The Platin Multi-Target Worst-Case Analysis Tool

Emad Jacob Maroun
Technical University of Denmark, Lyngby, Denmark

Eva Dengler
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Christian Dietrich
Technische Universität Braunschweig, Germany

Stefan Hepp
Technische Universität Wien, Austria

Henriette Herzog
Ruhr-Universität Bochum, Germany

Benedikt Huber
Technische Universität Wien, Austria

Jens Knoop
Technische Universität Wien, Austria

Daniel Wiltsche-Prokesch
Hitachi Rail, Wien, Austria

Peter Puschner
Technische Universität Wien, Austria

Phillip Raffeck
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Martin Schoeberl
Technical University of Denmark, Lyngby, Denmark

Simon Schuster
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Peter Wägemann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

With the increasing number of applications that require reliable runtime guarantees, the relevance of static worst-case analysis tools that can provide such guarantees increases. These analysis tools determine resource-consumption bounds of application tasks, with a model of the underlying hardware, to meet given resource budgets during runtime, such as deadlines of real-time tasks.

This paper presents enhancements to the PLATIN worst-case analysis tool developed since its original release more than ten years ago. These novelties comprise PLATIN's support for new architectures (i.e., ARMv6-M, RISC-V, and AVR) in addition to the previous backends for Patmos and ARMv7-M. Further, PLATIN now features system-wide analysis methods and annotation support to express system-level constraints. Besides an overview of these enhancements, we evaluate PLATIN's accuracy for the two supported architecture implementations, Patmos and RISC-V.

2012 ACM Subject Classification Software and its engineering → Automated static analysis; Software and its engineering → Compilers; Computer systems organization → Real-time systems

Keywords and phrases worst-case resource consumption, WCET, static analysis tool

Digital Object Identifier 10.4230/OASIcs.WCET.2024.2

Supplementary Material *Software (Platin Source Code)*: `https://github.com/t-crest/platin`

Funding The contributions of Emad Jacob Maroun have been funded by Huawei Technologies Sweden AB under agreement number TC20220819029. Authors from the Friedrich-Alexander-Universität Erlangen-Nürnberg have been funded in parts by the German Research Foundation (DFG) under the project number 502947440 (Watwa).

1 Introduction

Safety-Critical Applications & Worst-Case Analysis. The relevance of solving the worst-case execution time (WCET) problem [57] is higher than ever when considering the increasing number and the complexity of today's safety-critical application requirements running on modern processors. From small medical (implantable) devices over automotive and avionics applications to large industrial-control scenarios, systems that require resource budget guarantees for safe execution need tooling support to determine resource bounds analytically. The basic principle of worst-case analysis is to combine a representation of the system's program paths with a cost model of the underlying hardware. With this knowledge, the worst-case analysis generates a mathematically sound problem formulation, such as an integer linear program (ILP). When given to a mathematical solving tool, the solution of this formulation yields resource-consumption bounds, which are used for offline budgeting of runtime resources.

Resource Other Than Time: Worst-Case Energy Consumption. This method of combining a program-path model with a cost model was introduced in the 1990s [35, 39] and referred to as the *Implicit Path Enumeration Technique (IPET)*. The original purpose of the IPET targeted the timeliness of real-time systems. However, Jayaseelan et al. later demonstrated the applicability of this approach for determining worst-case energy consumption (WCEC) bounds [28]. In the same way, WCET bounds are crucial for meeting deadlines in real-time systems; WCEC estimates are helpful in energy-constrained settings to guarantee the safe completion of tasks under energy budgets. This paper addresses the two resources: time and energy within the PLATIN tool with WCET/WCEC analyses.

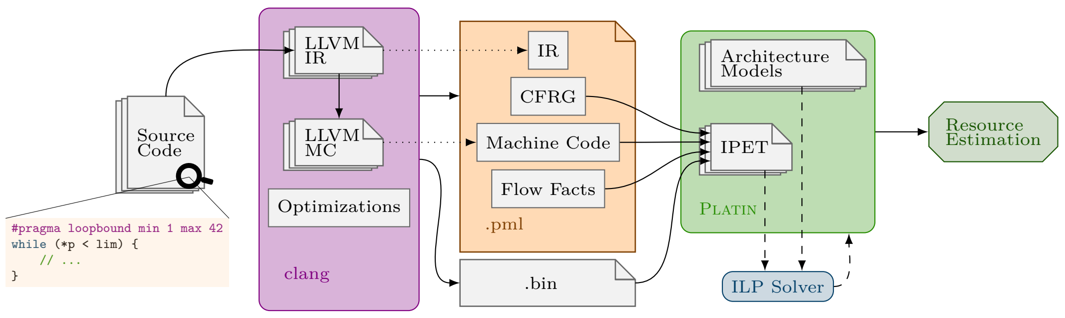
Necessity for Open Architectures & Open Tooling Infrastructures. The PLATIN tool was originally introduced more than ten years ago [23, 40] as a portable LLVM annotation and timing toolkit. PLATIN's development started with the T-CREST project [44], targeting time-predictable multi-core architectures. With the entire technology available as open source, we argue that the research community requires both open processor architectures and the respective worst-case tooling support to advance state-of-the-art without unnecessary barriers (i.e., licensing, closed-source infrastructures). In line with this rationale, all our improvements and extensions to the PLATIN tool have been published as open-source over the last few years.

Contributions. The core contribution of this work is an overview of these novelties compared to PLATIN's initial release [23]. The novelties include both pillars of worst-case analysis: (1) Regarding the hardware-dependent cost modeling, PLATIN now supports four new architectures (ARMv6-M, ARMv7-M, RISC-V, AVR). (2) Given the hardware-agnostic program-path analysis, we give insight into the introduced support for system-wide resource-consumption analysis and PLATIN's annotation infrastructure. Besides the overview of existing work, we evaluate PLATIN for the Patmos and RISC-V (RV32IMC ISA) architectures.

Paper Organization. The paper is structured as follows: Section 2 gives a general overview of the tooling infrastructure of and around PLATIN. The existing and newly introduced architectures are part of Section 3. Section 4 describes extensions of PLATIN for whole system time and energy analysis. Section 5 presents evaluation results. Section 6 discusses related work. Section 7 concludes the paper.

2 Overview of the PLATIN Analysis Tool

The PLATIN ecosystem displayed in Figure 1 combines compilation and WCET analysis to make use of high-level information that the compiler already has [40]. The source code, potentially enriched with user-annotated control-flow information (so-called *flow facts*) such



```
#pragma loopbound min 1 max 42
while (*p < lim) {
    // ...
}
```

Figure 1 Overview of Platin's ecosystem for compiler-analysis integration providing analysis-aware compilation to improve accuracy with automatically collected program meta-information.

as loop bounds, is compiled by an extended version of the `clang` compiler into the final binary and a meta-information file (`.pml`). This meta-information contains the program control flow and flow facts in the YAML[1]-based Program Metainfo Language (PML) format specific to Platin. Specifically, it contains the program's control-flow graph (CFG) in an intermediate representation (IR) and on machine-code (MC) level. A control-flow–relation graph (CFRG) [25] matches program paths between the two representations even across different optimization-induced control-flow transformations. The CFRG is thus a useful tool to lower IR-level flow facts (both annotated and compiler-inferred) to the machine-code level, where the actual timing analysis is performed. Platin uses the CFGs and the lowered flow facts to derive an IPET formulation and, finally, transforms this formulation together with a target-specific cost model from Platin's architecture models into an integer linear program (ILP). An external ILP solver (e.g., `lp_solve`, `gurobi`) then yields the resource bounds.

At the heart of Platin are the architecture models, which provide the translation from control-flow information to platform-specific resource demands. The core component for each of Platin's architecture models is a cost model of the machine instructions, informing Platin in how many processor cycles each instruction is executed in the worst case. Modeling of the microarchitecture, such as processor pipelining or caches, refines the model, allowing for more accurate bounds than pessimistic assumptions about cache misses and pipeline stalls.

In the following, we give further insights into the analysis-aware compilation process (Section 2.1) and other tools of Platin besides the analysis (Section 2.2).

2.1 Analysis-Aware Compilation with Clang

For Platin to perform its analyses, it needs the control-flow and flow-fact information provided in the PML format. Platin uses this data-serialization format to store and retrieve relevant program information for the worst-case analysis. Our fork of the LLVM compiler framework [32] includes support to automatically create the accompanying PML files for each compilation unit with a mixture of user-annotated and compiler-generated knowledge. The LLVM/clang infrastructure's code base is rapidly changing, which leads to the challenge of keeping our analysis infrastructure up to date with new LLVM/clang releases. To make

[1] YAML data-serialization language: `https://yaml.org`

Table 1 Overview of PLATIN's supported architectures with respective processor implementations.

Architecture	Processor Implementation
Patmos	Chisel-based implementation with FPGA synthesis (Altera DE2-115)
ARMv6-M	NXP FRDM-KL46Z with ARM Cortex M0+ [2, 3, 20]
ARMv7-M	XMC4500 with ARM Cortex M4 [4, 5, 27]
RISC-V	ESP32-C3 supporting RV32IMC extensions [15, 43]
AVR	ATmega1284p [6, 7]

forward compatibility and version updating as easy as possible, we strive not to change the core LLVM code and only add code specific to our use cases. That way, we can benefit from improvements in the LLVM infrastructure (e.g., novel analysis passes) with minor changes (e.g., adapting the PML export logic).

The compiler first takes the C source code and compiles it into LLVM intermediate representation (LLVM-IR). Any flow fact information, including loop bounds provided as pragmas, is embedded in the LLVM-IR to maintain it through the compilation pipeline [25]. Besides the program code, required standard libraries for the target can be linked on the LLVM-IR level with `llvm-link`. This enables a whole program view for the remaining steps, including optimization and PML export.

For targets that cannot be linked with `clang` or where libraries are not available as source code, the compiler produces object files, which can then be linked (without further optimizations) with an external linker. Library functions are only available at link time; however, they are challenging if they are part of any program path beginning from the analysis entry point. A timing bound can be derived solely from the machine code or must be known from external sources.

The backend exports the control flow and flow-fact information at the last stage of the compilation, where the machine instructions and their final order have been determined. The PML format and the compiler code managing its export are architecture-independent, giving seamless support for all current and future architectures.

2.2 PLATIN's Supporting Tools

Besides worst-case analyses, the PLATIN ecosystem provides several accompanying tools that support the analysis. Visualization of the CFRG allows debugging in cases where the one-on-one mapping is violated. Likewise, ILP visualization makes understanding the analysis results possible, while an interactive version enables live analysis in large projects.

Integration with external analysis tools (e.g., aiT) and transformation tools from and to the PML format allow PLATIN to profit from existing analyses. A configuration tool inspired by `pkg-config` helps to invoke tools with the correct options (e.g., target-specific flags, analysis entry) to guarantee interoperability with PLATIN.

3 PLATIN's Support for Multiple Architectures

The original version of PLATIN had full support for the Patmos processor and initial support for the ARM architecture, expressing the hope that using LLVM as a basis would allow for quick development of further backends [23]. This hope proved warranted, as PLATIN now supports multiple architectures. Table 1 gives an overview of the available architecture backends and corresponding processor models, further described in the following.

Patmos Architecture. PLATIN was initially developed for the Patmos architecture as part of the T-CREST project [44]. It provides full support for the architecture, which is also the default target [45]. As Patmos was developed explicitly for real-time systems, it has a unique cache structure: It uses a method instead of an instruction cache [11], which loads complete methods or subsets of them at predefined points in the program. It also has a dedicated stack cache for caching stack-local data. PLATIN supports modeling both of these caches [26, 29]. Patmos also has a traditional data cache. As PLATIN does not have native data-cache modeling, it assumes all data-cache accesses miss. However, when integrating with the AbsInt aiT analysis tool, AbsInt's data-cache modeling can be leveraged for improved WCET bounds.

ARM Architectures. PLATIN currently has support for the ARMv6-M and ARMv7-M versions of the ARM architecture. For the ARMv6-M backend, namely for the NXP FRDM-KL46Z with a Cortex M0+ processor, we demonstrated the feasibility of automatic derivation of the cycle costs of the timing models [50, 51]. The Cortex M0+ has a comparatively simple microarchitecture, which is not modeled explicitly but is part of the derived model. The ARMv7-M backend, which is for the XMC4500 with a Cortex M4 processor, features integrated modeling of the processor pipeline and the instruction cache [41] building upon the concept of microarchitecture execution graphs [52].

RISC-V Architecture. Additionally, we introduced support for the open-source hardware standard RISC-V [43] as an additional backend [12]. The supported ESP32-C3 [15] system-on-chip, which uses the RV32IMC instruction set, features a 4-stage pipeline and zero-wait-state memory for both instruction and data access. Due to the lack of documentation on the timing behavior, the timing model is derived from measurements, including the effects of pipelining.

AVR Architecture. We further extended PLATIN to support the AVR architecture, often utilized for embedded systems and popular Arduino projects. AVR microcontrollers typically have a relatively simple microarchitecture that allows straight-forward hardware models and their integration into PLATIN, in our case for the ATmega1284p [6]. Almost all instructions are executed with constant timing, documented in the AVR Instruction Set manual [7]. As the ATmega1284p does not have integrated caches, PLATIN's AVR backend allows for accurate WCET-bound predictions. To underpin this statement based on the exemplary benchmark `count_negative` from the TACLeBench suite [17]: This benchmark avoids overestimations from the program-path analysis and, consequently, helps to reveal pessimism originating from the architecture modeling. PLATIN's AVR backend reports 24009 cycles while the (straight-line code) measurement counts 22560 cycles: These results indicate minor analysis pessimism with the overestimation by 6 % and highlight PLATIN's applicability for the predictable AVR architecture.

4 PLATIN's Path-Analysis & Annotation Extensions

Besides PLATIN's support for several architectures, several works extended the analysis toolkit to support whole-system analyses (see Section 4.1 and 4.2) and express semantic annotations across the system stack (see Section 4.3).

4.1 SysWCET: Whole-System Response-Time Analysis

With static real-time analysis, we calculate the response-time bounds of digital systems for (external) events. Usually, we first calculate the WCET of each task in isolation before the worst-case response-time (WCRT) analysis takes the surrounding execution context (i.e., other tasks, the operating system, IRQs) into account. While this two-step approach reduces complexity, it accumulates pessimism as program-level flow constraints cannot interact with system-level constraints. With SysWCET [14], PLATIN can express the WCRT analysis for a task as a WCET analysis of the whole system while executing that task.

SysWCET formulates an ILP that encodes not only the intra- and inter-procedural control flow graphs but also the system-state transition graph (SSTG) [13], thus allowing for function- and system-level flow constraints. To calculate the SSTG, we perform an abstract interpretation of the complete system, including the operating system, preempting interrupts, and all tasks with their (fixed-priority) scheduling semantics. Hence, the SSTG includes all synchronous and asynchronous control-flow transitions between tasks and interrupt handlers.

For SysWCET, we extended PML to store the different control-flow levels (function, task, system) and generalized the IPET to encode those levels into a single ILP simultaneously. Hence, ILP-encoded flow-fact constraints can include variables from all control-flow levels. For example, SysWCET can express that two branches in different tasks are mutually exclusive, further tightening the WCRT bounds. Furthermore, parametric annotation languages (see Section 4.3) allow for more accurate, context-sensitive timing bounds.

4.2 SysWCEC: Whole-System Energy-Consumption Analysis

A further extension to the system-state graph provides PLATIN with knowledge about the devices present in the system, their state (on/off), and how much power they draw in the respective state. Combined with the timing analysis, this enables PLATIN to yield worst-case bounds on the energy consumption of the analyzed systems [54]. Comparable to SysWCET, the energy-related analysis can determine the code's worst-case energy demand between two arbitrary program points. Thereby, the analysis determines the *worst-case response energy consumption* of tasks, that is, the demand from start to finish of an operation, including all power-state changes and the scheduling semantics. This interplay between types of worst-case analyses underlines the usability of analysis techniques originally introduced for timeliness to also work for the energy resource.

Beyond the modeling of simple on/off states of devices, an additional enhancement keeps track of internal device states and configurations, enabling fine-grained modeling of device behavior across system states [42]. As a result, this enables PLATIN to derive more accurate resource bounds, for example, for modeling the states of transceiver devices. The scope of these energy-related extensions goes beyond the worst-case execution-time analysis of real-time systems since these analyses are beneficial for highly energy-constrained systems, such as intermittently-powered embedded systems. One example is systems with intermittent power supply that, for example, harvest their energy through solar cells within the battery-free Internet of Things [1].

4.3 System-Wide Annotation Support

Within system-wide analyses, the operating system's kernel represents an interesting target for the static timing analysis, as the WCET of a system call is not static but heavily depends on the system state. One solution proposed [37] to resolve this lack of application information within kernel-level analyses is to move to a parametric analysis that can jointly

```
void func(void *data, size_t len) {
  for (size_t i = 0; i < len; i++) {
    #pragma platina lbound "max_len"
    /* ... */
  }
}
```

```
#pragma platina let "max_len=12"
func(input, 12);

#pragma platina let \

"max_len=NUM_TASKS"
func(tasks, numReadyTasks());
```

Figure 2 Parametric loop annotation in function `func` assigning context-sensitive values to the symbolic variable `max_len` at the call sites as manual loopbound (12) and system fact (`NUM_TASKS`).

Table 2 Reasoning for excluding some of the TACLeBench benchmarks from the evaluation.

Reason for exclusion	Benchmarks
Recursion	ammunition, anagram, bitcount, bitonic, fac, huff_enc, quicksort, recursion
Not self-contained	DEBIE, PapaBench, rosace
Infeasible loop bound	rijndael_dec, rijndael_enc

consider both applications and the backing RTOS. SWAN [48] addresses this challenge by introducing system facts, a unit of information gained from system-level analysis and referenced in source-level, parametric annotations within the operating system code. By evaluating the annotation expressions over the interaction's system facts and lowering the flow facts gained to the machine-code level with the aid of CFRGs [25], Platin can thus yield system-context–specific timing bounds for individual system calls. PragMetis [49] extends this parameterization from a per-system-interaction level to smaller structural contexts such as call- and loop-contexts, as shown by the example in Figure 2. This allows Platin to express parametricity within a single system call and eases the use of parametric annotations within application-level libraries that often exhibit similar context sensitivity.

5 Evaluation

We evaluate the performance of Platin on the TACLeBench benchmark suite [17]. However, we had to exclude some programs. As Table 2 shows, eight programs were excluded for using recursion, as Platin cannot handle recursion. The three `Parallel` benchmarks (`DEBIE`, `PapaBench`, `rosace`) were excluded for not being self-contained and needing OS support for threading. Finally, two benchmarks (`rijndael_dec`, `rijndael_enc`) include invalid loop bounds. We excluded the benchmarks as Platin requires correct bounds to produce meaningful results. After these necessary exclusions, 47 benchmarks remain for the evaluation.

In Table 3, we give the Platin-provided bound for the remaining benchmarks of the TACLeBench suite. We compare the measured execution time for each architecture against the Platin bound. The measured times for the Patmos target are with the data cache disabled, equivalent to Platin assuming all data-cache accesses miss. This can give us a slight sense of the efficiency of Platin, though we must stress that our measured times are not guaranteed to be the true WCETs since TACLe does not guarantee the input exercises the worst-case path. For the RISC-V target, we work with an ESP32-C3 processor, which features an RV32IMC instruction set and a single-cycle–accessible SRAM [15]. The SRAM has a storage capacity of 400 KB, which was large enough for our tests. We employed a measurement-based approach to determine the WCET of each instruction available on the processor. The benchmarks for the ESP32-C3 are executed on the `ESP32-C3-DevKitM-1 v1.0` development board. We use the CPU cycle counter available on the ESP32-C3 to determine the run time of each benchmark. For floating-point operations, a software-float implementation is used for the ESP32-C3. We omitted a WCET for the floating-point library instructions in PLATIN, and, therefore, the related benchmarks (marked with ≋) are skipped. For the Patmos target, five additional benchmarks were excluded for incorrect compilation (marked with 🐞). The 🖩 mark is used when PLATIN failed to provide a valid bound.

The third column of each target is the pessimism, i.e., by how much the given bound is higher than the measured execution time. For the Patmos target, we can see that the pessimism ranges widely. For some of the simpler programs, PLATIN was able to identify the measured execution time as the WCET. This is not surprising in simple cases like `countnegative`, however, it is surprising in more complex cases like `h264_dec`.[2] On the other hand, some benchmarks have very high pessimism. For example, the pessimism of `fft` is 25 820 %, which is likely down to this program having nested loops with big ranges between minimum and maximum loop bounds.[3] A similar picture emerges for the RISC-V evaluations: most of the pessimism ranges from 80 % to 300 %. The pessimism of `fft` again marks an outlier with 42 590 %, with the same reasoning as for the Patmos target. Unlike the Patmos target, PLATIN does not produce any bound equal to the WCET for the RISC-V target. This is mainly due to pessimism introduced by the “C” extension of RISC-V: The `compressed` instruction-set extension offers shorter codes (2-byte instead of 4-byte) for often-used instructions. Therefore, the control flow during branching instructions can enter at addresses that are not 4-byte-aligned. As the bus only supports loading 4-byte-aligned code, a 4-byte instruction may need two loads instead of one to fetch the entire instruction from memory. This pessimism at branching instructions leads to more overestimates than the Patmos target.

6 Related Work

Worst-Case Analysis Tools. To this point, numerous WCET analyzer tools have been developed for different hardware platforms. Several analyzers stem from academia [8, 16, 19, 21, 22, 24, 30, 31, 34, 36, 40, 46] and, based on these results, commercially available products are available [10, 19, 30]. This underlines the importance of WCET analysis in safety-critical real-time systems.

Hybrid WCET Analysis. With the increasing complexity of modern high-performance multicore microarchitectures, the use of hybrid WCET tools is gaining in importance: Determining accurate timing models of the target architecture can become practically infeasible with the lack of documentation and unpredictable components. In this context, the TimeWeaver tool [30] presents a hybrid approach: This approach combines timing information from measurements with static analysis techniques. Such hybrid resource-consumption approaches are also interesting in the context of PLATIN’s system-wide analysis techniques [14, 54].

LLVMTA. The infrastructure of LLVMTA [21] is related to the PLATIN infrastructure, with both projects relying on the LLVM framework. LLVMTA focuses on microarchitectural analysis and implements its analyses on the final assembler representation in the LLVM backend. LLVMTA has no integration into the clang compiler, comparable to PLATIN’s support of control-flow-relation graphs. That is, LLVMTA cannot exploit high-level source code information within the resource-consumption analysis.

[2] Remember, no data caches are used.
[3] Remember, the suite does not guarantee the programs exhibit WCET.

Table 3 Comparison of measured execution times and WCET bounds provided by PLATIN. Bounds with a '*' use the `gurobi` optimizer instead of the default `lp_solve`.

	Patmos			RISC-V		
benchmark	Measured	Bound	Pessimism	Measured	Bound	Pessimism
lift	2 567 285	6 506 322	153 %	1 738 754	3 846 697	121 %
powerwindow	12 601 467	24 599 225	95 %	3 930 880	10 387 998	164 %
binarysearch	369	449	22 %	232	409	76 %
bsort	492 507	961 942	95 %	322 824	1 086 659	237 %
complex_updates	591 526	1 047 923	77 %		💨	
cosf	12 755 728	37 280 642	192 %		💨	
countnegative	13 000	13 000	0 %	21 463	37 353	74 %
cubic	89 339 041	256 701 960	187 %		💨	
deg2rad	7 017 167	11 842 517	69 %		💨	
fft	2 474 043	641 279 936	25 820 %	1 288 291	549 970 763*	42 590 %
filterbank	1 687 548 481	4 175 572 460	147 %		💨	
fir2dim	1 874 513	3 742 975	100 %		💨	
iir	90 327	301 992	234 %		💨	
insertsort	10 080	16 160	60 %	2 804	7 205	157 %
isqrt		🐞		1 821 949	3 322 333*	82 %
jfdctint	8 000	8 000	0 %	4 553	8 396	84 %
lms	76 108 993	144 454 450	90 %		💨	
ludcmp		🐞			💨	
matrix1	98 586	98 586	0 %	37 517	66 523	77 %
md5	71 415 634	258 563 319	262 %	36 390 690	192 082 391	428 %
minver	351 107	1 568 068	347 %		💨	
pm		🖩			💨	
prime		🐞		1 580	4 879	209 %
rad2deg	7 065 136	11 809 717	67 %		💨	
sha		🐞		5 526 979	12 060 259	118 %
st	78 350 471	134 464 808	72 %		💨	
adpcm_dec	13 315	13 786	4 %	5 594	11 852	112 %
adpcm_enc	17 724	19 454	10 %	11 615	20 526	77 %
audiobeam	137 166 947	241 512 652	76 %		💨	
cjpeg_transupp	13 591 503	125 542 054	824 %	8 047 222	129 576 647	1 510 %
cjpeg_wrbmp	275 719	279 767	1 %	280 943	508 787	81 %
dijkstra	192 399 577	32 480 864 150*	16 782 %	121 641 591	16 510 486 058	13 473 %
epic	1 406 490 126	486 510 736 766*	34 490 %		💨	
fmref	256 117 759	795 672 591	211 %		💨	
g723_enc	2 488 842	3 756 308	51 %	1 402 196	4 911 683	250 %
gsm_dec	4 387 842	10 534 654	140 %	4 688 449	25 143 501	436 %
gsm_enc	14 680 073	20 184 547	37 %	9 834 175	31 178 349	217 %
h264_dec	49 621	49 621	0 %	144 429	445 724	209 %
huff_dec	820 259	2 293 420	180 %	529 762	2 104 049	297 %
mpeg2	1 040 665 400	56 804 616 615	5 358 %	535 872 047	41 835 888 109*	7 707 %
ndes	246 594	260 249	6 %	143 728	248 207	73 %
petrinet	8 960	36 728	310 %	2 517	7 648	204 %
statemate	67 364	117 113	74 %	81 270	290 443	257 %
susan		🐞			💨	
cover	2 098	2 758	31 %	58 740	199 374	239 %
duff	2 565	2 628	2 %		🖩	
test3	1 943 674 306	2 029 568 191	4 %	487 414 145	941 044 973	93 %

Compiler & WCET-Analysis Integration. Bernat and Holsti presented a wish list of compiler features that could aid WCET analysis [9]. The list included various features that would aid analysis, such as providing program control flow structure, various properties of the code, and controlling code generation. Many of the essential features are supported in our compiler, with data export through the PML format to aid PLATIN. This, of course, includes the control flow, flow facts, and user annotations. The compiler is missing most feature sets for source code-to-object code mapping and other features, such as the logical effects of code sub-sequences. Other compilers also implement dedicated support for WCET analysis: Li et al. introduced a framework for maintaining flow information during compiler optimizations [33]. Falk et al. introduced the WCET-aware C Compiler (WCC), which can automatically call the aiT WCET analyzer and change the code generation to minimize the WCET [18]. Schommer et al. extend the CompCert certified C compiler with support for AIS annotations [47]. These annotations are then embedded in a dedicated section in the ELF, which the WCET analyzer can consume.

Worst-Case Energy-Consumption Analysis. The original use of the IPET targeted the time resource for real-time systems. Later, Jayaseelan et al. [28] introduced the usability of WCET techniques for the energy resource to yield worst-case energy-consumption estimations, leading to further research on WCEC analysis [28, 38, 42, 53, 54, 55, 56]. With the PLATIN toolkit, we explored system-wide WCEC analysis [54] and the modeling of context-sensitive device states [42]. We consider the PLATIN framework a fundamental basis for our further work in this area for addressing energy-constrained systems.

7 Conclusion

Real-time systems need to prove the absence of deadline misses. To ensure this property, we need schedulability analysis and static WCET analysis of the individual tasks. This paper presented PLATIN, an open-source worst-case analysis tool targeting Patmos, RISC-V, ARM, and AVR processors. The PLATIN toolkit, initially introduced for WCET analysis, has also proven to be suitable for analyzing tasks' worst-case energy consumption to address energy-constrained systems. Extensions to PLATIN include system-wide analyses and expressive annotation support. In our evaluations, we show the multi-target capabilities of PLATIN by providing WCET bounds for two different processors (Patmos and a RISC-V variant) for the TACLe benchmarks. We envision that PLATIN will be extended to other real-time processors, e.g., the FlexPRET processor [58]. PLATIN is available as open-source software, simplifying cooperation between research groups on developing static worst-case analysis tools.

References

1 Saad Ahmed, Bashima Islam, Kasim Sinan Yildirim, Marco Zimmerling, Przemysław Pawełczak, Muhammad Hamad Alizai, Brandon Lucia, Luca Mottola, Jacob Sorber, and Josiah Hester. The internet of batteryless things. *Communications of the ACM*, 67(3):64–73, February 2024. doi:10.1145/3624718.

2 ARM Limited. ARMv6-M Architecture Reference Manual, 2010.

3 ARM Limited. Cortex-M0+ Technical Reference Manual, 2012.

4 ARM Limited. *ARMv7-M Architecture Reference Manual*, 2014.

5 ARM Limited. *ARM Cortex-M4 – Technical Reference Manual*, 2015. Revision: r0p1.

6 Atmel Corporation. *AATmega1284P – 8-bit Microcontroller with 128K Bytes In-System Programmable Flash*, 8059d–avr–11/09 edition, 2009. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/doc8059.pdf.

7 Atmel Corporation. *AVR Instruction Set Manual*, atmel-0856l-avr-instruction-set-manual_other-11/2016 edition, 2016. URL: `https://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`.

8 C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *Proceedings of the 8th International Workshop on Software Technolgies for Embedded and Ubiquitous Systems (SEUS '10)*, pages 35–46, 2010.

9 Guillem Bernat and Niklas Holsti. Compiler support for wcet analysis: a wish list. In *WCET*, pages 65–69, 2003.

10 Adam Betts, Nicholas Merriam, and Guillem Bernat. Hybrid measurement-based wcet analysis at the source level using object-level traces. In *Proceedings of the 10th International Workshop on Worst-Case Execution Time Analysis (WCET '10)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

11 Philipp Degasperi, Stefan Hepp, Wolfgang Puffitsch, and Martin Schoeberl. A method cache for Patmos. In *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*, pages 100–108, Reno, Nevada, USA, June 2014. IEEE. `doi:10.1109/ISORC.2014.47`.

12 Eva Dengler, Phillip Raffeck, Simon Schuster, and Peter Wägemann. Fusionclock: Energy-optimal clock-tree reconfigurations for energy-constrained real-time systems. In *Proceedings of the 35th Euromicro Conference on Real-Time Systems (ECRTS '23)*, volume 262, pages 6:1–6:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

13 Christian Dietrich, Martin Hoffmann, and Daniel Lohmann. Cross-kernel control-flow-graph analysis for event-driven real-time systems. In *Proceedings of the 2015 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '15)*, New York, NY, USA, June 2015. ACM Press. `doi:10.1145/2670529.2754963`.

14 Christian Dietrich, Peter Wägemann, Peter Ulbrich, and Daniel Lohmann. SysWCET: Whole-system response-time analysis for fixed-priority real-time systems. In *Proceedings of the 23nd Real-Time and Embedded Technology and Applications Symposium (RTAS '17)*, pages 37–48, 2017.

15 Espressif Systems. *ESP32-C3 Series Datasheet Ultra-Low-Power SoC with RISC-V Single-Core CPU*, 2024. Version 1.6. URL: `https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf`.

16 H. Falk and P. Lokuciejewski. A compiler framework for the reduction of worst-case execution times. *Real-time Systems*, 46(2):251–300, 2010.

17 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In Martin Schoeberl, editor, *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55 of *OpenAccess Series in Informatics (OASIcs)*, pages 2:1–2:10, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. `doi:10.4230/OASIcs.WCET.2016.2`.

18 Heiko Falk, Paul Lokuciejewski, and Henrik Theiling. Design of a wcet-aware c compiler. In *2006 IEEE/ACM/IFIP Workshop on Embedded Systems for Real Time Multimedia*, pages 121–126. IEEE, 2006.

19 Christian Ferdinand and Reinhold Heckmann. aiT: Worst-case execution time prediction by static program analysis. *Building the Information Society*, 156:377–383, 2004.

20 Freescale Semiconductor, Inc. *KL46 Sub-Family Reference Manual*, 2013.

21 Sebastian Hahn, Michael Jacobs, Nils Hölscher, Kuan-Hsun Chen, Jian-Jia Chen, and Jan Reineke. LLVMTA: An llvm-based wcet analysis tool. In Clément Ballabriga, editor, *Proceedings of the 20th International Workshop on Worst-Case Execution Time Analysis (WCET '22)*, pages 2:1–2:17, 2022.

22 D. Hardy, B. Rouxel, and I. Puaut. The Heptane static worst-case execution time estimation tool. In Jan Reineke, editor, *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET '17)*, volume 57 of *OpenAccess Series in Informatics (OASIcs)*, pages 8:1–8:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2017.8.

23 Stefan Hepp, Benedikt Huber, Jens Knoop, Daniel Prokesch, and Peter P. Puschner. The platin tool kit - the T-CREST approach for compiler and WCET integration. In *Proceedings 18th Kolloquium Programmiersprachen und Grundlagen der Programmierung, KPS 2015, Pörtschach, Austria, October 5-7, 2015*, 2015.

24 N. Holsti and S. Saarinen. Status of the Bound-T WCET tool. In *Proceedings of the 2nd International Workshop on Worst-Case Execution Time Analysis (WCET '02)*, pages 36–41, 2002.

25 B. Huber, D. Prokesch, and P. Puschner. Combined WCET analysis of bitcode and machine code using control-flow relation graphs. In *Proceedings of the 14th Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '13)*, pages 163–172, 2013.

26 Benedikt Huber, Stefan Hepp, and Martin Schoeberl. Scope-based method cache analysis. In *Proceedings of the 14th International Workshop on Worst-Case Execution Time Analysis (WCET 2014)*, pages 73–82, Madrid, Spain, July 2014. doi:10.4230/OASIcs.WCET.2014.73.

27 Infineon Technologies AG. *XMC4500 Microcontroller Series for Industrial Applications — Reference Manual*, 2016. V1.6 2016-07.

28 Ramkumar Jayaseelan, Tulika Mitra, and Xianfeng Li. Estimating the worst-case energy consumption of embedded software. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '06)*, pages 81–90, 2006. doi:10.1109/RTAS.2006.17.

29 Alexander Jordan, Florian Brandner, and Martin Schoeberl. Static analysis of worst-case stack cache behavior. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems (RTNS 2013)*, pages 55–64, New York, NY, USA, 2013. ACM. doi:10.1145/2516821.2516828.

30 Daniel Kästner, Markus Pister, Simon Wegener, and Christian Ferdinand. TimeWeaver: A tool for hybrid worst-case execution time analysis. In Sebastian Altmeyer, editor, *Proceedings of the 19th International Workshop on Worst-Case Execution Time Analysis (WCET '19)*, volume 72 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:11, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2019.1.

31 Raimund Kirner. The wcet analysis tool CalcWcet167. In *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '12)*, pages 158–172, 2012.

32 C. Lattner and V. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the International Symposium on Code Generation and Optimization (CGO '04)*, pages 75–86, 2004.

33 Hanbing Li, Isabelle Puaut, and Erven Rohou. Traceability of flow information: Reconciling compiler optimizations and wcet estimation. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, pages 97–106, 2014.

34 Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Science of Computer Programming*, 69(1):56–67, 2007.

35 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *ACM SIGPLAN Notices*, volume 30(11), pages 88–98. ACM, 1995.

36 B. Lisper. SWEET – a tool for WCET flow analysis. In *Proceedings of the 6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA '14)*, pages 482–485. Springer, 2014.

37 Mingsong Lv, Nan Guan, Yi Zhang, Qingxu Deng, Ge Yu, and Jianming Zhang. A survey of WCET analysis of real-time operating systems. In *Proceedings of the 2009 International*

Conference on Embedded Software and Systems (ICESS'09), pages 65–72, 2009. doi:10.1109/ICESS.2009.24.

38 James Pallister, Steve Kerrison, Jeremy Morse, and Kerstin Eder. Data dependent energy modeling for worst case energy consumption analysis. In *Proceedings of the 20th Workshop on Software and Compilers for Embedded Systems*, pages 51–59, 2017. doi:10.1145/3078659.3078666.

39 P. Puschner and A. Schedl. Computing maximum task execution times: A graph-based approach. *Real-Time Systems*, 13:67–91, 1997.

40 Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, and Gernot Gebhard. The t-crest approach of compiler and wcet-analysis integration. In *Proceedings of the 16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)*, pages 1–8, 2013.

41 Phillip Raffeck, Christian Eichler, Peter Wägemann, and Wolfgang Schröder-Preikschat. Worst-case energy-consumption analysis by microarchitecture-aware timing analysis for device-driven cyber-physical systems. In *Proceedings of the 19th International Workshop on Worst-Case Execution Time Analysis (WCET '19)*, pages 6:1–6:12, 2019. doi:10.4230/OASIcs.WCET.2019.4.

42 Phillip Raffeck, Johannes Maier, and Peter Wägemann. WoCA: Avoiding intermittent execution in embedded systems by worst-case analyses with device states. In *Proceedings of the 25th ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '24)*, 2024. URL: https://sys.cs.fau.de/publications/2024/raffeck_24_lctes.pdf.

43 RISC-V Foundation. *The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA*, December 2019. Document Version 20191213. URL: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf.

44 Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, Stefan Hepp, Benedikt Huber, Alexander Jordan, Evangelia Kasapaki, Jens Knoop, Yonghui Li, Daniel Prokesch, Wolfgang Puffitsch, Peter Puschner, André Rocha, Cláudio Silva, Jens Sparsø, and Alessandro Tocchi. T-CREST: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015. doi:10.1016/j.sysarc.2015.04.002.

45 Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, and Daniel Prokesch. Patmos: A time-predictable microprocessor. *Real-Time Systems*, 54(2):389–423, April 2018. doi:10.1007/s11241-018-9300-4.

46 Martin Schoeberl, Wolfgang Puffitsch, Rasmus Ulslev Pedersen, and Benedikt Huber. Worst-case execution time analysis for a Java processor. *Software: Practice and Experience*, 40/6:507–542, 2010. doi:10.1002/spe.968.

47 Bernhard Schommer, Christoph Cullmann, Gernot Gebhard, Xavier Leroy, Michael Schmidt, and Simon Wegener. Embedded program annotations for wcet analysis. In *WCET 2018: 18th International Workshop on Worst-Case Execution Time Analysis*, volume 63. Dagstuhl Publishing, 2018.

48 Simon Schuster, Peter Wägemann, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Proving Real-Time Capability of Generic Operating Systems by System-Aware Timing Analysis. In *Proceedings of the 25th Real-Time and Embedded Technology and Applications Symposium (RTAS '19)*, pages 318–330, Montreal, 2019. ieee.

49 Simon Schuster, Peter Wägemann, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Annotate once – analyze anywhere: Context-aware wcet analysis by user-defined abstractions. In *Proceedings of the 22nd ACM International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES '21)*, pages 54–66, 2021. doi:10.1145/3461648.3463847.

50 Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. An end-to-end toolchain: From automated cost modeling to static WCET and WCEC analysis. In *Proceedings of the 20th International Symposium on Real-Time Distributed Computing (ISORC '17)*, pages 1–10, 2017.

51 Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. Combining automated measurement-based cost modeling with static worst-case execution-time and energy-consumption analyses. *IEEE Embedded Systems Letters*, 11(2):38–41, 2019. doi:10.1109/LES.2018.2868823.

52 Ingmar Jendrik Stein. *ILP-based path analysis on abstract pipeline state graphs.* epubli, 2010.

53 D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. Worst-case energy consumption: A new challenge for battery-powered critical devices. *IEEE Transactions on Sustainable Computing*, pages 1–8, 2019. doi:10.1109/TSUSC.2019.2943142.

54 Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. Whole-system worst-case energy-consumption analysis for energy-constrained real-time systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS '18)*, volume 106, pages 24:1–24:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ECRTS.2018.24.

55 Peter Wägemann, Tobias Distler, Timo Hönig, Heiko Janker, Rüdiger Kapitza, and Wolfgang Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS '15)*, pages 105–114, 2015. doi:10.1109/ECRTS.2015.17.

56 Simon Wegener, Kris K. Nikov, Jose Nunez-Yanez, and Kerstin Eder. EnergyAnalyzer: Using static wcet analysis techniques to estimate the energy consumption of embedded applications. In Peter Wägemann, editor, *Proceedings of the 21th International Workshop on Worst-Case Execution Time Analysis (WCET '23)*, volume 114 of *Open Access Series in Informatics (OASIcs)*, pages 9:1–9:14, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.WCET.2023.9.

57 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (ACM TECS)*, 7(3):1–53, 2008.

58 Michael Zimmer, David Broman, Chris Shaver, and Edward A Lee. FlexPRET: A processor platform for mixed-criticality systems. In *Proceedings of the 19th Real-Time and Embedded Technology and Applications Symposium (RTAS '14)*, pages 101–110, 2014.