasible abstract traces. In fact, it follows from (5) that the set of feasible abstract traces is an overapproximation of $Traces$.

$$\bigcup_{t \in Traces \setminus Infeas} \gamma_{\text{trace}}(t) \supseteq Traces$$

Based on an abstract model $(\widetilde{Traces}, \gamma_{\text{trace}})$, which is an overapproximation of $Traces$, we define a set $Deriv(\widetilde{Traces}, \gamma_{\text{trace}})$ of further abstract models as follows:

$$Deriv(\widetilde{Traces}, \gamma_{\text{trace}}) = \{(\widetilde{Traces} \prime, \gamma_{\text{trace}} \prime) \mid \widetilde{Traces} \supseteq \widetilde{Traces} \prime \supseteq \widetilde{Traces} \setminus Infeas\}$$

Intuitively, each element of $Deriv(\widetilde{Traces}, \gamma_{\text{trace}})$ is an overapproximation of $Traces$. So we can calculate an upper bound to the WCET based on any member of $Deriv(\widetilde{Traces}, \gamma_{\text{trace}})$:

$$\forall(\widetilde{Traces} \prime, \gamma_{\text{trace}} \prime) \in Deriv(\widetilde{Traces}, \gamma_{\text{trace}}) : \max_{t \in \text{Traces} \prime} UB_{etc_i}(t) \geq WCET_{C_i}$$

As a consequence, we can ignore an arbitrarily chosen set of infeasible abstract traces in an abstract model. A WCET bound based on the remaining abstract traces is still guaranteed to be sound.

The calculation of WCET bounds is based on upper bounds on the execution times per abstract trace (3). If an abstract model makes conservative assumptions about the behavior at the shared resources, some infeasible abstract traces might assume an amount of shared-resource interference that exceeds the maximum possible amount for the concrete system. As upper bounds on the execution times of such infeasible abstract traces are likely to be very pessimistic, ignoring those abstract traces—as in (8)—might improve the tightness of the resulting WCET bound.

However, it depends heavily on the particular abstract model $(\widetilde{Traces}, \gamma_{\text{trace}})$ and the upper bounds on the execution times per abstract trace whether the WCET bound can be tightened by leaving out some infeasible abstract traces.

We introduced the abstract model to not have to materialize the set $Traces$. The definition of infeasible abstract traces, however, is also based on $Traces$. Therefore, we cannot directly use this definition to detect infeasible abstract traces. The following subsection describes how we can use properties of the system under consideration to detect some infeasible abstract traces.
D. System Properties

We assume properties to be boolean predicates on execution behaviors. System properties are properties that hold for each execution behavior of a concrete system. The existence of a bound on the shared-resource interference may for example be a system property. Let \( Prop = \{ P_1, \ldots, P_k \} \)

\[
\forall t \in \text{Traces} : \forall P_k \in Prop : P_k(t)
\]  

(9)

We want to use these system properties to detect some infeasible abstract traces. But so far, they only argue about execution behaviors of the concrete system. Therefore, we need to lift them to abstract traces. This means, we need to find \( \hat{P}_k \) such that the following criterion holds.

\[
\text{Soundness Criterion (C1):} \\
\forall \hat{t} \in \hat{\text{Traces}} : \\
\exists \hat{\gamma} \in \gamma_{\text{muc}}(\hat{t}) : \neg P_k(\hat{t}) \\
\Rightarrow \exists \hat{\gamma} \in \gamma_{\text{muc}}(\hat{t}) \in \text{Infeas} \\
\text{(C1)}
\]

The intuition behind soundness criterion (C1) gets more clear if we have a look at what it means if \( \hat{P}_k \) does not hold for an abstract trace \( \hat{t} \in \hat{\text{Traces}} \):

\[
\neg \hat{P}_k(\hat{t}) \\
\Rightarrow \forall \hat{\gamma} \in \gamma_{\text{muc}}(\hat{t}) : \neg P_k(\hat{t}) \\
\Rightarrow \gamma_{\text{muc}}(\hat{t}) \cap \text{Traces} = \emptyset \\
\Rightarrow \hat{t} \in \text{Infeas}
\]

(10)

So if a lifted system property does not hold for an abstract trace, this means that the abstract trace is infeasible. From now on, the lifted version of any system property shall be identified by the name of the system property with an additional hat on top.

E. Property Lifting Example

The following example will illustrate how we can find a \( \hat{P}_k(\hat{t}) \) satisfying (C1) without using \( \gamma_{\text{muc}}(\hat{t}) \) directly, which is mandatory for an efficient use of an abstract model.

Example: Assume that we have an upper bound on the number of bus accesses performed by a particular processor core \( C_i \) per abstract trace.

\[
\forall \hat{t} \in \hat{\text{Traces}} : \\
\forall \hat{\gamma} \in \gamma_{\text{muc}}(\hat{t}) : \\
\#\text{accesses}_{C_i}(\hat{t}) \geq \#\text{accesses}_{C_i}(\hat{t}) \\
\text{(a)}
\]

We only use \( \gamma_{\text{muc}} \) to argue about the soundness of the bounds. But we assume that each bound is given by a preceding analysis in the same way as the corresponding abstract trace is.

In addition, we assume to have a lower bound on the number of cycles that core \( C_i \) is blocked at a shared bus per abstract trace.

\[
\forall \hat{t} \in \hat{\text{Traces}} : \\
\forall \hat{\gamma} \in \gamma_{\text{muc}}(\hat{t}) : \\
\#\text{blockedCycles}_{C_i}(\hat{t}) \leq \#\text{blockedCycles}_{C_i}(\hat{t}) \\
\text{(b)}
\]

Now assume that the concrete system we consider uses a Round-Robin policy to arbitrate its shared bus. Therefore, all its execution behaviors fulfill the property \( P_n \):

\[
P_n(t) \iff \#\text{blockedCycles}_{C_i}(t) \\
\leq \#\text{accesses}_{C_i}(t) \cdot (n - 1) \cdot \text{maxCyclesPerAccess} \\
\text{(c)}
\]

The intuition behind this system property (implicitly assumed in [24]) is that with Round-Robin arbitration, each concurrent core (there are \( n - 1 \) of them) can at most perform one access to the bus before an access of core \( C_i \) is granted. Together with an upper bound on the number of cycles that a granted bus access can at most take to complete on the concrete system, we arrive at an upper bound on the number of cycles that any access of core \( C_i \) can be blocked at the bus. Knowledge about how many accesses to the bus are performed by core \( C_i \) allows us to bound the overall amount of bus blocking experienced by core \( C_i \) in a particular execution behavior.

We can safely lift \( P_n \) to abstract traces in a way that satisfies soundness criterion (C1) by applying (a) and (b):

\[
\hat{P}_n(\hat{t}) \iff \#\text{blockedCycles}_{C_i}(\hat{t}) \\
\leq \#\text{accesses}_{C_i}(\hat{t}) \cdot (n - 1) \cdot \text{maxCyclesPerAccess}
\]

(11)

According to (10) any abstract trace \( \hat{t} \) with \( \neg \hat{P}_n(\hat{t}) \) can safely be considered as infeasible. [Example end]

F. Removing Infeasible Abstract Traces

We define a compound property \( \hat{P} \) for abstract traces to be the conjunction over the lifted versions of the considered system properties.

\[
\forall \hat{t} \in \hat{\text{Traces}} : \\
\hat{P}(\hat{t}) \iff \forall P_k \in Prop : \hat{P}_k(\hat{t})
\]

(11)

If \( \hat{P} \) does not hold for an abstract trace \( \hat{t} \) then this means that \( \hat{t} \) is infeasible:

\[
\neg \hat{P}(\hat{t}) \\
\Rightarrow \exists \hat{P}_k \in Prop : \neg \hat{P}_k(\hat{t})
\]

(12)

We can use \( \hat{P} \) to define an alternative set \( LessTraces \) of abstract traces based on \( \text{Traces} \):

\[
LessTraces = \{ \hat{t} \mid \hat{t} \in \text{Traces} \land \hat{P}(\hat{t}) \}
\]

(13)

\( LessTraces \) is the subset of abstract traces in \( \text{Traces} \) that cannot be classified as infeasible by any of the \( \hat{P}_k \).

\[
LessTraces \supseteq \text{Traces} \setminus \text{Infeas}
\]

(14)

Consequently, we can derive a sound WCET bound from the abstract model \( LessTraces, \gamma_{\text{muc}} \):

\[
\max_{\hat{t} \in LessTraces} \text{WCET}_{\hat{t}}(\hat{t}) \geq WCET_{C_i}
\]

(15)

\( LessTraces, \gamma_{\text{muc}} \) can improve the precision, as the set \( LessTraces \) potentially prunes some of the infeasible abstract traces still included in \( \text{Traces} \). In that context, \( \text{Traces, } \gamma_{\text{muc}} \) is referred to as baseline abstract model as it is the starting point for further improvements of precision.

This concludes the description of the concept of property lifting. Intuitively, the main idea is to start with a sound approximation as baseline. Lifted versions of system properties are used to detect some infeasible abstract traces of the baseline approximation. Removing them may result in more precise WCET bounds.
V. ITERATIVE OVER_APPROXIMATION

Property lifting—as described in Section IV—requires a baseline abstract model arguing about all processor cores in detail in order to profit from system properties that interrelate the behaviors of all processor cores. Section V-A uses an exemplary system property to illustrate this requirement.

In Section V-B, we derive a compound abstract model from a set of abstract models—each focusing on one processor core. The compound abstract model argues about all cores in detail. Hence, system properties interrelating the behaviors of all cores can effectively be lifted to it.

However, the high number of abstract traces in the compound abstract model will likely become unmanageable. Thus, we project the analysis results from the compound abstract model back to the different component abstract models (Section V-C). Finally, we present an iterative approach to overapproximate these projections without having to materialize the compound abstract model (Section V-D).

A. Relating the Behavior of one Processor Core to that of Other Cores

Consider system properties that relate the behavior of one processor core to that of other cores. Such properties are typical for systems that do not provide performance isolation between their cores [24], [6].

Example: We introduce a property $P_{wc}$ that holds for certain systems that enforce a work conserving bus arbitration policy.

$$P_{wc}(t) \Leftrightarrow \sum_{C_j \in \{\text{Cores}\setminus\{C_i\}\}} \#\text{accessCycles}_{C_j}(t) \leq \#\text{blockedCycles}_{C_i}(t)$$

Essentially, it states that the number of cycles processor core $C_i$ is blocked at the shared bus cannot exceed the number of cycles in which concurrent cores (here $C_j$) are granted access to the shared bus.

Assume that we have an upper bound on the number of bus access cycles performed by a particular processor core $C_j$ per abstract trace.

$$\forall C_j \in \text{Cores} :$$

$$\forall t \in \text{Traces} :$$

$$\forall \gamma_{\text{core}}(t) :$$

$$\text{UB}_{\gamma_{\text{core}}(t)} \#\text{accessCycles}_{C_j}(t) \geq \#\text{accessCycles}_{C_j}(t)$$

Using these upper bounds, we can lift the property $P_{wc}$ to abstract traces.

$$\overline{P_{wc}}(t) \Leftrightarrow \sum_{C_j \in \{\text{Cores}\setminus\{C_i\}\}} \text{UB}_{\gamma_{\text{core}}(t)} \#\text{accessCycles}_{C_j}(t) \leq \#\text{blockedCycles}_{C_i}(t)$$

If an abstract model only focuses on one processor core, it has to assume arbitrary behaviors for the other cores. For now, assume that the abstract model $(\text{Traces}, \gamma_{\text{core}})$ is only focused on core $C_i$. Thus, it cannot exclude arbitrarily high numbers of bus access cycles for all other cores.

$$\forall C_j \in \{\text{Cores}\setminus\{C_i\}\} :$$

$$\forall t \in \overline{\text{Traces}} :$$

$$\text{UB}_{\gamma_{\text{core}}(t)} \#\text{accessCycles}_{C_j}(t) = \infty$$

As a consequence, the lifted property $\overline{P_{wc}}$ holds for all abstract traces of the abstract model. Hence, it does not detect any infeasible abstract traces. [Example end]
assume a compound abstract model as baseline. It shall be composed of one abstract model per processor core.

Models = Cores

Furthermore assume that each abstract model can only provide detailed information about the processor core it is specialized on. In particular, this means:

\[ \forall C_i \in \text{Cores} : \]
\[ \forall t \in \text{Traces} : \]
\[ \forall C_j \in (\text{Cores} \setminus \{C_i\}) : \]
\[ \text{LB} \# \text{blockedCycles}_{\text{C}_j}(\check{t}^{C_j}) = 0 \land \]
\[ \text{UB} \# \text{accessCycles}_{\text{C}_j}(\check{t}^{C_j}) = \infty \]

In combination with (21) and (22) this implies the following equalities:

\[ \forall C_i \in \text{Cores} : \]
\[ \text{LB} \# \text{blockedCycles}_{\text{C}_i}(\check{t}) = \text{LB} \# \text{blockedCycles}_{\text{C}_i}(\check{t}^{C_i}(\check{t})) \land \]
\[ \text{UB} \# \text{accessCycles}_{\text{C}_i}(\check{t}) = \text{UB} \# \text{accessCycles}_{\text{C}_i}(\check{t}^{C_i}(\check{t})) \]

This allows us to rewrite the lifted property \( \tilde{P}_{wc} \) as follows:

\[ \forall t \in \text{Traces} : \]
\[ \check{P}_{wc}(\check{t}) \]
\[ \phi \left[ \text{LB} \# \text{blockedCycles}_{\text{C}_i}(\check{t}) \right] \]
\[ \leq \sum \text{UB} \# \text{accessCycles}_{\text{C}_j}(\check{t}^{C_j}(\check{t})) \]  
\[ \phi \left[ \text{LB} \# \text{blockedCycles}_{\text{C}_i}(\check{t}^{C_i}(\check{t})) \right] \]
\[ \leq \sum \text{UB} \# \text{accessCycles}_{\text{C}_j}(\check{t}^{C_j}(\check{t})) \]

This time, the lifted property \( \tilde{P}_{wc} \) is not guaranteed to hold for all abstract traces. Hence, it can effectively detect infeasible abstract traces. [Example end]

However, the cross product in the definition of \( \text{Traces} \) already gives a hint that \( \text{Traces} \) might become quite large. Thus, the compound consideration of several abstract models is likely too complex in most cases.

\section{C. Projections of the Compound Results}

Taking a closer look at the set \( \text{LessTraces} \) derived from the compound abstract model, it turns out that we are not really interested in the set of all combinations of abstract traces from the different abstract models. It would be sufficient to know for each \( M_a \in \text{Models} \) which members of \( \text{Traces}^{M_a} \) are contained in a compound abstract trace of \( \text{LessTraces} \). Those subsets can be obtained by projecting the members of \( \text{LessTraces} \) to their different components. We define the projections in a general way on arbitrary subsets \( \text{SomeTraces} \) of \( \text{Traces} \).

\[ \forall M_a \in \text{Models} : \]
\[ \pi^{M_a}(\text{SomeTraces}) = \{ \pi^{M_a}(\check{t}) \mid \check{t} \in \text{SomeTraces} \} \]

Obviously, each projection \( \pi^{M_a}(\text{SomeTraces}) \) is a subset of the set of abstract traces of the corresponding abstract model.

\[ \forall \text{SomeTraces} \subseteq \text{Traces} : \]
\[ \forall M_a \in \text{Models} : \]
\[ \pi^{M_a}(\text{SomeTraces}) \subseteq \text{Traces}^{M_a} \]

Please note that \( \text{SomeTraces} \) is a subset of the cross product over its projections.

\[ \forall \text{SomeTraces} \subseteq \text{Traces} : \]
\[ \text{SomeTraces} \subseteq \pi^{M_1}(\text{SomeTraces}) \times \cdots \times \pi^{M_m}(\text{SomeTraces}) \]

Furthermore, it is rather obvious that the projection functions \( \pi^{M_a} \) are monotonous.

\[ \forall M_a \in \text{Models} : \]
\[ \{ \text{SomeTraces} \subseteq \text{OtherTraces} \} \Rightarrow \{ \pi^{M_a}(\text{SomeTraces}) \subseteq \pi^{M_a}(\text{OtherTraces}) \} \]

Each projection \( \pi^{M_a}(\text{LessTraces}) \) is a superset of the feasible abstract traces of the corresponding \( \pi^{M_a}(\text{SomeTraces}) \). Consider (27) for a formal proof of this statement. According to (7), (24) and (27), each abstract model \( (\pi^{M_a}(\text{LessTraces}), \gamma^{M_a}) \) is a member of \( \text{Deriv}(\text{Traces}^{M_a}, \gamma^{M_a}) \).

\[ \forall M_a \in \text{Models} : \]
\[ (\pi^{M_a}(\text{LessTraces}), \gamma^{M_a}) \in \text{Deriv}(\text{Traces}^{M_a}, \gamma^{M_a}) \]

Thus, each abstract model \( (\pi^{M_a}(\text{LessTraces}), \gamma^{M_a}) \) can be used to calculate a WCET bound based on it.

\[ \forall M_a \in \text{Models} : \]
\[ \max_{t \in \pi^{M_a}(\text{LessTraces})} \text{UB etc}_i(t^{M_a}) \geq \text{WCET}_{C_i} \]

But how precise is a WCET bound based on the projection of \( \text{LessTraces} \) compared to one that is directly based on \( \text{Traces} \)? In general, we might lose precision by restricting ourselves to the projections \( \pi^{M_a}(\text{LessTraces}) \).

\[ \text{WCET}_{C_i} \]
\[ \leq \max_{t \in \text{LessTraces}} \text{UB etc}_i(t) \]
\[ = \max_{M_a \in \text{Models} : t \in \text{LessTraces}} \min_{M_a \in \text{Models} : t \in \text{LessTraces}} \text{UB etc}_i(t^{M_a}(t)) \]
\[ \leq \min_{M_a \in \text{Models} : t \in \text{LessTraces}} \max_{M_a \in \text{Models} : t \in \text{LessTraces}} \text{UB etc}_i(t^{M_a}(t)) \]

We can prove the second \( \leq \) relation used in (30) by assuming its opposite and deriving a statement from it that contradicts to the definition of the minimum.

\[ \max_{t \in \text{LessTraces}} \min_{M_a \in \text{Models} : t \in \text{LessTraces}} \text{UB etc}_i(t^{M_a}(t)) \]
\[ > \min_{M_a \in \text{Models} : t \in \text{LessTraces}} \max_{M_a \in \text{Models} : t \in \text{LessTraces}} \text{UB etc}_i(t^{M_a}(t)) \]
\[ \Rightarrow \exists t^* \in \text{LessTraces} : \exists M_a \in \text{Models} : \]
\[ \text{UB etc}_i(t^{M_a}(t^*)) \]

However, we additionally assume each abstract model is focused on one processor core.

\[ \text{Models} = \text{Cores} \]
This in particular means that each abstract model has to make maximally pessimistic assumptions about the execution times of the cores it is not focused on.

∀Ci ∈ \text{Cores}:

∀\hat{C}_i ∈ \text{Traces}^{\hat{C}_i} :

∀C_j ∈ (\text{Cores} \setminus \{C_i\}) : \forall b \# et_{C_j}(\hat{F}_{\hat{C}_i}) = \infty

(33)

Under those additional assumptions, we are guaranteed to not lose any precision by restricting ourselves to WCET bounds based on the projections \( \pi^{M_a}(\text{LessTraces}) \).

\[ WCET_{C_i} \leq \max_{(14) i \in \text{LessTraces}} \max_{(5) M_a \in \text{Models}} \max_{(26) l} \min_{(15) \text{LessTraces}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \]

\( = \min_{(22) l \in \text{Traces}^{\hat{C}_i}, M_a \in \text{Models}} \min_{(32) l \in \text{Cores}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \)

\( = \max_{(33) l \in \text{LessTraces}} \max_{(30) C_j \in \text{Cores}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \)

\( = \max_{(33) l \in \text{LessTraces}} \max_{(30) C_j \in \text{Cores}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \)

\( = \max_{(15) i \in \text{LessTraces}} \max_{(5) M_a \in \text{Models}} \max_{(16) l} \min_{(15) \text{LessTraces}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \)

\( = \max_{(33) l \in \text{LessTraces}} \max_{(30) C_j \in \text{Cores}} \# et_{C_j}(\hat{F}_{\hat{C}_i}(l)) \)

(34)

So we see that, in general, the projections \( \pi^{M_a}(\text{LessTraces}) \) can be used to derive WCET bounds based on them. We do not need to know all combinations of abstract traces contained in \( \text{LessTraces} \). Under the additional assumptions (32) and (33), we do not lose any precision compared to WCET bounds derived from \( \text{LessTraces} \) directly. However, in most cases we will not be able to precisely obtain the projections \( \pi^{M_a}(\text{LessTraces}) \) without first materializing the set \( \text{LessTraces} \). As discussed before, it is expected to be computationally too expensive to derive the set \( \text{LessTraces} \). Therefore, we are interested in overapproximations of these projections.

D. Overapproximating the Projections

This subsection describes an iterative approach that overapproximates the projections \( \pi^{M_a}(\text{LessTraces}) \). It starts with very conservative assumptions about all projections. Intuitively, the overapproximation of a particular projection can be improved by incorporating information from the overapproximations of the other projections.

Clearly, it is possible to obtain an overapproximation of a projection \( \pi^{M_a}(\text{LessTraces}) \) by considering the abstract model \( (\text{Traces}^{M_a}, \gamma^{M_a}) \) in isolation and providing the set \( \text{LessTraces}^{M_a} \). In this case, however, the lifted versions \( P^{M_a}_k \) of properties \( P_k \) do not help us in detecting infeasible abstract traces if the \( P_k \) need to argue about aspects of the system that are not modeled by \( (\text{Traces}^{M_a}, \gamma^{M_a}) \). Therefore, the overapproximation of a projection \( \pi^{M_a}(\text{LessTraces}) \) should be able to incorporate (likely cumulative) information from the overapproximations of the other projections.

We propose an approach that overapproximates each projection \( \pi^{M_a}(\text{LessTraces}) \) by a corresponding approximation variable \( \text{Approx}^{M_a} \). We use \( \text{Approx} \) to refer to the vector of all approximation variables.

\[ \text{Approx} = (\text{Approx}^{M_1}, \ldots, \text{Approx}^{M_m}) \]

(35)

In the beginning, each \( \text{Approx}^{M_a} \) is initialized to the corresponding \( \text{Traces}^{M_a} \).

\[ \text{Approx} \leftarrow (\text{Traces}^{M_1}, \ldots, \text{Traces}^{M_m}) \]

(36)

Then, the approximation variables are updated according to the following recursive equation system.

\[ \forall M_a \in \text{Models} : \forall \text{Approx}^{M_a} \]

\[ \text{Approx}^{M_a} \leftarrow \text{Approx}^{M_a} \]

\[ = \{ l \in \text{Traces}^{M_a} \} \text{ such that } \Pi^{M_a}_a (\text{Approx}^{M_a}) \]

\( = (a \in \text{Approx}^{M_a}) \)

(37)

We refer to \( \Pi^{M_a}_a \) as the update function of \( \text{Approx}^{M_a} \). From (36) and (37) we can immediately follow that each \( \text{Approx}^{M_a} \) is guaranteed to always be a subset of \( \text{Traces}^{M_a} \).

\[ \forall M_a \in \text{Models} : \forall \text{Approx}^{M_a} \]

\[ \text{Approx}^{M_a} \subseteq \text{Traces}^{M_a} \]

(38)

Therefore, the value range of the vector of approximation variables can be restricted as follows.

\[ \text{Approx} \in \mathcal{P}(\text{Traces}^{M_1}) \times \cdots \times \mathcal{P}(\text{Traces}^{M_m}) \]

(39)
The boolean predicate $\overline{P_k}$ used in $F^M_a$ takes an abstract trace from $\overline{Traces}^M_a$ and the current vector of approximation variables as parameters. It is defined as follows.

$$\forall M_a \in Models : \overline{P_k} (\overline{M_a}, \overline{Approx}) \Rightarrow \forall P_k \in \text{Prop} : P_k^M (t^M_a, \overline{Approx})$$

The $\overline{P_k}$ are properties that overapproximate the $P_k$ lifted to the compound abstract model. They shall fulfill the following criterion with respect to the $\overline{P_k}$.

**Soundness Criterion (C2):**

$$\forall \overline{M_a} \in \overline{Traces}^M_a : [\exists (t^M_1, \ldots, t^M_{m-1}) \in \overline{M_a} : (t^M_1, \ldots, t^M_{m-1}) \in \overline{Approx}^M_1 \times \cdots \times \overline{Approx}^M_{m-1} ;
\overline{P_k} (t^M_1, \ldots, t^M_{m-1}, t^M_m, \overline{Approx}) \Rightarrow P_k^M (t^M_a, \overline{Approx})] \quad (\text{C2})$$

Criterion (C2) allows us to show that each approximation variable is guaranteed to be an overapproximation of the corresponding projection after arbitrary sequences of updates of the approximation variables:

$$\forall M_a \in Models : \pi^M_a (\text{LessTraces}) \subseteq \overline{Approx}^M_a \quad (\text{H1})$$

**Proof:** As a consequence of (24), the claim in (H1) trivially holds for the initial values of the approximation variables as specified in (36). For the general case, however, we assume the hypothesis (H1) to hold after a given sequence of approximation variable updates. In an inductive way, we can use this assumption to show that the hypothesis is preserved by an additional simultaneous update of an arbitrarily chosen set of the approximation variables. For the details of this induction step, please refer to (41) and (42). The inequation chain in (41) shows that all sets contained in it are in fact equal. This information is used in (42) to show that the $F^M_a (\overline{Approx})$ are guaranteed to be supersets of the projections $\pi^M_a (\text{LessTraces})$. According to the equation system given by (37), the approximation variables $\overline{Approx}^M_a$ are updated to the values of the functions $F^M_a (\overline{Approx})$. This proves that the simultaneous update of an arbitrarily chosen set of approximation variables is guaranteed to preserve the hypothesis given by (H1).

As a consequence of (42), we can optionally use the alternative definitions of the update functions $F^M_a$ given in (43). Their use is equivalent to the use of the definitions in (37). Depending on the implementation details of an instance of our framework, it could be more straightforward to use one style of definition or the other.

$$\forall M_a \in Models : F^M_a (\overline{Approx}) := \{ t^M_a | \overline{M_a} \in \overline{Approx}^M_a \land P_k^M (t^M_a, \overline{Approx}) \} \quad (43)$$

It follows from hypothesis (H1) that we can bound the content of the sets $\overline{Approx}^M_a$ from above and from below:

$$\overline{Traces}^M_a \supseteq \overline{Approx}^M_a \supseteq \pi^M_a (\text{LessTraces}) \supseteq \overline{Traces}^M_a \setminus \text{Infeas}^M_a \quad (44)$$

As a consequence of (7) and (44), we see that each abstract model $(\overline{Approx}^M_a, \gamma_{\text{trace}})$ is a member of $\text{Deriv} (\overline{Traces}^M_a, \gamma_{\text{trace}})$:

$$\forall M_a \in Models : (\overline{Approx}^M_a, \gamma_{\text{trace}}) \in \text{Deriv} (\overline{Traces}^M_a, \gamma_{\text{trace}}) \quad (45)$$

Thus, each abstract model $(\overline{Approx}^M_a, \gamma_{\text{trace}})$ can be used to calculate a WCET bound based on it.

$$\forall M_a \in Models : \max_{t^M_a \in \overline{Approx}^M_a} \text{UB}_{\text{ETC}_i} (t^M_a) \geq \text{WCET}_{\text{C}_i} \quad (46)$$

In addition, the $\overline{P_k}$ shall fulfill the following criterion.

**Monotonicity Criterion (C3):**

$$\forall \overline{Approx} \in \overline{Traces}^M_a : [\forall M_a \in Models : \overline{Approx}^M_a \subseteq \overline{Approx}^M_a \Rightarrow \overline{P_k} (t^M_a, \overline{Approx}) \Rightarrow P_k^M (t^M_a, \overline{Approx})] \quad (47)$$

Let $\overline{Approx}$ be the vector of approximation variables after an arbitrary sequence of updates of the approximation variables. Criterion (C3) allows us to show that the following additional hypothesis holds:

$$\forall M_a \in Models : F^M_a (\overline{Approx}) \subseteq \overline{Approx}^M_a \quad (H2)$$

**Proof:** According to (36) and (37), the hypothesis (H2) trivially holds for a vector $\overline{Approx}$ just initialized. For the inductive step, assume that hypothesis (H2) holds for a given vector $\overline{Approx}$ of approximation variables. Let $\overline{Approx}$ be the successor of $\overline{Approx}$ after the simultaneous update of an arbitrarily chosen set of approximation variables:

$$\overline{Approx} = (\overline{Approx}^M_1, \ldots, \overline{Approx}^M_m) \quad (47)$$

$$\forall M_a \in Models : \overline{Approx}^M_a \in \{ \overline{Approx}^M_a, F^M_a (\overline{Approx}) \} \quad (48)$$

It follows from (H2) and (48) that each component of $\overline{Approx}$ is a subset of its corresponding counterpart in $\overline{Approx}$.

$$\forall M_a \in Models : \overline{Approx}^M_a \subseteq \overline{Approx}^M_a \quad (49)$$
\[
\pi_{\mathcal{M}}(\text{LessTraces}) \\
\equiv \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{LessTraces} \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces}^{M_1} \times \cdots \times \text{Traces}^{M_m} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces}^{M_1} \times \cdots \times \text{Traces}^{M_m} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{LessTraces} \right\}
\]

(41)

\[
\pi_{\mathcal{M}}(\text{LessTraces}) \\
\equiv \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Approx}^{M_1} \times \cdots \times \text{Approx}^{M_m} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces}^{M_1} \times \cdots \times \text{Traces}^{M_m} \wedge \hat{P}(\hat{t}) \right\} \\
\subseteq \left\{ \pi_{\text{trace}}(\hat{t}) \mid \hat{t} \in \text{Traces} \wedge \hat{P}(\hat{t}) \right\} \\
= \pi_{\mathcal{M}}(\text{Approx})
\]

(42)

We assume that the update functions \( F^{M_a} \) can be applied to \( \text{Approx} \) in the same way as to \( \text{Approx} \). Equation (50) shows that \( F^{M_a}(\text{Approx}) \) is a subset of \( F^{M_a}(\text{Approx}) \), proof of (H2):

\[
\text{Approx}^{M_a} \\
\equiv \left\{ \text{Approx}^{M_a} \right\} \\
\subseteq \left\{ F^{M_a}(\text{Approx}) \right\} \\
\subseteq \left\{ F^{M_a}(\text{Approx}) \right\}
\]

(51)

\begin{align*}
\text{Approx}^{M_a} & \subseteq F^{M_a}(\text{Approx}) \\
\implies & F^{M_a}(\text{Approx})
\end{align*}

(50)

The intuition behind (H2) is that the update of an approximation variable is guaranteed to never increase its set of abstract traces. As the calculation of the WCET bounds is based on the abstract trace sets, we can be sure that the update of some approximation variables can never result in worse WCET bounds.

Example: Coming back to the example of Sections V-A and V-B, we can further lift the property \( F_m \)—as defined in equation (m)—in a way that satisfies criteria (C2) and (C3):

\[
\text{Prop}^{C_i} (t^{C_i}, \text{Approx}^{C_1}, \ldots, \text{Approx}^{C_n}) \\
\equiv \left\{ \text{Prop}^{C_i} (t^{C_i}, \text{Approx}^{C_1}, \ldots, \text{Approx}^{C_n}) \right\}
\]

(48)

\[
\sum_{C_j \in \text{Cores}(C_i)} \max_{t^{C_j} \in \text{Approx}^{C_j}} \text{Prop}^{C_j} (t^{C_j}) \leq \text{Prop}^{C_i} (t^{C_i}, \text{Approx}^{C_1}, \ldots, \text{Approx}^{C_n})
\]

(50)

Based on those results, it is straightforward to show that the hypothesis also holds for \( \text{Approx}^\prime \), which concludes the inductive
Note that the right-hand side of the inequation in property $P_{\text{Approx}}^C$ does not depend on the abstract trace $t_C$. It contains cumulative information about the processor cores competing against $C_i$. Thus, this right-hand side is constant over all evaluations of $P_{\text{Approx}}^C$ during an update of $Approx_C^C$. Therefore, we can precompute the constant right-hand side based on the other approximation variables before updating $Approx_C^C$. The constant right-hand side can subsequently be used in an integer linear programming constraint. [17] 

Moreover, we can use hypothesis (H2) to show that all sets contained in the following cyclic inequation chain are equal.

$$
E_{\text{Approx}}^M = \{ t_M \mid t_M \in E_{\text{Approx}}^M (\text{Approx}) \land P_{\text{Approx}}^M (t_M, \text{Approx}) \} 
$$

$$
\subseteq \{ t_M \mid t_M \in Approx^M \land P_{\text{Approx}}^M (t_M, \text{Approx}) \} 
$$

$$
\subseteq \{ t_M \mid t_M \in Traces^M \land P_{\text{Approx}}^M (t_M, \text{Approx}) \} 
$$

$$
= \{ t_M \mid t_M \in E_{\text{Approx}}^M (\text{Approx}) \} 
$$

An interesting consequence of (52) is that, in particular, the following equation holds.

$$
\{ t_M \mid t_M \in Approx^M \land P_{\text{Approx}}^M (t_M, \text{Approx}) \} = \{ t_M \mid t_M \in Traces^M \land P_{\text{Approx}}^M (t_M, \text{Approx}) \} 
$$

Using (53) in the induction step (42) of the soundness proof instead of (38) allows us to replace the last $\subseteq$ by an equal. With this improvement in place, we see that (42) has only left a single $\subseteq$. This means, we can exactly point out where the update of an approximation variable may lose precision compared to the projections of the compound consideration of all models at once. The compound consideration of all models only keeps those $t_M$ that occur in a combination with abstract traces from the other models that fulfills all lifted properties $P_k$. However, the specially lifted properties $P_{\text{Approx}}^M$ may lead to different results. In their presence, an abstract trace $t_M$ is not pruned as soon as for each $P_k$ there is a particular combination with abstract traces from the other models that fulfills $P_k$—this is a consequence of (C2). The intersection of those sets of combinations of abstract traces for the different $P_k$ could possibly be empty for a particular $t_M$. In that case, this $t_M$ is not contained in $\pi^M (\text{LessTraces})$, but in its overapproximation $Approx^M$.

Please note that the additional abstract trace introduced in this way by overapproximation are inherent to considering each abstract model on its own. They can occur even if we choose the $P_{\text{Approx}}^M$ in a way that the $\Rightarrow$ in criterion (C2) can be shown to be replaceable by a $\Leftrightarrow$. This inherent amount of overapproximation only depends on the abstract models $(Traces^M, \pi^M)$ and the way in which the $P_k$ are chosen. Of course, it might lead to further overapproximation if we choose the $P_{\text{Approx}}^M$ in a way that we cannot prove the additional equivalence relation.

VI. ADVANTAGES OF THE FRAMEWORK

This section highlights the benefits of using our framework.

Standard derivation procedure: The framework is a common starting point for the derivation of future WCET analyses for multi-core processors. It has been successfully used in the development of a novel analysis that avoids the restrictions of previous approaches (cf. Section II).

Soundness guarantee: We show in this paper that an instance of our framework is a sound WCET analysis. This soundness is a consequence of a sound baseline analysis and the property lifting according to the criteria presented above. A sound baseline analysis is easily obtained by adapting a single-core WCET analysis in a way that makes it maximally pessimistic with respect to the shared-resource interference [17]. Hence, our framework essentially reduces the soundness proofs of its instances to showing the soundness of the property lifting steps involved in their derivations.

Assumptions about the system always explicit: The declarative style of our framework makes it mandatory to explicitly list all properties that a derived analysis assumes about the system under analysis. This makes sure that a derived analysis does not rely on implicit assumptions (except those that its baseline analysis already relies on).

Clean separation between concrete system and approximation: Existing analyses often try to directly incorporate properties of the concrete system in their level of approximation. However, this is mostly based on intuition and, thus, very error-prone. The concept of property lifting, in contrast, provides a clean separation between system properties and their implications on the approximation.

Trade-off between efficiency and precision: The iterative over-approximation (Section V-D) forms a trade-off between the efficiency of analyzing the programs of one processor core in isolation (Section V-A) and the precision of performing a simultaneous consideration of the detailed behaviors of the programs executed on all processor cores (Section V-B).

Not limited to multi-core processors: The principles presented throughout this paper are not limited to the analysis of multi-core processors. Some of the techniques used in single-core WCET analysis can also be seen as instances of our framework. The micro-architectural analysis, for example, typically has no notion of loop bounds. Thus, it pessimistically assumes that each loop body in the program can be executed indefinitely. Loop bounds of the concrete program are subsequently lifted to the level of approximation that the path analysis operates on. The lifted loop bounds are typically implemented as additional constraints in an implicit path enumeration [25].

VII. FRAMEWORK INSTANTIATION WORKFLOW

Figure 5 sketches the typical workflow of deriving WCET analyses as instances of our framework. A derivation that only relies on the concept of property lifting (cf. Section IV) comprises two logical steps. The derivation of an analysis that iteratively overapproximates the results of properties lifted to a compound abstract model (cf. Section V) requires an additional lifting step.

We successfully used our framework for the derivation of two novel WCET analyses for multi-core processors with a shared bus and Round-Robin bus arbitration: a co-runner-insensitive analysis and a co-runner-sensitive one [17]. This section describes—at a high level—how the derivation of each of these analyses follows the instantiation workflow sketched in Figure 5.

The derivation of our co-runner-insensitive analysis comprises two steps:

Step 1: The derivation starts from a baseline analysis focusing on one core and assuming that each access request to the shared bus can be blocked indefinitely by the bus arbiter. Furthermore, we consider a system property that bounds the maximum amount of blocked cycles per access to the shared bus under Round-Robin arbitration.

Step 2: The Round-Robin property is lifted to the baseline analysis. The lifted property is subsequently added to the implementation of...
the baseline analysis in order to prune infeasible behavior at its level of approximation.

The derivation of our co-runner-sensitive analysis, in contrast, comprises three steps:

**Step 1:** The derivation starts from a compound baseline abstract model that consists of one co-runner-insensitive analysis per processor core. Thus, the compound baseline analysis argues about all processor cores. We consider a property that bounds the blocked cycles of the core under analysis based on the access cycles of the concurrent cores assuming a work-conserving bus arbitration (like e.g. Round-Robin).

**Step 2:** The work-conserving property is lifted to the baseline abstract model. However, it would be impractical to enumerate all combinations of abstract traces of the analyses for the different cores (since the compound abstract model is defined as cross product over its components).

**Step 3:** Hence, we further lift the already lifted property to the component analysis only focusing on the core under analysis. To this end, we assume per concurrent core the maximum amount of access cycles possible in any interval no longer than the current WCET bound of the core under analysis. The resulting analysis starts by calculating the co-runner-insensitive WCET bound for the core under analysis. Then, it calculates upper bounds on the concurrent access cycles and subsequently recalculates the WCET bound. This process is repeated until a fixed point is reached.

### VIII. Experimental Evaluation

We evaluate our analysis prototype for multi-core processors with ARM® instruction set, a shared memory bus, and Round-Robin bus arbitration. Our experiments consider cores with in-order pipelines (five stages) as well as cores that support out-of-order execution (Tomasulo dynamic scheduling, three functional units, and speculative (five stages) as well as cores that support out-of-order execution (Tomasulo dynamic scheduling, three functional units, and speculative execution). We also consider two scenarios with respect to the local instruction memories of the cores. First, we assume a local instruction scratchpad that is statically initialized with all programs executed on the core. Secondly, we consider a local instruction cache (1KiB size, least-recently-used [LRU] replacement policy) that is connected to the shared bus. Table II lists the four resulting core configurations.

The programs executed on the concurrent cores. As we have shown before [17], a co-runner-sensitive analysis can lead to a significant reduction of the WCET bounds.

In contrast to classical compositional approaches [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], our analysis prototype supports hardware platforms exhibiting indirect interference effects (cf. Section III). We estimate the additional cost of considering indirect interference effects by comparing the analysis runtime to the runtime of an analysis that ignores all interference effects (which is the main part of classical compositional timing analysis). The average increase in analysis runtime is moderate (up to 5.4 percent) for hardware platforms with in-order execution or instruction scratchpads (\(Conf^\text{io}, Conf^\text{ins}, Conf^\text{ic}\)). The combination of out-of-order execution and instruction caches (\(Conf^\text{ic}\)), however, leads to a significantly higher—though still bearable—increase in analysis runtime (up to 15.9 percent on average). Intuitively, the complexity of modeling the pipeline features multiplies with the complexity of modeling the shared-bus interference by non-determinism. Note that these runtime results are a significant improvement compared to the numbers we reported in our earlier work. The improvement stems from engineering improvements (which are not in the scope of this paper) of the implementation of our analysis.

The average runtime increase factors for dual-core, quad-core, and octa-core processors with the same core configuration are essentially identical for all our experiments (the small deviations are caused by the heavy use of hash sets in our prototype implementation). Intuitively, for the considered processor core configurations, the core pipelines already converge for each access to the shared bus in a dual-core processor. As a consequence, further cycles blocked at the shared bus do not result in new pipeline states. An optimization (fast-forwarding of converged chains [17]) in our analysis prototype exploits this convergence. For analyses relying on the enumeration of all interleavings of bus access requests by the different processor cores [20], in contrast, each additional core increases the analysis runtime by a factor. Thus, such analyses do not scale to high numbers of processor cores.

### IX. Future Work

Our current prototype implementation only takes into account shared-bus interference. We plan to also consider shared caches and

---

cache coherence in a future version of our tool. A long-term goal is the modeling of commercially available multi-core processors with our techniques.

Furthermore, we plan to study the impact of complex processor core features like store buffers and speculation on the performance of our analysis approach. In this context, we will investigate performance improvements of our tool in order to further reduce the performance overhead due to the consideration of shared-resource interference. As a result of our studies, we will give recommendations for the design of future multi-core hardware platforms to enable their use in timing-critical embedded system.

X. SUMMARY

We present a framework for the derivation of WCET analyses for multi-core processors. It centers around the concept of property lifting. Instances of the framework are sound WCET analyses. The framework has been successfully used in the development of a novel analysis that avoids the restrictions of existing approaches.

ACKNOWLEDGMENTS

The authors would like to thank Mihail Asavoaie, Florian Haupenthal, Max John, Jan Reineke, and Reinhard Wilhelm for many comments and interesting discussions. This work was supported by the DFG as part of the Transregional Collaborative Research Centre SFB/TR 14 (AVACS) and by the Saarbrücken Graduate School of Computer Science which receives funding from the DFG as part of the Excellence Initiative of the German Federal and State Governments.

REFERENCES