Improving the Precision of Approximations in WCET Analysis for Multi-Core Processors

Michael Jacobs (jacobs@cs.uni-saarland.de)
Saarland University, Saarbrücken, Germany

***Abstract*—The worst-case execution time (WCET) analysis for multi-core processors is a challenge. An explicit consideration of all possible interference effects caused by the shared resources is in many cases combinatorially infeasible. Therefore, approximations are used to reduce this complexity. Current approaches to WCET analysis for multi-core processors are specific to particular ways of approximating the underlying system. Furthermore they are only applicable to rather restricted classes of processors. We identified a common methodology behind existing approaches and formalized it in a unified meta approach. Our meta approach is not restricted to a particular way of approximation. It allows to improve the precision of the obtained WCET bounds by incorporating properties of the system under consideration.**

I. Introduction

Multi-core processors consist of several processor cores, which share common resources such as buses or caches. Their use 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 [1].

However, resource sharing can have a significant impact on the overall performance of a system [2] because several cores compete for the shared resources. This effect is commonly referred to as *shared resource interference*.

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 time-critical application may consist of several programs that interact. Knowledge about the worst-case execution time (WCET) [3] of each such program allows 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 in this verification step. However, the timeliness of an application can often only be verified if these upper bounds are relatively tight. WCET analyses can be used to derive these upper bounds on the execution times of programs. The execution times of a program depend heavily on possible execution behaviors at the microarchitectural level of the processor that executes the program. From now on, we just use *behaviors* to refer to the execution behaviors at the microarchitectural level of a processor.

Very simple processors allow for a precise WCET analysis by instruction counting or time measurement of a single execution run of the system [4]. Modern processors, however, 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 microarchitectural details in order to reduce the inherent complexity [5], [6]. Approximation often comes at the cost of a less tight WCET bound.

The WCET analysis of programs executed on multi-core processors is a special challenge. It needs to take into account all possible interference effects due to resource sharing. A precise consideration of all such effects might in many cases require an exhaustive enumeration of all possible interleavings of accesses to the shared resources by the different processor cores. Such a consideration can be looked upon as combinatorially too complex.

Current approaches to WCET analysis for multi-core processors [7], [8], [9], [10], [11], [12], [13] try to find a level of approximation that avoids this complexity without sacrificing precision too much. Unfortunately, the existing approaches are only applicable to programs executed on very restricted classes of processors. Furthermore, each approach is specific to a particular way of approximating the behavior of the considered system.

In our opinion, a first step toward overcoming these limitations is to identify the common ideas of existing approaches. This will help in designing future approaches in a more generic and uniform way. Our contribution is a meta approach that is not restricted to a particular way of approximation. It allows to improve the precision of the obtained WCET bounds by incorporating properties of the system under consideration.

II. Related Work

The existing approaches to WCET analysis for multi-core processors are derived in formally quite different ways. Yet, the different approaches loosely follow a common two-step methodology.

As a common starting point, all considered approaches assume a level of approximation that does not restrict the amount of shared resource interference. Schranzhofer et al. represent a program executed on a particular core as a sequence of superblocks [7], [8], [9]. A superblock only bounds the number of processor cycles (not blocked at the bus) and the number of bus accesses for a part of the program. Liang et al. assume upper and lower bounds on the number of processor cycles per basic block [10]. These bounds ignore possible penalty cycles induced by the memory hierarchy. The approach of Chattopadhyay et al. [11] bounds the points in time that each program instruction can spend in each pipeline stage of the processor core.

Furthermore, all approaches assume properties of the concrete system under analysis, that provide bounds on the amount of shared resource interference. The approaches considering a shared bus provide bounds on the number of blocked cycles. The intuition behind the bounds stems from the protocol used for bus arbitration (Time Division Multiple Access [7], [11], Round-Robin [12], [13], First-Come-First-Served [12], [8]). The approaches considering a shared cache exploit system properties that guarantee that certain accesses to the shared cache cannot miss [10], [11]. All approaches use these bounds on the shared resource interference to exclude some of the spurious execution behaviors introduced by approximation.

Our meta approach depicts this common methodology in a formal and generic way.

III. Concrete Execution Behavior and Time

We consider a multi-core processor consisting of the set *Cores* of processor cores. For simplicity, we assume that each core only runs one program and that each program may at most be executed once per system run. In the following, we use the term *system* to refer to the combination of the hardware containing 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* contains the execution behaviors of arbitrary systems.

$$\mathit{Universe} \supseteq \mathit{Traces}$$

Each core C (or the program executed on it) can be assigned an execution time per execution behavior. This time is given by the function et_C.

$$et_C : \mathit{Universe} \to \mathbb{N} \cup \{\infty\}$$

The WCET of a core C is the maximal execution time for C over all execution behaviors of the considered system.

$$\mathit{WCET}_C = \max_{t \in \mathit{Traces}} et_C(t) \tag{1}$$

IV. 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 is given by the tuple $(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})$. $\widehat{\mathit{Traces}}$ is the set of abstract traces of the model. The function γ_{trace} maps those abstract traces to subsets of the universe of execution behaviors. Please note that $\mathcal{P}(\mathit{Universe})$ denotes the power set of this universe of execution behaviors.

$$\gamma_{\mathit{trace}} : \widehat{\mathit{Traces}} \to \mathcal{P}(\mathit{Universe})$$

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

$$\bigcup_{\hat{t} \in \widehat{\mathit{Traces}}} \gamma_{\mathit{trace}}(\hat{t}) \supseteq \mathit{Traces} \tag{2}$$

We assume that for each core C there is an upper bound on its execution times per abstract trace. This bound shall be given by ${}^{UB}et_C$.

$$\begin{aligned} {}^{UB}et_C &: \widehat{\mathit{Traces}} \to \mathbb{N} \cup \{\infty\} \\ \forall \hat{t} \in \widehat{\mathit{Traces}} &: {}^{UB}et_C(\hat{t}) \geq \max_{t \in \gamma_{\mathit{trace}}(\hat{t})} et_C(t) \end{aligned} \tag{3}$$

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

$$\max_{\hat{t} \in \widehat{\mathit{Traces}}} {}^{UB}et_C(\hat{t}) \geq \mathit{WCET}_C \tag{4}$$

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

A sound abstract model for a multi-core processor can be derived in a similar way as for a single-core processor by only focussing on one core. This will likely result in very conservative assumptions about the behavior of this core when accessing shared resources as the behavior of shared resources and concurrent cores is not explicitly considered. The baseline approximations of the approaches discussed in Section II follow this paradigm.

V. Infeasible Abstract Traces

The method used to obtain the set of abstract traces (e.g. static analysis) might introduce imprecision. Therefore, there may be abstract traces that only describe spurious execution behavior. We call them *infeasible* abstract traces.

$$\widehat{\mathit{Infeas}} = \{\hat{t} \mid \hat{t} \in \widehat{\mathit{Traces}} \wedge \gamma_{\mathit{trace}}(\hat{t}) \cap \mathit{Traces} = \emptyset\} \tag{5}$$

Correspondingly, we refer to $\widehat{\mathit{Traces}} \setminus \widehat{\mathit{Infeas}}$ 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*.

$$\bigcup_{\hat{t} \in \widehat{\mathit{Traces}} \setminus \widehat{\mathit{Infeas}}} \gamma_{\mathit{trace}}(\hat{t}) \supseteq \mathit{Traces} \tag{6}$$

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

$$\begin{aligned} &\mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})} = \\ &\{(\widehat{\mathit{Traces}'}, \gamma_{\mathit{trace}}) \mid \widehat{\mathit{Traces}} \supseteq \widehat{\mathit{Traces}'} \supseteq \widehat{\mathit{Traces}} \setminus \widehat{\mathit{Infeas}}\} \end{aligned} \tag{7}$$

Consider an element $(\widehat{\mathit{Traces}'}, \gamma_{\mathit{trace}})$ of set $\mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})}$. According to (7), $\widehat{\mathit{Traces}'}$ is a subset of $\widehat{\mathit{Traces}}$ that contains at least all feasible abstract traces of $\widehat{\mathit{Traces}}$. It follows from (6) and (7) that each element of $\mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})}$ is an overapproximation of *Traces*.

$$\begin{aligned} \forall (\widehat{\mathit{Traces}'}, \gamma_{\mathit{trace}}) \in \mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})} &: \\ \bigcup_{\hat{t} \in \widehat{\mathit{Traces}'}} \gamma_{\mathit{trace}}(\hat{t}) &\supseteq \mathit{Traces} \end{aligned} \tag{8}$$

In a similar way as (2) and (3) imply (4), it is a consequence of (8) and (3) that we can calculate an upper bound to the WCET based on any member of $\mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})}$:

$$\begin{aligned} \forall (\widehat{\mathit{Traces}'}, \gamma_{\mathit{trace}}) \in \mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})} &: \\ \max_{\hat{t} \in \widehat{\mathit{Traces}'}} {}^{UB}et_C(\hat{t}) &\geq \mathit{WCET}_C \end{aligned} \tag{9}$$

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 (9)—might improve the tightness of the resulting WCET bound significantly.

However, it depends heavily on the abstract model $(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})$ and the upper bounds on the execution times per abstract trace if the WCET bound can be tightened by leaving out some infeasible abstract traces. Consider the particular case that the calculation of the WCET bound is dominated by an infeasible abstract trace. Further assume that each feasible abstract trace has an upper bound on its execution times that is strictly smaller than the calculated WCET bound. Then we can obtain a strictly smaller WCET bound by basing its calculation only on the feasible abstract traces. In fact, this proves that the precision of the WCET bound can be improved by pruning infeasible abstract traces.

VI. 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* be a set of properties of the system under consideration:

$$\begin{aligned} \mathit{Prop} &= \{P_1, \ldots, P_{\#\mathit{Prop}}\} \\ \forall t \in \mathit{Traces} &: \forall P_i \in \mathit{Prop} : P_i(t) \end{aligned} \tag{10}$$

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 $\widehat{P}_i$ such that:

$$\begin{aligned}&\forall \hat{t} \in \widehat{\mathit{Traces}} : \\ &\quad [\, \exists t \in \gamma_{\mathit{trace}}(\hat{t}) : P_i(t)\,] \Rightarrow \widehat{P}_i(\hat{t})\end{aligned} \tag{11}$$

The intuition behind that requirement gets more clear if we have a look at what it means if $\widehat{P}_i$ does not hold for an abstract trace $\hat{t} \in \widehat{\mathit{Traces}}$:

$$\begin{aligned}&\neg \widehat{P}_i(\hat{t}) \\ &\underset{(11)}{\Rightarrow} \forall t \in \gamma_{\mathit{trace}}(\hat{t}) : \neg P_i(t) \\ &\underset{(10)}{\Rightarrow} \gamma_{\mathit{trace}}(\hat{t}) \cap \mathit{Traces} = \emptyset \\ &\underset{(5)}{\Leftrightarrow} \hat{t} \in \widehat{\mathit{Infeas}}\end{aligned} \tag{12}$$

So if a lifted 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.

VII. Property Lifting Example

The following example will illustrate how we can find a $\widehat{P}_i(\hat{t})$ satisfying (11) without using $\gamma_{\mathit{trace}}(\hat{t})$ directly, which is mandatory for an efficient use of an abstract model.

Assume that we have an upper bound on the number of bus accesses performed by a particular processor core C per abstract trace.

$$\begin{aligned}&\forall \hat{t} \in \widehat{\mathit{Traces}} : \\ &\quad \forall t \in \gamma_{\mathit{trace}}(\hat{t}) : \\ &\qquad {}^{\mathit{UB}}\#\mathit{accesses}_C(\hat{t}) \geq \#\mathit{accesses}_C(t)\end{aligned} \tag{a}$$

We only use γ_{trace} 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 is blocked at a shared bus per abstract trace.

$$\begin{aligned}&\forall \hat{t} \in \widehat{\mathit{Traces}} : \\ &\quad \forall t \in \gamma_{\mathit{trace}}(\hat{t}) : \\ &\qquad {}^{\mathit{LB}}\#\mathit{blockedCycles}_C(\hat{t}) \leq \#\mathit{blockedCycles}_C(t)\end{aligned} \tag{b}$$

Now assume that the concrete system we consider uses a Round-Robin policy to arbitrate its shared bus. Therefore, all its execution behaviors shall fulfill the property P_{rr}:

$$\begin{aligned}P_{rr}(t) \Leftrightarrow [\, &\#\mathit{blockedCycles}_C(t) \\ &\leq \#\mathit{accesses}_C(t) \cdot (\#\mathit{Cores} - 1) \\ &\quad \cdot \mathit{maxCyclesPerAccess}\,]\end{aligned} \tag{c}$$

The intuition behind this property (implicitly assumed in [12]) is that with Round-Robin arbitration, each concurrent core (there are $\#\mathit{Cores} - 1$ of them) can at most perform one access to the bus before an access of core C 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 can be blocked at the bus. Knowledge about how many accesses to the bus are performed by core C allows us to bound the overall amount of bus blocking experienced by core C in a particular execution behavior.

We can safely lift P_{rr} to abstract traces in a way that satisfies (11) by applying (a) and (b):

$$\begin{aligned}&\forall \hat{t} \in \widehat{\mathit{Traces}} : \\ &\quad \exists t \in \gamma_{\mathit{trace}}(\hat{t}) : P_{rr}(t) \\ &\underset{(c)}{\Leftrightarrow} \exists t \in \gamma_{\mathit{trace}}(\hat{t}) : \\ &\qquad \#\mathit{blockedCycles}_C(t) \\ &\qquad\quad \leq \#\mathit{accesses}_C(t) \cdot (\#\mathit{Cores} - 1) \\ &\qquad\quad \cdot \mathit{maxCyclesPerAccess} \\ &\underset{(a)(b)}{\Rightarrow} {}^{\mathit{LB}}\#\mathit{blockedCycles}_C(\hat{t}) \\ &\qquad \leq {}^{\mathit{UB}}\#\mathit{accesses}_C(\hat{t}) \cdot (\#\mathit{Cores} - 1) \\ &\qquad\quad \cdot \mathit{maxCyclesPerAccess} \\ &\Leftrightarrow: \widehat{P_{rr}}(\hat{t})\end{aligned} \tag{d}$$

$\widehat{P_{rr}}$ as defined in (d) clearly satisfies the soundness criterion (11) for lifted properties. According to (12) any abstract trace $\hat{t}$ with $\neg\widehat{P_{rr}}(\hat{t})$ can safely be considered as infeasible.

VIII. Improving the Approximation

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

$$\begin{aligned}&\forall \hat{t} \in \widehat{\mathit{Traces}} : \\ &\quad \widehat{P}(\hat{t}) \Leftrightarrow \forall P_i \in \mathit{Prop} : \widehat{P}_i(\hat{t})\end{aligned} \tag{13}$$

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

$$\begin{aligned}&\neg \widehat{P}(\hat{t}) \\ &\underset{(13)}{\Leftrightarrow} \exists P_i \in \mathit{Prop} : \neg \widehat{P}_i(\hat{t}) \\ &\underset{(12)}{\Rightarrow} \hat{t} \in \widehat{\mathit{Infeas}}\end{aligned} \tag{14}$$

We can use $\widehat{P}$ to define an alternative set $\widehat{\mathit{LessTraces}}$ of abstract traces based on $\widehat{\mathit{Traces}}$:

$$\widehat{\mathit{LessTraces}} = \{\hat{t} \mid \hat{t} \in \widehat{\mathit{Traces}} \wedge \widehat{P}(\hat{t})\} \tag{15}$$

$\widehat{\mathit{LessTraces}}$ is the subset of abstract traces in $\widehat{\mathit{Traces}}$ that cannot be classified as infeasible by any of the $\widehat{P}_i$. It follows from (7), (14) and (15) that $(\widehat{\mathit{LessTraces}}, \gamma_{\mathit{trace}})$ is a member of $\mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})}$.

$$(\widehat{\mathit{LessTraces}}, \gamma_{\mathit{trace}}) \in \mathit{Deriv}_{(\widehat{\mathit{Traces}}, \gamma_{\mathit{trace}})} \tag{16}$$

As a consequence of (9) and (16), we can derive a sound WCET bound from $(\widehat{\mathit{LessTraces}}, \gamma_{\mathit{trace}})$:

$$\max_{\hat{t} \in \widehat{\mathit{LessTraces}}} {}^{\mathit{UB}}\mathit{et}_C(\hat{t}) \geq \mathit{WCET}_C \tag{17}$$

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

IX. Reducing the Simplifying Assumptions

So far, we assume that each core only runs one program and that each program may at most be executed once per system run. Please note that these restrictions are not inherent to our meta approach. They are ment to facilitate the focus on the essential ideas.

Our meta approach can easily be extended to support several programs per processor core by introducing a set *Programs* of program identifiers. Such an extension will assume the existence of et_{Prog} and ${}^{UB}et_{Prog}$ for each program $Prog \in Programs$. We define the WCET of program *Prog* as follows:

$$WCET_{Prog} = \max_{t \in Traces} et_{Prog}(t)$$

It is straightforward to derive an upper bound to this WCET based on an abstract model that is an overapproximation of the considered system's behaviors:

$$\max_{\hat{t} \in \widehat{Traces}} {}^{UB}et_{Prog}(\hat{t}) \geq WCET_{Prog}$$

In case a program *Prog* can be executed more than once per system run, it is no longer possible to assign a single execution time per program to each execution behavior. Therefore, we use the helper function $runs_{Prog}$ to extract the different execution runs of *Prog* from an execution behavior. In this context, et_{Prog} assigns an execution time to each execution run of *Prog*. An extended definition of the WCET of *Prog* incorporates the execution runs of *Prog*:

$$WCET_{Prog} = \max_{t \in Traces} \max_{run \in runs_{Prog}(t)} et_{Prog}(run)$$

However, the previous definition of a WCET bound for *Prog* can anyway be reused provided that ${}^{UB}et_{Prog}$ fulfills the following criterion:

$$\forall \hat{t} \in \widehat{Traces} : \\ {}^{UB}et_{Prog}(\hat{t}) \geq \max_{t \in \gamma_{trace}(\hat{t})} \max_{run \in runs_{Prog}(t)} et_{Prog}(run)$$

X. Conceptual Approach and Generality

It should be noticed that we describe a *conceptual approach*. Implementations do not necessarily have to stick to its two-step character of first accumulating the set $\widehat{Traces}$ and then sorting out provably infeasible abstract traces.

Note that we did not restrict the form or structure of abstract traces by any means. Therefore, we expect that our meta approach can be mapped to the different ways of approximation used in WCET analyses.

Furthermore, the use of our meta approach is by no means restricted to a scenario of WCET analysis for multi-core processors. Whenever there exists an abstract model that provides an overapproximation of a system's behavior, our meta approach can serve to further improve the precision by additional properties of the system.

XI. Future Work

We plan to instantiate our meta approach for the aiT WCET analyzer[1]. In that way, we intend to come up with an overall approach that supports a wide range of complex processor core features.

In addition to the instantiation of the meta approach for a powerful baseline abstraction, it will be crucial to find a reasonable set of system properties for each supported processor. Those properties have to bound the shared resource interference in a way such that sufficiently tight WCET bounds can be obtained.

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 [12], [8]. If an abstract model only focuses on one processor core, then it has to assume arbitrary spurious behaviors for the other cores. The lifted version of a property relating the behavior of the focused core to that of others would have to pessimistically assume the other cores to behave in a way that the property always holds. Therefore, our meta approach currently only profits from such properties if it is used in combination with a baseline abstract model that argues in detail about several processor cores at the same time. But abstract models that argue in detail about several processor cores can be seen as combinatorially too complex. A goal of future work will be to allow for the use of baseline abstract models that focus on one processor core, but that still enable us to incorporate sound assumptions about the behavior of concurrent processor cores.

[1] http://www.absint.com/ait

XII. Conclusion

We presented a conceptual meta approach to WCET analysis for multi-core processors. It points out a common methodology behind previous approaches. Yet, it does not depend on a particular way of approximating the system under consideration. Bounds on the shared resource interference of the concrete system can uniformly be incorporated to improve the precision of the obtained WCET bound.

Acknowledgment

The author would like to thank Sebastian Hahn and Jan Reineke for many comments and interesting discussions.

References

[1] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *Proceedings of the ninth European Dependable Computing Conference*, 2012, pp. 132–143.

[2] A. Abel *et al.*, "Impact of resource sharing on performance and performance prediction: A survey," in *CONCUR*, 2013, pp. 25–43.

[3] R. Wilhelm *et al.*, "The worst-case execution-time problem — overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[4] P. Puschner, "The single-path approach towards WCET-analysable software," in *Proceedings of the IEEE International Conference on Industrial Technology*, vol. 2, 2003, pp. 699–704.

[5] S. Thesing, "Safe and precise WCET determination by abstract interpretation of pipeline models," Ph.D. dissertation, 2004.

[6] X. Li *et al.*, "Modeling out-of-order processors for WCET analysis," *Real-Time Syst.*, vol. 34, no. 3, pp. 195–227, Nov. 2006.

[7] A. Schranzhofer *et al.*, "Timing analysis for TDMA arbitration in resource sharing systems," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2010, pp. 215–224.

[8] R. Pellizzoni *et al.*, "Worst case delay analysis for memory interference in multicore systems," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2010, pp. 741–746.

[9] A. Schranzhofer *et al.*, "Timing analysis for resource access interference on adaptive resource arbiters," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2011, pp. 213–222.

[10] Y. Liang *et al.*, "Timing analysis of concurrent programs running on shared cache multi-cores," *Real-Time Systems*, vol. 48, pp. 638–680, 2012.

[11] S. Chattopadhyay *et al.*, "A unified WCET analysis framework for multi-core platforms," in *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2012, pp. 99–108.

[12] R. Pellizzoni and M. Caccamo, "Impact of peripheral-processor interference on WCET analysis of real-time embedded systems," *IEEE Transactions on Computers*, vol. 59, pp. 400–415, 2010.

[13] D. Dasari *et al.*, "WCET analysis considering contention on memory bus in COTS-based multicores," in *Proceedings of the 16th IEEE Conference on Emerging Technologies Factory Automation*, 2011, pp. 1–4.