# Obstacles in Worst-Case Execution Time Analysis *

Raimund Kirner, Peter Puschner
Institut für Technische Informatik
Technische Universität Wien
Treitlstraße 3/182/1
A-1040 Wien, Austria
{raimund,peter}@vmars.tuwien.ac.at

## Abstract

*The analysis of the worst-case execution time (WCET) requires detailed knowledge of the program behavior. In practice it is still not possible to obtain all needed information automatically.*

*In this paper we present the current state of the art of WCET analysis and point to the main problems to be solved. The most eminent problem is the* state problem*, i.e., the precise determination of possible processor states at different program locations. The* path problem *refers to the fact that current tools are not able to calculate all (in)feasible paths automatically. We discuss how the main open problems manifest themselves in static and in measurement-based WCET analysis methods.*

## 1 Introduction

The knowledge of the worst-case execution time (WCET) of software components is a prerequisite for ensuring the timeliness of a real-time system. Since the end of the 1980s significant effort has been spent on research towards the development of WCET analysis tools.

The two main tasks of WCET analysis tools are the *control-flow analysis* (also called *path analysis* [5] or *high-level analysis*) that determines the (in)feasible paths in a program and the *processor-behavior analysis* (also known as *hardware modeling* [5] or *low-level analysis*) that assesses instruction timing [15]. Within this paper we discuss open problems of these two tasks of WCET analysis.

There are currently two main approaches towards WCET analysis, the static analysis and the measurement-based analysis. The latter has been developed more recently and provides an alternative to static analysis when retargetability is an issue. Adapting a static analysis tool to new hardware is very expensive. In contrast, measurement-based WCET analysis can typically be retargeted with modest effort. We discuss the open problems in WCET analysis for both approaches.

As the hardware models constructed for deterministic WCET analsis methods and in empiric WCET analysis methods are of different quality, we investigate the preconditions that must be fulfilled so that a *WCET estimate* is a *WCET bound*.

## 2 Requirements on WCET Analysis Tools

From a theoretical point of view, WCET analysis is perfectly decidable, because real computers have a finite state space. From the practical point of view however, we face the complexity of real computer systems that have too many states to be discretely enumerated for a full analysis. Thus, the main topic of WCET analysis is to find feasible abstractions and analysis methods such that:

1. the necessary development effort of the WCET analysis tool is affordable,
2. the calculated WCET estimates are sufficiently precise,
3. the analysis problems are tractable with acceptable resource requirements, and

*The research leading to these results has received funding from the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) within the research project "Compiler-Support for Timing Analysis" (COSTA) and the European Community's Sixth Framework Programme [FP6/2002-2006] under grant agreement n004527 (ARTIST2, http://www.artist-embedded.org/).

4. the WCET tool is easy to use.

There is no general acceptance criterion for each of these requirements. What is acceptable depends on the application domain.

For example, in the avionics or space industry, there is high pressure to ensure that the timing of the software goes right. Thus, people accept high cost and high efforts to get a WCET bound that has a neglectably low probability of underestimation. On the other hand, the high cost for re-validating the software in case of even small changes is a serious problem.

In other industry sectors that do not have that high safety demands, the requirements on WCET tools may be weighted completely different. For example, in highly cost-optimized sectors of mass production the computer platform may change quite frequently, because production costs are more important than development costs. In this case the developer would need timing analysis for several different platforms to select the cheapest one that is powerful enough to guarantee the deadlines.

Here, the development effort of the tool, especially the effort and cost for retargeting the WCET analysis tool to a new hardware platform, becomes quite important. Evaluating the preciseness or safety of the WCET estimate is not of first concern for such a platform. Also during development time a a rough and fast WCET analysis may be preferable over a precise analysis that demands high computation resources and user interaction.

Thus, each of the two WCET assessment strategies described in Section 5 has its own well-suited application domain.

## 3 WCET Estimate or WCET Bound?

In the early years of WCET analysis, i.e., at the end of the 1980s, processors with a relatively simple timing behavior were analyzed. At this time there was no doubt that a calculated WCET value was a safe upper bound of the real WCET.

Facing the complexity of most of today's processors, it is almost impossible to prove whether a hardware model of the processor correctly covers the real WCET. For example, the WCET analysis for the Motorola ColdFire 5307 described in [8] assumes an empty cache for the WCET analysis (worst-case assumption) whenever the cache content is unknown. This strategy fails if the cache is configured as "copy back". Furthermore, measurement-based WCET analysis techniques have emerged that determine the hardware model empirically.

Thus, we have to review whether the results of today's WCET analysis methods should be called a *WCET bound* or in a more modest form a *WCET estimate*. The term *WCET bound* seems to be adequate only for the results of WCET analysis methods/tools that were explicitly constructed with the aim to fully cover the worst-case timing behavior. In this case, whenever a deficiency of the hardware model has been discovered, it can be corrected.

In contrast, with heuristic methods one cannot guarantee that at least the known worst-case timing behavior will be covered by the WCET analysis.

Measurement-based WCET analysis techniques are empiric methods that typically use heuristics to construct the hardware model. However, it is also possible to use measurement-based WCET analysis without heuristics, e.g., by enforcing hardware states at those locations where the hardware state would otherwise vary.

## 4 Problems in WCET Analysis

Before looking on current solutions, we review the main obstacles towards WCET analysis.

### 4.1 The Path Problem

To put it simple, WCET analysis is about finding the longest feasible path through a program, where length means execution time.

To find the longest path, we first have to determine the set of all feasible program paths, or at least a tight approximation thereof. Control-flow analysis can be used to find (in)feasible paths automatically. Here we already face a serious complexity problem, as the number of different paths may grow exponentially with program size.

Any brute-forth approach like executing or simulating the program with all possible input data is doomed to fail due to the high number of paths and typically even higher number of different values of input data.

Shifting the path problem from the WCET analysis tool to the user by asking him for additional control-flow information to be provided in code annotations is not a desired solution either. Providing this information requires expert knowledge, a lot of manpower that potentially has to be re-invested whenever the program changes, and is also a quite error-prone approach.

### 4.2 The State Problem

After solving the *path problem* we have to calculate the execution time of each path by summing up the

execution times of all instructions along that path.

This simplified strategy becomes quickly intractable due to the potentially large number of different paths in a program. Thus, the obvious solution to this problem is to use *divide and conquer* by local WCET calculations, i.e., if we have an `if-else` statement we calculate the WCET along the `then` branch and along the `else` branch and take the maximum of it. Then we virtually replace the `if-else` statement with an artificial statement whose execution time is the calculated maximum. Repeating this in a bottom-up manner for all branches in the control-flow graph yields a conservative approximation of the WCET. In principle, this strategy is used by the tree-based [11] and the path-based [2] WCET calculation method.

### 4.2.1 Non-Locality of Instruction Timing

Local WCET calculation techniques work well on processors whose instruction timing does not depend on the current processor state (Definition 4.1), i.e., architectures where the TRPS (Definition 4.2) is (almost) empty.

**Definition 4.1 Processor State:** *The processor state is defined by the content of all memory elements inside the processor.*

**Definition 4.2 Timing-Relevant Processor State (TRPS)** *The timing-relevant processor state TRPS consists of those elements of the processor state PS whose values can influence the execution time of at least one instruction. Conversely, the values of the state elements not included in TRPS do not influence the instruction timing.*

Modern processor architectures host many features for improving the peak performance, whose timing depends on the current processor state. Examples of such hardware features are caches, pipelines, speculation units, etc.

In case the timing of instructions does not depend on the processor-state, the timing of each instruction can be derived by looking at single instructions only. But to calculate an instruction timing that depends on the processor state one has to apply the principle "to predict the future one has to learn the past". More technically speaking, the execution history has to be taken into account to predict the processor state at the start of the execution of a particular instruction. Given an initial processor state, the processor's state is advanced by the instructions executed along the control-flow path. In addition, the content of the data memory influences the processor state through memory-read operations.

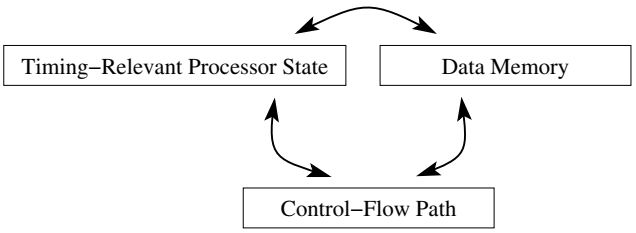


**Figure 1. Interference between taken control-flow, content of data memory, and the timing-relevant processor state (TRPS)**

The challenge of using local WCET calculation for modern processors is that there are typically interferences between the control-flow path, the data memory, and the TRPS (see Figure 1). The consequence of these interferences is:

**Theorem 4.3 Non-Locality of Instruction Timing:** *If there is an interference of the TRPS with*

1. *the taken control flow, or*
2. *the content of the data memory, then*

*the calculation of a WCET bound using local WCET calculations is in general not possible without overestimation.*

**Proof** Whenever there is an interference of the TRPS with the taken control-flow, the instruction timing of an instruction $I_j$ depends on the TRPS $s$: $t_{I_j} = T(I_j, s)$. The precise WCET bound for a set of reachable states $RS$ is given by $WCET(I_j, RS) = max(\{T(I_j, s) | s \in RS\})$. Whenever for an instruction $I_j$ the set $RS$ of reachable states is only a subset of the state space $S$ ($RS \subset S$), it may happen that $\exists s_1 \in S\ \forall s_2 \in FS : T(I_j, s_1) > T(I_j, s_2)$. Due to global control-flow decisions, local WCET calculation ($WCET_{cl}(I_j)$) does not know which states are reachable. Thus, it has to consider the whole state space $S$, resulting in an overestimation for the assumed case: $WCET_{cl}(I_j) = max(\{T(I_j, s) | s \in S\}) > WCET(I_j, RS)$

Second, if there is an interference of the TRPS with the content of the data memory, on each read operation the values of the data memory become part of the TRPS. If the read data elements have not been written locally, their content is unknown, i.e., the read operation forces some elements of TRPS to become locally unknown. As for the first case, we have to consider the whole state space $S$ for unknown elements in the TRPS, which again can result in an overestimation of execution time. □

The second case of Theorem 4.3 shows that not only the lack of control-flow information leads to overestimation. Even in straight-line code without control-flow decisions overestimation of execution time may occur.

In most real processors there is an interference between the taken control flow, the content of the data memory, and the TRPS. Therefore, a global program analysis is needed to calculate a precise instruction timing. Compared to the control-flow analysis for finding (in)feasible paths, the hardware-state analysis has a higher urge for automatism, as a user of the WCET analysis tool is typically not capable to annotate sets of feasible hardware states. Of course, manual annotations about the (in)feasible paths are also error-prone and labor-intensive, but the software developer is at least able to derive this information from the code.

Thus, in practice, hardware-state analysis is done for the whole program with relatively low hardware-state abstractions. The maximal possible precision of the analysis comes at high computational cost. To simplify the analysis, one can partion the TRPS with respect to the different hardware components. For example, some WCET tools analyze the pipeline behavior and cache behavior separately.

Such hardware state partitioning techniques are of limited applicability.

Section 4.2.2 presents the applicability requirements for two hardware-state partitioning techniques.

### 4.2.2 Timing-Composable TRPS Partitioning

The idea of timing-composable partitioning is to calculate the WCET of an instruction sequence $I = I_0 \circ I_1 \circ \ldots \circ I_n$ in two steps. The TRPS $S$ is partitioned into $S = A + B$, where $A$ is the state space of some processor component $A$ and $B$ is the state space of other components in the processor. For example, the hardware component $hw_A$ may be the instruction cache and the state fraction $B$ covers the pipeline and the other processor components.

In the first step, the timing of processor component $hw_A$ is analyzed. Based on this result, the overall processor timing is analyzed in the second step by searching the state space $B$ while keeping state $A$ constant.

The following notation is used to explain the concrete partitioning techniques:

$T(I, s)$ ... the timing of an instruction sequence $I = I_0 \circ I_1 \circ \ldots \circ I_n$ with the initial processor state $s$.

$T_{hw_A}(I, a)$ ... the cumulated time a processor component $hw_A$ is active when executing instruction sequence $I$ with initial local state $a \in A$. A processor component is said to be active whenever it performs some data processing. Conversely, it is inactive whenever it only holds its state without data processing.

$T_{max}(I) = max(\{T(I, s) \mid s \in S\})$ ... the WCET of $I$.

$\Delta(I, s, s') = T(I, s) - T(I, s')$ ... timing difference between initial states $s$ and $s'$.

$\Delta_{hw_A}(I, a, a') = T_{hw_A}(I, a) - T_{hw_A}(I, a')$ ... activation-time difference of processor component $hw_A$ between local initial states $a$ and $a'$.

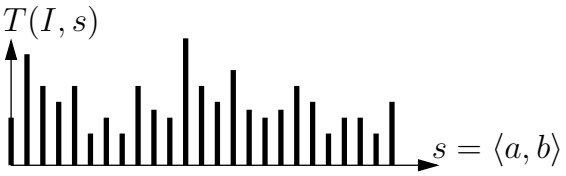


**Figure 2. Timing of Non-Partitioned TRPS**

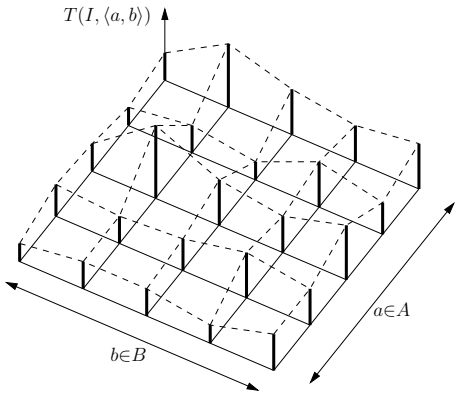


**Figure 3. Timing of Partitioned TRPS**

Figure 2 shows an example of the timing function $T(I, s)$ of an instruction sequence $I$. As expected from its definition, the execution time depends on the TRPS $s \in S$. The execution time of $T(I, s)$ based on the partitioning of the TRPS into two sets $A$ and $B$ is shown in Figure 3. The challenge is to find a composable timing calculation method that can be used to calculate safe WCET bounds for the target processor of interest. We describe two different timing-composition techniques.

**Delta-Composition of TRPS** The principle of *delta-composition* is given in Figure 4: 1) $\Delta_{hw_A,max}$,

the maximum variability ($\Delta$) of $T_{hw_A}(I, a)$ is determined; 2) $a_{hw_A,min}$, the local state where $T_{hw_A}(I, a)$ is minimal, is determined; 3) the overall timing function $T(I, \langle a_{hw_A,min}, b\rangle)$ with fixed local state $a_{hw_A,min}$ is selected; 4) $b_{dc,max}$, the partial state $b \in B$ where $T(I, \langle a_{hw_A,min}, b\rangle)$ is maximal, is determined; 5) $\Delta_{hw_A,max}$ is added to $T(I, \langle a_{hw_A,min}, b_{dc,max}\rangle)$.

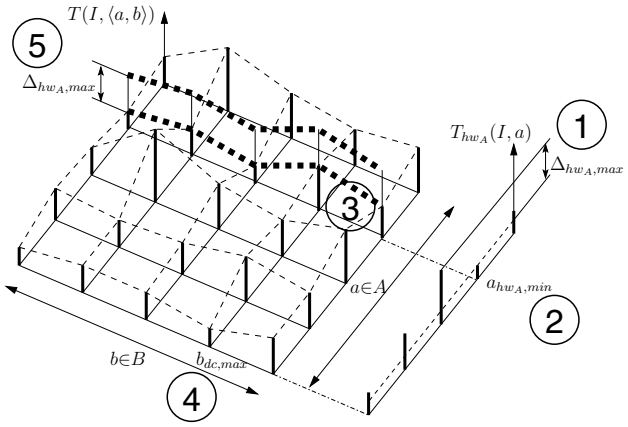


**Figure 4. Delta-Composition of TRPS**

Equation 1 shows how the instruction timing $T_{dc}(I)$ is calculated with *delta-composition*.

$$\begin{aligned} T_{dc}(I) &= T(I, \langle a_{hw_A,min}, b_{dc,max}\rangle) + \Delta_{hw_A,max} \\ \Big(T_{dc}(I) &\geq T_{max}(I)\Big) \end{aligned} \quad (1)$$

The delta-composition can provide a safe WCET bound only on processor hardware whose timing characteristics obey Equation 2.

$$\begin{aligned} \forall a, a' \in A, b \in B.\ \Delta_{hw_A}(I, a, a') > 0 \rightarrow \\ \Delta(I, \langle a, b\rangle, \langle a', b\rangle) \leq \Delta_{hw_A}(I, a, a') \end{aligned} \quad (2)$$

Equation 2 states that whenever the state of a hardware component $hw_A$ changes from state $a$ to state $a'$, the resulting change in the execution time of instruction-sequence $I$ ($\Delta(I, \langle a, b\rangle, \langle a', b\rangle)$) is not higher than the change in the activation time of the hardware component $hw_A$ ($\Delta_{hw_A}(I, a, a')$).

**Max-Composition of TRPS** The principle of *max-composition* is given in Figure 5: 1) $a_{hw_A,max}$, the local state where $T_{hw_A}(I, a)$ is maximal, is determined; 2) the overall timing function $T(I, \langle a_{hw_A,max}, b\rangle)$ with fixed local state $a_{hw_A,max}$ is selected; 3) $b_{mc,max}$, the partial state $b \in B$ where $T(I, \langle a_{hw_A,max}, b\rangle)$ is maximal, is determined.

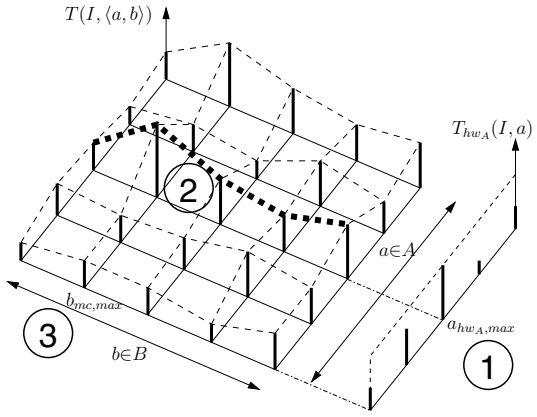


**Figure 5. Max-Composition of TRPS**

Equation 3 shows the calculation of the instruction timing $T_{mc}(I)$ with *max-composition*.

$$\begin{aligned} T_{mc}(I) &= T(I, \langle a_{hw_A,max}, b_{mc,max}\rangle) \\ \Big(T_{mc}(I) &= T_{max}(I)\Big) \end{aligned} \quad (3)$$

The max-composition can provide a safe WCET bound only on processor hardware whose timing characteristics obey Equation 4.

$$\begin{aligned} \forall a, a' \in A, b \in B.\ T_{hw_A}(I, a) > T_{hw_A}(I, a') \rightarrow \\ T_{hw_A}(I, \langle a, b\rangle) \geq T_{hw_A}(I, \langle a', b\rangle) \end{aligned} \quad (4)$$

Equation 4 states that whenever the state of hardware component $hw_A$ changes from state $a$ to state $a'$ and as a consequence the activation time of hardware component $hw_A$ decreases ($T_{hw_A}(I, a) > T_{hw_A}(I, a')$), then the execution time of instruction sequence $I$ must not increase ($T_{hw_A}(I, \langle a, b\rangle) \geq T_{hw_A}(I, \langle a', b\rangle)$).

**Timing Anomalies** With all the complex features in modern processors it can happen that the local timing maximum of a certain hardware feature does not correspond with the processor's timing maximum. Such behavior is already known in the scheduling domain, but Lundqvist at al. pointed out that such timing anomalies are also relevant for WCET analysis [9]. Wenzel et al. found that timing anomalies can only occur in processors with dynamic resource allocations [14]. Reineke et al. pointed out that besides the known timing-anomalies examples based on scheduling,

timing anomalies can also be caused by *speculative execution* and the *pseudo-round robin* cache replacement strategy [13].

**Definition 4.4 (Timing Anomaly)** *Given a partitioned TRPS $S = A + B$ with the timing behavior of hardware component $hw_A$ modeled as $T_{hw_A}(I, a)$, the timing behavior $T(I, \langle a, b\rangle)$ of an instruction sequence $I$ on a processor is called a* timing anomaly*, iff at least one of the following two properties holds:*

**TA1:** $\exists a, a' \in A, b \in B.$
$\Delta_{hw_A}(I, a, a') > 0 \wedge \ \Delta(I, \langle a, b\rangle, \langle a', b\rangle) < 0$

**TA2:** $\exists a, a' \in A, b \in B.$
$0 < \Delta_{hw_A}(I, a, a') < \Delta(I, \langle a, b\rangle, \langle a', b\rangle)$

To show how timing anomalies are connected with the problem of state analysis, Definition 4.4 defines timing anomalies in terms of changes of the hardware state.

**Theorem 4.5 Timing-Composability and Timing Anomalies:** *On processor hardware exhibiting timing anomalies of type* **TA1** *the* delta-composition *still provides safe WCET bounds, but the* max-composition *does not. Conversely, on processor hardware exhibiting timing anomalies of type* **TA2** *the* max-composition *still provides safe WCET bounds, but the* delta-composition *does not. (proof omitted)*

**Corollary 4.6** *Concluding from Theorem 4.5, processor hardware exhibiting timing anomalies of at most one of the types* **TA1** *or* **TA2** *can be safely analyzed by applying a combination of* delta-composition *(Equation 1) and* max-composition *(Equation 3):*

$$T_{dmc}(I) = max\Big(T_{dc}(I),\ T_{mc}(I)\Big)$$
$$\Big(T_{dmc}(I) \geq T_{max}(I)\Big)$$

### 4.3 The Translation Problem

With today's compilers there is the problem that control-flow information that is extracted from or annotated at source-code level cannot be directly used for WCET analysis, because the code optimizations performed by the compiler may invalidate this control-flow information. One has the choice between disabling all code optimizations or annotating flow information on the output level of the compiler, the assembly program.

Techniques have been already developed to transform flow information with help of the compiler for arbitrary code optimizations [4]. However, besides a few research compilers, support of this technique is still missing in current compilers.

## 5 WCET Analysis Techniques

In the following the research challenges of the two main WCET analysis methods and the implications from Section 4 are described.

### 5.1 Static WCET Analysis

Static WCET analysis uses program analysis and explicit hardware models to derive a WCET bound. The main challenge static WCET analysis is facing today is the *state problem*. The analysis of the *path problem* has not been fully automatized as well, but the *state problem* is more severe, due to the high complexity of today's processors and the short hardware life cycles.

Static WCET analysis with separated cache and pipeline analyzes [1] have been found inadequate for modern processors. Thus, WCET analysis tools with integrated cache and pipeline analysis have been developed [8]. This integrated cache and pipeline analysis already hits the complexity wall of today's processors.

Initially, separated cache and pipeline analyzes have been combined for the sake of precision, resulting in very high analysis times. Symbolic representations are currently considered to reduce the memory demands of the analysis. As an interesting insight to the problem, Corollary 4.6 can help to construct a timing-composable WCET analysis tool in case that for a concrete state partitioning maximally one of the timing anomaly types **TA1** or **TA2** does occur.

### 5.2 Measurement-Based WCET Analysis

Measurement-based WCET analysis techniques use execution-time measurements to construct a hardware model [7]. Without further provisions, measurement-based WCET analysis techniques are heuristic approaches for obtaining *WCET estimates*.

One of the main challenges of measurement-based WCET analysis is the automatic and systematic generation of test data for execution-time measurements. Further research is also needed on the *state problem* to achieve a sufficient coverage of hardware states with a reasonable number of test data. The use of highly abstracted, non-timed hardware models is considered for guiding the generation of test-data.

## 6 Conclusion and Outlook

Despite of almost twenty years of highly active research on WCET analysis, there are still serious open problems in the area. The most challenging problem is the high complexity of today's processors. Features like caches and pipelines create a huge state space. Static WCET analysis tools and measurement-based WCET analysis tools have evolved as complementary approaches, but both suffer from the *state problem*. Also the fully automatic path analysis without user interaction is an open problem in both approaches.

Besides the WCET analysis problems discussed in this paper there are further research issues to be solved towards the verification of the global timing of a real-time system. One issue is the clear separation of WCET analysis and scheduling analysis. The flaw in common software and hardware designs is that no clear separation is possible.

Thus, more predictable software and hardware designs are needed to overcome the complexity problem. The improved predictability would also simplify WCET analysis itself. For example, a design is generally more predictable if decisions are already resolved at compile time rather than at runtime [3]. The predictability of caches depends on the implemented replacement policy [12]. A fully time-predictable software and hardware architecture based on single-path execution and periodic task activation is proposed in [10, 6]. WCET analysis of such an architecture is easy.

Paradoxically, people working on probabilistic WCET analysis seek for more randomness in the timing behavior of systems, as this would simplify the probabilistic composition of probabilistic timing results.

## References


[1] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In *Proc. of the 1st International Workshop on Embedded Software (EMSOFT 2001)*, pages 469–485, Tahoe City, CA, USA, Oct. 2001.

[2] C. A. Healy, R. D. Arnold, F. Mueller, D. Whalley, and M. G. Harmon. Bounding pipeline and instruction cache performance. *IEEE Transactions on Computers*, 48(1), Jan. 1999.

[3] A. Kadlec and R. Kirner. On the difficulty of building a precise timing model for real-time programming. In *14. Kolloquium Programmiersprachen und Grundlagen der Programmierung*, Timmendorfer Strand, Germany, Oct. 2007.

[4] R. Kirner. *Extending Optimising Compilation to Support Worst-Case Execution Time Analysis*. PhD thesis, Technische Universität Wien, Vienna, Austria, May 2003.

[5] R. Kirner and P. Puschner. Classification of WCET analysis techniques. In *Proc. 8th IEEE International Symposium on Object-oriented Real-time distributed Computing*, pages 190–199, Seattle, WA, May 2005.

[6] R. Kirner and P. Puschner. Time-predictable task preemption for real-time systems with direct-mapped instruction cache. In *Proc. 10th IEEE International Symposium on Object-oriented Real-time distributed Computing*, pages 87–92, Santorini Island, Greece, May 2007.

[7] R. Kirner, I. Wenzel, B. Rieder, and P. Puschner. *Intelligent Systems at the Service of Mankind*, volume 2, chapter Using Measurements as a Complement to Static Worst-Case Execution Time Analysis, pages 205–226. UBooks Verlag, Augsburg, Germany, Jan. 2006. ISBN: 3-86608-052-2.

[8] M. Langenbach, S. Thesing, and R. Heckmann. Pipeline modeling for timing analysis. In *Proc. 9th International Static Analysis Symposium*, pages 294–309. Springer, Sep. 2002. LNCS 2477.

[9] T. Lundqvist and P. Stenström. Timing analysis in dynamically scheduled microprocessors. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 12–21, Dec. 1999.

[10] P. Puschner and R. Kirner. From time-triggered to time-deterministic real-time systems. In *Proc. 5th IFIP Working Conference on Distributed and Parallel Embedded Systems*, pages 115–124, Braga, Portugal, Oct. 2006.

[11] P. Puschner and C. Koza. Calculating the maximum execution time of real-time programs. *The Journal of Real-Time Systems*, 1:159–176, 1989.

[12] J. Reineke, D. Grund, C. Berg, and R. Wilhelm. Timing predictability of cache replacement policies. *Journal of Real-Time Systems*, 37(2):99–122, Nov. 2007.

[13] J. Reineke, B. Wachter, S. Tesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker. A definition and classification of timing anomalies. In *Proc. 6th International Workshop on Worst-Case Execution Time Analysis*, Dresden, Germany, July 2006.

[14] I. Wenzel, R. Kirner, P. Puschner, and B. Rieder. Principles of timing anomalies in superscalar processors. In *Proc. 5th International Conference of Quality Software*, Melbourne, Australia, Sep. 2005.

[15] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckman, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom. The worst-case execution time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, (Accepted January 2007).