

## Results

**Cite this article:** Schoeberl M, Jellum E, Lin S, Jerad C, Maroun EJ, Lohstroh M, and Lee EA (2025). Worst-case performance for real-time systems. *Research Directions: Cyber-Physical Systems*, 3, e2, 1–8. <https://doi.org/10.1017/cbp.2025.1>

Received: 16 January 2024

Revised: 10 December 2024

Accepted: 22 January 2025




**Keywords:**

real-time systems; time-predictable systems; precision timed machines

**Corresponding author:**

Martin Schoeberl; Email: [masca@dtu.dk](mailto:masca@dtu.dk)

# Worst-case performance for real-time systems

Martin Schoeberl<sup>1</sup> , Erling Jellum<sup>2</sup>, Shaokai Lin<sup>3</sup> , Chadlia Jerad<sup>4</sup>,  
Emad Jacob Maroun<sup>1</sup>, Marten Lohstroh<sup>3</sup> and Edward A. Lee<sup>3</sup> 

<sup>1</sup>Department of Applied Mathematics and Computer Science, Technical University of Denmark, Lyngby, Denmark;

<sup>2</sup>Norwegian University of Science and Technology, Trondheim, Norway; <sup>3</sup>University of California, Berkeley, Berkeley, CA, USA and <sup>4</sup>Universite de La Manouba, Manouba, Tunisia

## Abstract

Real-time systems need to be built out of tasks for which the worst-case execution time is known. To enable accurate estimates of worst-case execution time, some researchers propose to build processors that simplify that analysis. These architectures are called precision-timed machines or time-predictable architectures. However, what does this term mean? This paper explores the meaning of time predictability and how it can be quantified. We show that time predictability is hard to quantify. Rather, the worst-case performance as the combination of a processor, a compiler, and a worst-case execution time analysis tool is an important property in the context of real-time systems. Note that the actual software has implications as well on the worst-case performance. We propose to define a standard set of benchmark programs that can be used to evaluate a time-predictable processor, a compiler, and a worst-case execution time analysis tool. We define worst-case performance as the geometric mean of worst-case execution time bounds on a standard set of benchmark programs.

## Introduction

Within computer architecture and in the broader scope of natural science, our objective is to engage in measurement and quantification. Progress in computer architecture is marked by our ability to assert, for instance, that processor A outperforms processor B by a factor of 2. Naturally, we tend to favor the notion that “bigger is better.”

It is important to acknowledge that this oversimplified perspective does not always hold in practice, as the performance of processors also depends on the specifics of the workload in average-case scenarios. However, when we possess a well-defined workload, such as those represented by standard performance evaluation corporation (SPEC) benchmarks (Bucek et al. 2018), we can perform meaningful comparisons of this nature.

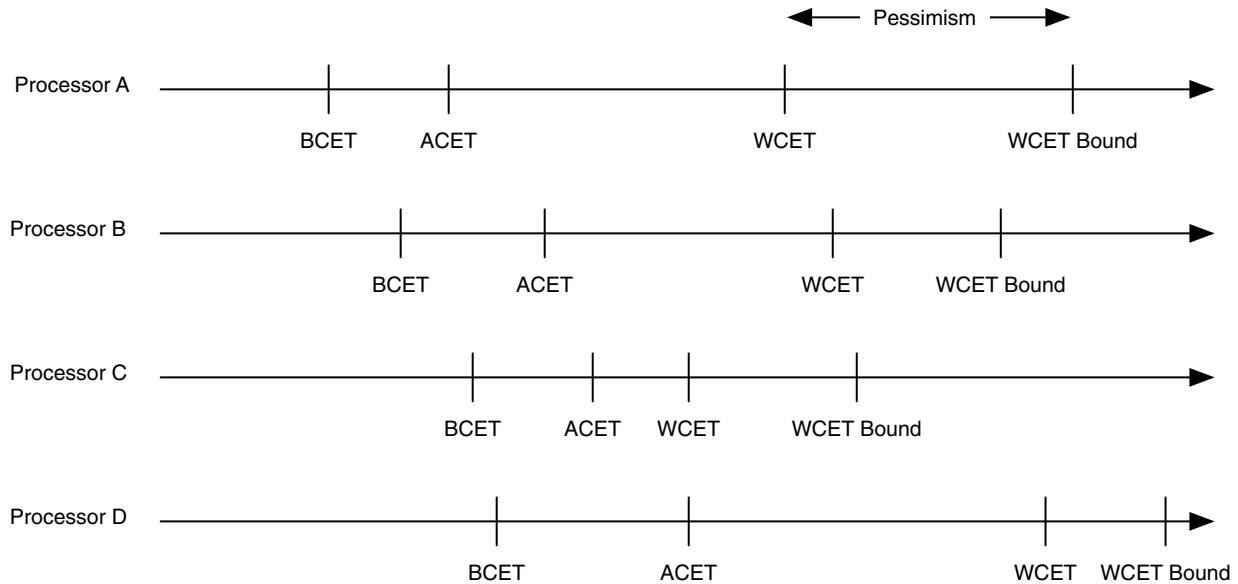
In real-time systems, a critical requirement is understanding the worst-case execution time (WCET). However, the complexity of modern processors makes it impossible to know the actual WCET. Instead, a bound can be determined, which must be safe (above the actual WCET) and precise (as close to the actual WCET as possible.) Both safety and precision of WCET bounds have become exceedingly challenging for a WCET analyzer. Consequently, researchers specializing in real-time systems and computer architecture have initiated efforts to design processors that simplify the static estimation of the WCET. These specialized architectures are widely considered “time-predictable.”

The resulting questions are: what is the precise definition of being “time-predictable” and can this property be quantified? Is there a means by which we can confidently assert, for instance, that “processor A exhibits better time predictability than processor B” or, even more precisely, that “processor A is twice as time-predictable as processor B”?

This paper tries to answer what time predictability means and whether it can be quantified. Because time predictability is hard to quantify, we introduce “benchmarking” of worst-case performance. The definition of the worst-case performance uses the geometric mean of the WCET bounds for programs from the TACLe benchmark suite (Falk et al. 2016). This definition is the first step in agreeing on how to compare different processors, compilers, and analyzers within the real-time systems community.

Schoeberl (2012) has initially asked that question. We have extended that paper by a rigorous literature review on definitions of time predictability. Furthermore, we argue for the notion of quantifying worst-case performance and provide an example of such quantification with WCET analysis of several real-time benchmarks on the Patmos processor.

© The Author(s), 2025. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.



**Figure 1.** Distribution of the best-case, average-case and worst-case execution times and the WCET bound of a task on different architectures.

### Background and problem statement

Thiele and Wilhelm (2004) argue for a novel research discipline dedicated to time-predictable embedded systems. Berg et al. (2004) outline key design principles for a time-predictable processor:

... recoverability from information loss in the analysis, minimal variation of the instruction timing, non-interference between processor components, deterministic processor behavior, and comprehensive documentation.

Computer architects and researchers specializing in real-time computing are investigating processors and architectural designs tailored specifically for real-time systems (Edwards and Lee 2007). When we refer to “optimized for real-time systems,” we mean architectures that exhibit some form of time predictability. This may mean statically determining a tight upper bound on the worst-case execution time (WCET).

To compare various approaches, it becomes essential to quantify this time predictability. In other words, we need methods to assess and gauge it accurately. This paper explores diverse techniques for conducting such quantification.

Schoeberl (2009) describes the architecture issues that make WCET analysis hard or even impossible. He then describes a time-predictable architecture. This paper does not define what time predictability means. However, its Figure 1 explains intuitively how a time-predictable processor differs from a commercial-of-the-shelf processor. We adopt this figure also as our Figure 1 to discuss the problem we want to solve with this paper.

Edwards et al. (2009) define an architecture with repeatable timing. Liu (2012) presents in his PhD thesis a precision timed machine (PRET) that is an implementation of an architecture with repeatable timing. Repeatable timing is a strong property where every execution of a sequence of instructions with the same input data leads to the same execution time. When we include the same initial internal state of a processor (e.g., clearing its caches), a time-predictable processor will also deliver repeatable timing.

Figure 1 shows the different execution times of a single program on four different processors (A, B, C, and D). The different execution times are: The best-case execution time (BCET), the

average-case execution time (ACET), and the worst-case execution time (WCET). Execution times between BCET and WCET are possible. The execution time depends on several factors: the program (task), the state of the processor, and input data. Here, we consider the execution times of uninterrupted tasks.

The figure also has an entry for the WCET bound. The WCET bound is an upper, hopefully safe, bound on the execution time a WCET analysis tool can give. There are several reasons why it is usually not possible to give the exact WCET: (a) the complexity of *software* analysis, e.g., proving infeasible execution paths, and (b) the simplifications on *hardware* models required to make the analysis computationally practical. Regarding software, infeasible execution paths may significantly impact the WCET bound because the static analysis cannot prove that these paths may never be executed. Similarly, on the hardware side, dynamic features such as branch prediction and cache contents often need to be modeled conservatively to prevent an explosion of the analysis complexity. The difference between the WCET bound and the real WCET is called pessimism. This pessimism is usually not known because it is practically infeasible to know the real, ground-truth WCET. The general aim of WCET analysis tools is to provide a tight upper bound of the real WCET.

Let us focus on the first three processors and their distributions of the execution times. In the general purpose computing view, Processor A is the *best* processor, as it has the lowest ACET. Due to the complexity of processor A, we have a relatively high WCET bound and pessimism. Processor B is slower than processor A. It is slower in the ACET and also in the WCET. However, the WCET bound of processor B is lower than that of processor A. Therefore, even with a higher WCET than processor A, processor B is a better choice for a real-time system. A processor for real-time systems aims to optimize the WCET bound, even when ACET suffers. Therefore, processor C, now with a lower WCET and WCET bound than processors A and B is the preferred choice for a real-time system. For hard real-time systems, the likely increase in the ACET and BCET is acceptable, because the complete system needs to be designed to reduce the WCET. Note that a processor optimized for low WCET will be generally slower in the average

case than a processor optimized for low ACET. Those are two different design optimizations.

Note that those timing distributions are just for a single task, not as a measurement of a processor for all tasks. The variability of the software is included in that figure.

### Time-predictability definitions

Several groups have defined different metrics to talk about time-predictable architectures. We describe them and discuss them in chronological order of publication.

Thiele and Wilhelm (2004) introduced a foundational definition of time-predictability. A WCET analyzer is assumed to provide an upper bound on the WCET rather than the exact WCET. Furthermore, with complex hardware, it is often impossible to prove that the upper bound is accurate since the timing model used in the analysis is usually based on guesses from reverse engineering the undocumented timing behavior of the processor.

Their approach involves assessing predictability by evaluating the divergence between the actual WCET and the bounded WCET, commonly referred to as the *pessimism* of the analysis. They also incorporate the disparity between the lower bound and the best-case execution time (BCET) into their framework for predicting execution times. The main issue with this definition is that we usually do *not* know the exact WCET. Therefore, we cannot compute that predictability metric. Another issue with this definition is that it does not include the execution time. Therefore, assume two architectures, e.g., C and D from Figure 1, that we compare. Assume C has caches that result in a conservative WCET bound. Assume that D is a simple architecture without caches and a simple sequential processor implementation. Therefore, we can *better* analyze that architecture, and the WCET bound is close to the WCET. However, due to its simplicity, architecture D's WCET is higher than that of architecture C. The factor between the WCET Bound and the WCET would favor processor D, which is slower than processor C.

However, the question of which processor to choose might also depend on other factors, e.g., power consumption.

Schoeberl (2009) defines:

Under the assumption that only feasible execution paths are analyzed, a time-predictable processor's WCET bound is close to the real WCET.

However, how useful is this definition when we do not know the WCET? And is this the only useful property? Let us discuss processor D in Figure 1. For processor D, the bound is closest to the real WCET. However, it is also the processor with the highest bound and WCET. Is this the better processor for a real-time system? For practical reasons, probably not.

He proposes time-predictable solutions for common architectural artifacts in the rest of the paper. E.g., special caches (method, stack, and special cache), and TDMA-based access to external memory for chip multiprocessors. Pitter and Schoeberl (2010) use TDMA-based memory sharing of a multicore version of the JOP Java processor.

Kirner and Puschner (2011) introduce various definitions of time-predictability. The initial definition centers on the interval between the BCET and the WCET denoted as [BCET, WCET]. A narrower interval signifies enhanced predictability, with BCET equating to WCET indicating the highest predictability. More informally, they also define that the time-predictability of a *model* is the ability to calculate the execution time of a *concrete* system.

Specifically, they define predictability by multiplying *analyzability* with *stability*, including weighting factors. The stability is given by the quotient of  $\frac{BCET}{WCET}$  of the *timing model*. For WCET-predictability, the stability is ignored, and its factor is set to zero. The analyzability is now equal to the WCET-predictability, as the quotient of WCET and WCET bound. The analyzability of a model is the computational effort needed to calculate the relevant timing properties of the model. For example, a program with data-dependent control flow is less analyzable than one without. Likewise, a processor with memory caching will be less analyzable than one with scratchpad memories. No attempt is made to quantify the analyzability of a timing model. The stability of a timing model is the inherent variability of the model.

Grund et al. (2011) delve into the realm of predictability and introduce a framework for its definition. This template encompasses three fundamental elements: (1) the property subject to prediction, (2) the origin of uncertainty, and (3) a metric for assessing quality. Their work focuses on the property of time and defines time-predictability, expressed as the ratio between the minimum and maximum execution times. A system achieving a quotient of 1 is considered perfectly predictable. They argue that every system inherently possesses a predictability factor independent of WCET analysis. Therefore, they argue that this assessment will not include the WCET analysis. However, the challenge lies in the typical absence of knowledge regarding real WCET and BCET, rendering the “measurement” of time-predictability impractical.

### Program dependency

Time-predictability metrics usually depend on the actual program under consideration. For example, there is a large difference in the WCET bound dependent on where the data is allocated. If the program uses statically allocated data only, a WCET tool can compute the static addresses and model the data cache behavior. However, if the data is allocated on the heap, there is no static information on the data addresses available. Therefore, the cache analysis cannot model the cache behavior and needs to assume cache misses. This will lead to a large difference in the WCET bounds for almost the same program. It is often possible to write a more or less WCET analysis-friendly program, implementing the same logic.

Puschner and Burns (2002) present single-path programming as an extreme approach to minimize program-dependent execution time. To produce single-path code, the compiler takes any WCET-analyzable code and converts its control flow to only have a single execution path. If-conversion is used to eliminate data-dependent branching (Allen et al. 1983), while loops are forced to iterate a fixed number of times (the maximum possible.) Predication or conditional-move instructions ensure the program maintains the same behavior. Addition techniques efficiently remove execution-time variation from function calls and memory accesses (Maroun et al. 2023). While it can only be used for WCET-analyzable code, e.g., requiring all loops to have a known upper iteration bound, single-path code is a code-generation technique that is general enough for any real-time application. The limiting factors for single-path code are its hardware requirements, including predicated instructions or conditional-move support and fixed memory access latency.

Paired with a sufficiently time-predictable processor, like Patmos, single-path code can achieve constant execution times. Simply measuring a program's runtime will thus always produce

the same execution time regardless of the input data. The WCET and BCET are thus the same. However, that programming approach has so far been challenged by lower performance than what can be achieved using traditional analysis. More optimization research is therefore required to establish the true performance potential of the single-path approach. However, single-path code exemplifies how two toolchains can differ in predictability only based on the code generation (i.e., the compiler) and analyzer used: analyzed traditional code versus measured single-path code.

### Time-predictable processor projects

Several research and commercial projects aim to build a time-predictable processor for real-time systems. Here we list just a few examples; an exhaustive list would be a survey paper on its own.

Dinechin et al. (2014) designed the Kalray manycore processor for time-critical computation. The processor is organized into 16 clusters of 16 cores. The clusters are connected by a NoC with rate control at the sender to support time-predictable message passing with rate control in the sender. Each core within a cluster and the NoC are connected to an SPM with 16 independent memory banks. By carefully selecting the allocation of data and access to the memory banks, the notion of ownership can be implemented in software. The individual processor cores have been designed according to the proposal of Wilhelm et al. (2009).

Zimmer et al. (2014) and Zimmer (2015) extend the PRET architecture by Lickly et al. (2008), calling it FlexPRET to support mixed-criticality systems with fine-grain multithreading. FlexPRET supports two different thread types, hard and soft real-time threads, in the hardware. Both thread types have fixed slots assigned in the fine-grained thread scheduling. However, slots not used by a thread (e.g., because of stalling or because a thread has finished its current release) can be used by the soft real-time threads. FlexPRET implements the RISC-V instruction set, as defined by Waterman et al. (2011).

May (2012) present XCore, a multicore architecture developed by XMOS in the vision of PRET architectures. XCore supports fine-grained multithreading similar to the FlexPRET architecture. XMOS supports XCore with a WCET analysis tool.

Schoeberl et al. (2018) present Patmos, a processor specially designed to be time-predictable. All features, e.g., caches, have been chosen to be WCET analysis friendly. In this paper, we use Patmos as an example for the calculation of the worst-case performance.

### Time-predictable processor components

If it is impossible to determine a quantitative assessment of time-predictability for whole processors, it may be useful to look at processor components. For example, caches can be designed for a tighter WCET bound than caches optimized for average performance. One way of doing so is by choosing the cache line replacement strategy. Reineke et al. (2007) investigate the predictability of various replacement policies. They classify predictability as how fast a policy regains information about the cache's contents after an information-losing event, such as accessing an unknown address. They establish that the least recently used LRU replacement policy is the most predictable. This is because every access at least establishes information about the cache line accessed (it becomes the most recently used).

The predictability definition used by Reineke et al. (2007) hinges on the existence of an analyzer that uses knowledge about the replacement policy to track what can and cannot be in the

cache at a given program point. This means that the predictability of a cache component is dependent on the analyzer, without which the predictability classification has no effect.

Reineke et al. (2007) also does not account for the influence of specific programs on the cache predictability. A program performing only dynamic memory accesses will prohibit an analyzer from ever establishing any knowledge about cache contents, regardless of replacement policy. Alternatively, a program that does not use the cache or has a small data memory footprint where all data fit into one way of the cache is not affected by the replacement policy. Another example is a WCET analysis tool, such as platin, which we use in the evaluation. It does not contain a data cache analysis and, therefore, needs to assume a miss on all access to data that is not allocated on the stack. Therefore, the theoretical knowledge that a specific policy is “usually” most beneficial is ineffective in such cases.

In less extreme program examples, there would be a ratio between “predictable” and “unpredictable” cached memory accesses, which should be incorporated into the evaluation of a system, as the ratio would affect how useful the predictability of the cache replacement policy is to the overall system.

Again, we cannot quantify the time-predictability in absolute terms for individual components. We can only state that replacement policy A gives a tighter WCET bound than policy B with a given program and a given analyzer.

Ferdinand et al. (2001) have shown that the perfect analyzer is impractical, as practical analysis speed is best achieved with modular analysis. However, for practical and modern systems, an integrated analysis “is the only one that can give useful results” (Heckmann et al. 2003). Analyzers have to therefore trade accuracy for practicality, never quite achieving the perfect result.

Axer et al. (2014) also investigate the predictability of specific components of a system, from architectural components like the pipeline or cache to the compiler and system design. They provide methods of determining the predictability (a value between 0 and 1) of system components, like the cache, and the system as a whole. They recognize that compositionality would be a useful property of predictability, i.e., by composing the predictability metrics of the system components, the predictability of the whole system can be determined. However, they identify no way of achieving compositionality and recognize that the interactions between components might cancel out the predictability of some of them. For example, an unpredictable pipeline would negate the predictability of the cache. Lastly, the predictability measure used is assumed in the presence of “a system-specific optimal analysis” (Axer et al. 2014). Without an analyzer with such an optimal analysis implementation for each system component, the predictability measures cannot be used. Therefore, even when ignoring the lack of composition of component predictability, without a perfect analyzer, everything breaks down.

### Worst-case performance

As we have seen, time-predictability is hard to define to be used in practice. For many real-time systems, the WCET bounds of tasks are more important. We propose to quantify this as the worst-case performance (WCP) of a processor and its tooling. For average-case performance, industry and academia have converged on certain benchmark suites, such as the SPEC benchmark. The same can be done for the WCP. We propose using the TACLe benchmark suite (Falk et al. 2016) to quantify WCP.



Similar to SPEC benchmarks, WCP benchmarks constitute an end-to-end *quantification* that includes the program, the compiler, the processor, and the memory (hierarchy). As in SPEC benchmarking, we use the geometric mean for the WCP benchmarks as it provides a more balanced comparison. However, while the SPEC benchmarks are executed on actual hardware or emulations of hardware, the WCP benchmarks are being performed by a WCET analysis tool, whose bounds are used for the quantification. A WCP benchmark score for a particular processor will depend on the WCET analysis tool and the processor's available timing model. This will incentivize designers of time-predictable processors to provide accurate timing models to achieve the highest possible score on the WCP benchmarks. A separate verification flow is needed to verify that the timing model is correct (meaning that the WCET bound is always greater than the actual WCET).

### TACLeBench

The TACLeBench suit is self-hosted, which means it does not need an operating system and not even a standard library. The TACLeBench suit contains several different categories of benchmarks:

- Kernel benchmarks
- Sequential benchmarks
- Application benchmarks
- Tests
- Parallel benchmarks

We argue that the set of benchmarks for the WCP should be restricted to reasonable examples. Kernels are small examples to experiment with, but kernels shall be excluded as in classic benchmarking for performance. We also argue that the three test benchmarks should be excluded. Furthermore, as we have no good proposal for using the parallel benchmarks without an operating system, we exclude them as well.

This results in using the sequential benchmarks and the two application benchmarks. The sequential benchmarks are typical embedded library functions such as encoders and decoders. The application benchmarks contain two applications: *lift* is a real-world application ported from Java to C. *powerwindow* is an application written as an example of a real-time system (not an application in real use).

Similar to the SPEC benchmark, we shall provide a geometric mean of the result. The geometric mean is frequently used in benchmarks, especially when workloads have substantial differences in runtime, often spanning orders of magnitude. In our case, we report execution time, which means lower numbers are better.

The benchmarks are self-contained. That means all input data is part of the C code. Furthermore, the benchmarks are self-checking: at the end of the benchmark, the computed result is compared with the expected result. In that case, the test passes, and the benchmark returns with 0. To avoid the compiler optimizing the whole code away due to the known input data, those fields are marked *volatile*.

However, with the input data fixed, there is no data-dependent control variation, and therefore also no data-dependent timing variation. Therefore, we can also get some insight into architectures when simply measuring the execution time on a platform. Zimmer et al. (2014) present FlexPRET, an architecture without hidden state. When using FlexPRET with single-threaded execution, this measured execution time can give some insight into

**Table 1.** WCET bounds and single-path execution times in microseconds for each of the chosen benchmarks and their geometric mean

Benchmark	WCET Bound ( $\mu$ s)	Single-Path ( $\mu$ s)
lift	81,325	59,153
powerwindow	307,490	268,190
adpcm_dec	170	207
adpcm_enc	241	291
audiobeam	3,018,904	–
cjpeg_transupp	1,569,274	2,529,957
cjpeg_wrbmp	3497	3604
Dijkstra	406,010,800	–
g723_enc	46,952	–
gsm_dec	131,679	–
h264_dec	618	944
huff_dec	28,666	–
mpeg2	710,057,703	–
ndes	3249	3420
petrinet	458	–
statemate	1462	1528
geometric mean	187,388	

the worst-case performance. However, the input data might not trigger the worst-case path in the program.

### Example WCP

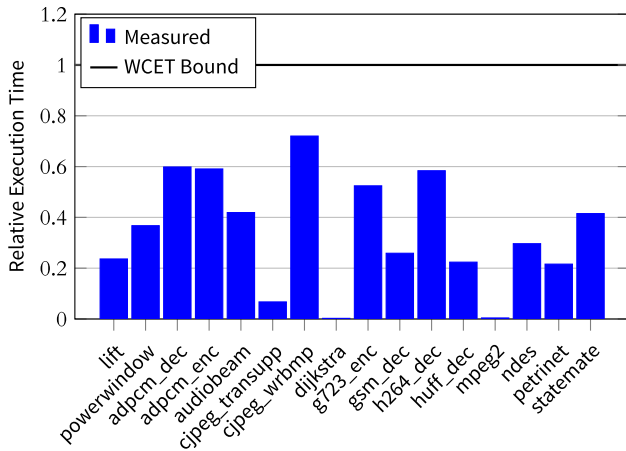
For example, we perform WCP benchmarking for Patmos. Hepp et al. (2015), Huber et al. (2011), and Maroun et al. (2024) provide the WCET analysis tool *platin*, which is part of the T-CREST platform (Schoeberl et al. 2015). It is integrated with the compiler (LLVM) to generate flow facts (e.g., loop bounds). *Platin* has a timing model for Patmos, including memory timings for different field programmable gate array (FPGA) boards.

Sieh et al. (2017) have extended *platin* to support the ARM and RISC-V instruction sets. Therefore, we also envision exploring the usage of *platin* with FlexPRET.

We target Patmos in the version implemented in the low-cost FPGA board DE2-115 which contains an Intel Cyclone-IV FPGA. We configure Patmos with 4 KB method cache, 2 KB stack cache, and 4 KB data cache. Note that our WCET analysis tool does not include a data cache analysis; therefore, each data access is considered a miss with a penalty of 21 clock cycles. However, *platin* includes a method cache and stack cache analysis.

In Table 1, we show in the second column the WCET bounds for each of the chosen benchmarks produced by *platin*. We scale the raw cycle counts *platin* produced to the default 80 MHz frequency of our setup. We only include the programs compiled, run, and analyzed correctly. For two of the sequential programs, loop bounds were invalid and so could not be analyzed. For seven programs *platin* failed to produce WCET bounds. That leaves us with 16 programs. At the bottom, we have added the geometric mean, which would be the WCP of the combination of our end-to-end toolchain and Patmos processor.

Another option is to use single-path transformation to achieve constant execution time. Table 1's third column shows the



**Figure 2.** Execution time of running each program with traditional code compared to the WCET bound of the traditional code.

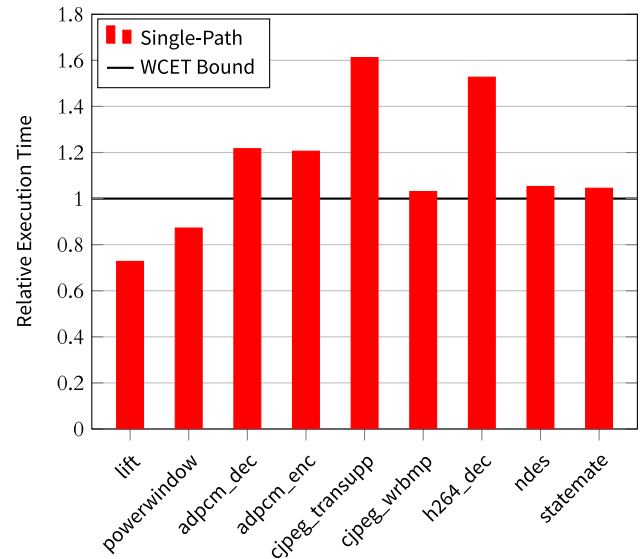
execution time of each program using single-path code instead of WCET analysis. Because the compiler is immature and has bugs, it failed to compile some of the benchmarks to correct single-path code, so only those compiled and executed correctly are used. Since the execution time of single-path code is both the best and worst-case, it can be directly compared to bounds from tools like platin.

Looking at the actual execution time numbers, we can see that single-path code does not include values for the most long-running programs. Therefore, we do not present the geometric mean for the single-path measurements. Single-path code is longer-running than traditional, analyzed code for all but the two application programs. Therefore, for any fair comparison using WCP, every program chosen as part of the benchmark must be represented in all comparison setups.

The results highlight a problem: We can see that analyzers and compilers can have bugs and fail to produce results. However, do we even know if the results we do have are correct? A WCP would be irrelevant if it were wrong, meaning the first task of any benchmarking is to provide some confidence in the results' correctness.

One way to gain confidence in WCET analysis results is to run the program repeatedly on different input data and measure the execution time. With sufficient data variation and no instances of execution times exceeding the WCET bound, we can have confidence that the bound is valid. An analogous thing could be done for single-path code, where we ensure the execution is identical for all runs. We propose that a future version of TACLe benchmarks should facilitate such confidence-building measures as crucial for trusting the eventual WCP.

The TACLe benchmark does not provide any such facility. Every program comes with one set of input data and no more. Therefore, it cannot be used to validate the correctness of the bound produced by an analyzer or the execution time of single-path code. We imagine the possibility of building a benchmark that provides diverse input data sets for each program or an automated way of generating data on the fly. For the latter, care must be taken that all the data generated is valid and fits the constraints expected by the program. Otherwise, invalid data might cause errors, e.g., by making a loop iterate an unreasonable number of times.



**Figure 3.** Execution time of running each program with single-path code compared to the WCET bound of the traditional code.

One observation from comparing WCET bounds with single-path generated code is that better time predictability comes at a cost: a higher execution time. If (average-case) performance-enhancing features, such as out-of-order execution and branch predictors are avoided, we will get lower (average-case) performance. One future challenge is developing performance-enhancing features that help lower the worst-case performance.

In Figure 2, we take the WCET bound produced by platin for each program and compare it to the actual execution time of simply running the program. Notice that for the Dijkstra and mpeg2, the values are incredibly small at close to 0.003 and 0.004, respectively.

We have investigated the two extreme cases Dijkstra and mpeg2. We explored different input data looking to maximize the measured execution time, but it did not increase by more than a factor of 2, which is not a meaningful change compared to their respective WCET bound. The code of Dijkstra is not WCET-friendly. There are many places where the analysis tool has to assume the worst loop bound when, most often, that loop is taken only a few times. For example, the queue of waiting nodes can be up to 1000 nodes long, but most often, it is not. It makes sense that platin produces a bound that is very pessimistic.

The code of mpeg2 is similarly unfriendly to WCET analysis. It consists of many branches where one path does almost nothing while the other performs many computations. Additionally, these branches are nested. To find a safe WCET bound, the analyzer must assume the paths taken are the ones where all the complex branch paths are taken. However, it is likely that such a path will be very rare or impossible to take at runtime.

In summary, these two examples are probably not written to be used in real-time systems where the WCET bound is of primary importance. General guides should be followed when writing programs for real-time systems (Harmon et al. 2008). Therefore, updating the TACLe benchmark suit with programs written for real-time systems would be a good idea. Furthermore, it would be interesting to have a mechanism to vary the input values for some research questions.

We also compare, in Figure 3, the WCET bound to the execution time of the program when using single-path code. We also only include the programs that compile and run correctly. Recall that single-path code has constant execution time, so it can be directly compared to the WCET bound, unlike execution times, which do not necessarily represent the WCET. One would assume that the execution time of single-path code is usually higher than the WCET bound produced by WCET analysis. The two application benchmarks are more efficient in single-path code. This shows options for improvement in platin, and concrete the cache analysis.

## Discussion

As we have seen with the large WCET overestimation on Dijkstra, the style of writing code for real-time systems is also important. The entire application software stack is crucial, particularly because simplified software structures can help mitigate some of the analyzability limitations inherent in a specific hardware platform. Therefore, it has to be stressed that the style of writing software for real-time systems, e.g., use static allocated data, recycle buffers, and being aware of loops with reasonable bounds is important for the whole system performance.

To factor the programming style into the benchmark results, we would need to change the benchmarks from being provided in source to providing specifications and tests of typical real-time problems. In that case, writing the benchmarks would be part of the benchmarking. We consider that (unusual) form of benchmarks as an idea for future work.

## Conclusion

To build real-time systems, we aim to use time-predictable architectures where we can provide proof of the upper bounds of worst-case execution times of tasks. We have explored several approaches to defining the *meaning* of time-predictability, but concluded that it is probably a property that cannot be easily defined and might not be useful in practice. Therefore, we defined worst-case performance as a metric to compare different systems, including a processor, a compiler, and tools, such as WCET analysis or single-path code generation. We propose to use the TACLeBench benchmark suit with a worst-case analysis tool or single-path code generation to provide a worst-case performance metric.

**Data availability statement.** The data that support the findings of this study are openly available in the T-CREST GitHub repository at <https://github.com/t-crest> and the TACLe benchmarks at <https://github.com/tacle/tacle-bench>.

**Acknowledgment.** We acknowledge that we used the free version of ChatGPT 3 to help us improve our English writing (as a better grammar checker). We did not use ChatGPT to produce original text.

**Author contribution statement.** All authors have contributed to the conception and writing of the article. Emad Jacob Maroun has performed the WCET analysis and single-path code measurements for the worst-case performance section.

**Funding statement.** This research was supported by grants from Huawei Sweden under project number 112707.

**Competing interests.** None.

**Ethics.** Ethical approval and consent are not relevant to this article type.

**Connections references.** Lee EA, Woodcock J (2023). Time-sensitive software. *Research Directions: Cyber-Physical Systems*. 1: e1. <https://doi.org/10.1017/cbp.2023.1>

## References

- Allen JR, Kennedy K, Porterfield C and Warren J (1983) Conversion of control dependence to data dependence. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp. 177–189.
- Axer P, Ernst R, Falk H, Girault A, Grund D, Guan N, Jonsson B, Marwedel P, Reineke J, Rochange C, *et al.* (2014) Building timing predictable embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)* 13(4), 1–37.
- Berg C, Engblom J and Wilhelm R (2004) Requirements for and design of a processor with predictable timing. In Thiele L and Wilhelm R (eds.), *Perspectives Workshop: Design of Systems with Predictable Behaviour*. Dagstuhl Seminar Proceedings 03471. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- Bucek J, Lange K-D and Kistowski Jv (2018) SPEC CPU2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE '18*. Berlin, Germany: Association for Computing Machinery, pp. 41–42. <https://doi.org/10.1145/3185768.3185771>.
- Dinechin BD, van Amstel D, Poulhies M and Lager G (2014) Time-critical computing on a single-chip massively parallel processor. In *Conference on Design, Automation and Test in Europe, DATE '14*. Dresden, Germany: European Design/Automation Association, pp. 97:1–97:6.
- Edwards SA, Kim S, Lee EA, Liu I, Patel HD and Schoeberl M (2009) A disruptive computer design idea: architectures with repeatable timing. In *Proceedings of IEEE International Conference on Computer Design (ICCD 2009)*. Lake Tahoe, CA: IEEE.
- Edwards SA and Lee EA (2007) The case for the precision timed (PRET) machine. In *Design Automation Conference (DAC)*.
- Falk H, Altmeyer S, Hellinckx P, Lisper B, Puffitsch W, Rochange C, Schoeberl M, Sørensen RB, Wägemann P and Wegener S (2016) TACLeBench: a benchmark collection to support worst-case execution time research. In Schoeberl M (ed.), *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, vol. 55. OpenAccess Series in Informatics (OASICS). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 2:1–2:10. <https://doi.org/10.4230/OASICS.WCET.2016.2>.
- Ferdinand C, Heckmann R, Langenbach M, Martin F, Schmidt M, Theiling H, Thesing S and Wilhelm R (2001) Reliable and precise WCET determination for a real-life processor. In Henzinger TA and Kirsch CM (eds.), *Emsoft*, vol. 2211. Lecture Notes in Computer Science. Springer, pp. 469–485.
- Grund D, Reineke J and Wilhelm R (2011) A template for predictability definitions with supporting evidence. In Lucas P, Thiele L, Triquet B, Ungerer T and Wilhelm R (eds.), *Bringing theory to practice: predictability and performance in embedded systems*, vol. 18. OpenAccess Series in Informatics (OASICS). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 22–31. <https://doi.org/10.4230/OASICS.PPES.2011.22>.
- Harmon T, Schoeberl M, Kirner R and Klefstad R (2008) Toward libraries for real-time Java. In *Proceedings of the 11th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (Isorc 2008)*, pp. 458–462. Orlando, Florida, USA: IEEE Computer Society. <https://doi.org/10.1109/ISORC.2008.73>.
- Heckmann R, Langenbach M, Thesing S and Wilhelm R (2003) The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE* 91(7), 1038–1054.
- Hepp S, Huber B, Knoop J, Prokesch D and Puschner PP (2015) The platin tool kit - The T-CREST approach for compiler and WCET integration. In *Proceedings 18th kolloquium programmiersprachen und grundlagen der programmierung, KPS 2015, Pörtlach, Austria, October 5–7, 2015*.

- Huber B, Puffitsch W and Puschner P** (2011) Towards an open timing analysis platform. In *Proceedings of the 11th International Workshop on Worst-Case Execution Time (WCET) Analysis*, pp. 6–15. Talk: 11th International Workshop on Worst-Case Execution Time Analysis, Porto; 2011-07-05.
- Kirner R and Puschner P** (2011) Time-predictable computing. In Min S, Pettit R, Puschner P and Ungerer T (eds.), *Software Technologies for Embedded and Ubiquitous Systems*, vol. 6399. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, pp. 23–34.
- Lickly B, Liu I, Kim S, Patel HD, Edwards SA and Lee EA** (2008) Predictable programming on a precision timed architecture. In Altman ER (ed.), *Proceedings of the International Conference on compilers, Architecture, and Synthesis for Embedded Systems (CASES 2008)*, pp. 137–146. Atlanta, GA, USA: ACM.
- Liu I** (2012) Precision timed machines. PhD diss., EECS Department, University of California, Berkeley.
- Maroun EJ, Schoeberl M and Puschner P** (2023) Compiler directed constant execution time on flat memory systems. In *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE.
- Maroun EJ, Dengler E, Dietrich S, Hepp C, Herzog B, Huber H, Knoop J, Prokesch D, Puschner P, et al.** (2024) The platin multi-target worst-case analysis tool. In *22th International Workshop on Worst-Case Execution Time Analysis (WCET 2024)*.
- May D** (2012) The XMOS architecture and XS1 chips. *IEEE Micro* 32(6), 28–37. <https://doi.org/10.1109/MM.2012.87>.
- Pitter C and Schoeberl M** (2010) A real-time Java chip-multiprocessor. *ACM Transactions on Embedded Computing Systems (New York, NY, USA)* 10(1), 1–34. <https://doi.org/10.1145/1814539.1814548>.
- Puschner P and Burns A** (2002) Writing temporally predictable code. In *Proceedings of the Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*. Washington, DC, USA: IEEE Computer Society, 85–94. <https://doi.org/10.1109/WORDS.2002.1000040>.
- Reineke J, Grund D, Berg C and Wilhelm R** (2007) Timing predictability of cache replacement policies. *Journal of Real-Time Systems* 37(2), 99–122.
- Schoeberl M** (2009) Time-predictable computer architecture. *EURASIP Journal on Embedded Systems* 2009, Article ID 758480, 17 pages. <https://doi.org/10.1155/2009/758480>.
- Schoeberl M** (2012) Is time predictability quantifiable? In *International Conference on Embedded Computer Systems (SAMOS 2012)*. Samos, Greece: IEEE.
- Schoeberl M, Abbaspour S, Akeson B, Audsley N, Capasso R, Garside J, Goossens K, Goossens S, Hansen S, Heckmann R, Hepp S, Huber B, Jordan A, Kasapaki E, Knoop J, Li Y, Prokesch D, Puffitsch W, Puschner P, Rocha A, Silva C, Sparso J and Tocchi A** (2015) T-CREST: Timepredictable multi-core architecture for embedded systems. *Journal of Systems Architecture* 61(9), 449–471. <https://doi.org/10.1016/j.sysarc.2015.04.002>.
- Schoeberl M, Puffitsch W, Hepp S, Huber B and Prokesch D** (2018) Patmos: A time-predictable microprocessor. *Real-Time Systems* 54(2), 389–423. <https://doi.org/10.1007/s11241-018-9300-4>.
- Sieh V, Burlacu R, Honig T, Janker H, Raffeck P, Wagemann P and Schroder-Preikschat W** (2017) An end-to-end toolchain: from automated cost modeling to static wcet and wcec analysis. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*. Los Alamitos, CA, USA: IEEE Computer Society, pp. 158–167. <https://doi.org/10.1109/ISORC.2017.10>.
- Thiele L and Wilhelm R** (2004) Design for timing predictability. *Real-Time Systems* 28(2–3), 157–177.
- Waterman A, Lee Y, Patterson DA and Asanovic K** (2011) *The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA*. Technical report UCB/EECS-2011-62. EECS Department, University of California, Berkeley.
- Wilhelm R, Grund D, Reineke J, Schlickling M, Pister M and Ferdinand C** (2009) Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on CAD of Integrated Circuits and Systems* 28(7), 966–978.
- Zimmer M** (2015) Predictable processors for mixed-criticality systems and precision-timed I/O. PhD diss., EECS Department, University of California, Berkeley.
- Zimmer M, Broman D, Shaver C and Lee EA** (2014) FlexPRET: A processor platform for mixed-criticality systems. In *Proceedings of the 20th IEEE Real-Time and Embedded Technology and Application Symposium (RTAS)*. Berlin, Germany.