An Empirical Evaluation of the Influence of the Load-Store Unit on WCET Analysis*

Mohamed Abdel Maksoud and Jan Reineke

Saarland University
Saarbrücken, Germany
{mohamed,reineke}@cs.uni-saarland.de

Abstract

Due to the complexity of today's micro-architectures, the micro-architectural analysis usually constitutes the most time-consuming step in worst-case execution time (WCET) analysis.

In this paper, we investigate the influence of the design of the load-store unit (LSU) in the PowerPC 7448 on WCET analysis. To this end, we introduce a simplified variant of the existing design of the LSU by reducing its queue sizes. Using AbsInt's `aiT` WCET analysis toolchain we determine the resulting WCET bounds and analysis times.

For the modified version of the LSU with reduced queue sizes, analysis time is reduced by more than 50% on a set of benchmarks from the Mälardalen suite, while there is little change in the WCET bound.

1998 ACM Subject Classification B.8.2 Performance Analysis and Design Aids

Keywords and phrases empirical evaluation, architecture complexity effect, WCET analysis precision, WCET analysis performance, PowerPC 7448, Load-Store Unit

Digital Object Identifier 10.4230/OASIcs.WCET.2012.13

1 Introduction

The increasing complexity of today's micro-architectures makes the construction of sound and precise timing models an increasingly time-consuming and error-prone task. Furthermore, the resulting, complex timing models lead to a state-explosion problem in the micro-architectural (also known as low-level) analysis, drastically increasing overall WCET analysis times.

Most micro-architectural innovations, causing this increase in complexity, like speculation and out-of-order execution, are undertaken to improve average-case performance. In the WCET community it is often argued that many of these innovations do not improve, or even harm, a processor's worst-case timing behavior. There is, however, little hard evidence supporting such claims. This paper intends to contribute some hard evidence by performing an empirical evaluation using AbsInt's `aiT` WCET analysis toolchain.

The PowerPC 7448 is a high-performance microprocessor used in safety-critical real-time scenarios featuring caches, pipelining, speculation and out-of-order execution. To support speculation and out-of-order execution, the PowerPC 7448 includes a load-store unit (LSU), maintaining queues of memory instructions in different execution states. We investigate the influence of the lengths of these queues on both WCET bounds and WCET analysis times.

* The research leading to these results was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

12th International Workshop on Worst-Case Execution Time Analysis (WCET 2012).
Editor: Tullio Vardanega; pp. 13–24

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

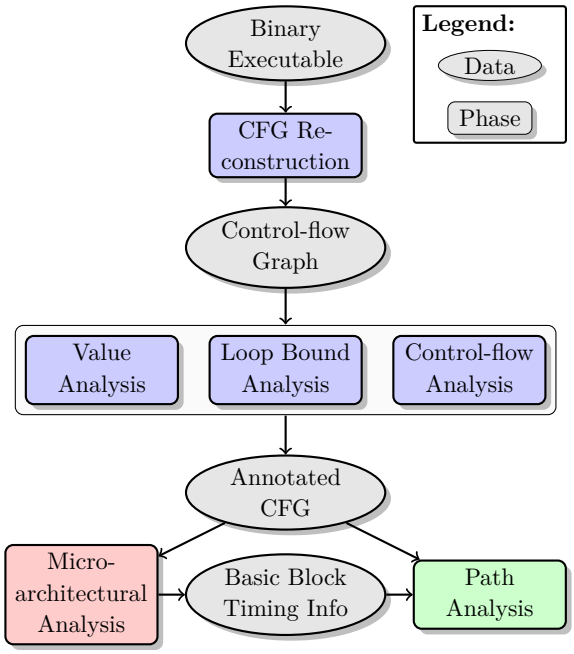

Figure 1 Main components of a timing-analysis framework and their interaction.

To this end, we introduce a simplified variant of the existing LSU, reducing its queue sizes to a minimum.

We compare the simplified design with the original design on various benchmarks from the Mälardalen benchmark suite. Surprisingly, we observe slightly decreased WCET bounds, and, as expected, strongly reduced analysis times.

2 Background

2.1 WCET Analysis Flow

Over roughly the last decade, a more or less standard architecture for timing-analysis tools has emerged. Figure 1 gives a general view of this architecture. The following list presents the individual phases and describes their objectives.

1. *Control-flow reconstruction* [21] takes a binary executable to be analyzed, reconstructs the program's control flow and transforms the program into a suitable intermediate representation. Problems encountered are dynamically computed control-flow successors, e.g. those stemming from switch statements, function pointers, etc.
2. *Value analysis* [4] computes an over-approximation of the set of possible values in registers and memory locations by an interval analysis and/or congruence analysis. The computed information is used for a precise data-cache analysis and in the subsequent control-flow analysis. Value analysis is the only one to use an abstraction of the processor's arithmetic. A subsequent pipeline analysis can therefore work with a simplified pipeline where the arithmetic units are removed. There, one is not interested in what is computed, but only in how long it will take.
3. *Loop bound analysis* [8, 14] identifies loops in the program and tries to determine bounds on the number of loop iterations; information indispensable to bound the execution time.

Problems are the analysis of arithmetic on loop counters and loop exit conditions, as well as dependencies in nested loops.

4. *Control-flow analysis* [8, 20] narrows down the set of possible paths through the program by eliminating infeasible paths or by determining correlations between the number of executions of different blocks using the results of value analysis. These constraints will tighten the obtained timing bounds.
5. *Micro-architectural analysis* [7, 23, 10, 5] determines bounds on the execution time of basic blocks by performing an abstract interpretation of the program, combining analyses of the processor's pipeline, caches, and speculation. Static cache analyses determine safe approximations to the contents of caches at each program point. Pipeline analysis analyzes how instructions pass through the pipeline accounting for occupancy of shared resources like queues, functional units, etc.
6. *Path Analysis* [17, 22] finally determines bounds on the execution times for the whole program by implicit path enumeration using an integer linear program (ILP). Bounds of the execution times of basic blocks are combined to compute longest paths through the program. The control flow is modeled by Kirchhoff's law. Loop bounds and infeasible paths are modeled by additional constraints. The target function weights each basic block with its time bound. A solution of the ILP maximizes the sum of those weights and corresponds to an upper bound on the execution times. In the following, we refer to the kind of path analysis described above as *traditional* ILP-based analysis.

The commercially available tool `aiT` by AbsInt, cf. `http://www.absint.de/wcet.htm`, implements this architecture. It is used in the aeronautics and automotive industries and has been successfully used to determine precise bounds on execution times of real-time programs [10, 9, 24, 15].

The ILP-based path analysis in `aiT` comes in two variants depending on how micro-architectural state graphs are constructed [2]:

1. *Traditional ILP-based analysis*, where an ILP is solved to find the worst-case path through the program, given worst-case timings of all basic blocks (possibly in various contexts). This approach may be imprecise, because the worst-case timings of some basic blocks may not occur simultaneously on a single architectural path through the program.
2. *Prediction-file-based ILP analysis*, where a global state graph consisting of micro-architectural states is constructed, and an ILP is solved to find the worst-case path. This results in a more precise WCET bound since architecturally-infeasible paths are excluded. However, it comes at the cost of a much larger ILP to be solved.

To illustrate the difference between the two path-analysis methods, consider the example analysis shown in Figure 2. An ILP-based path analysis computes a global WCET bound solely based

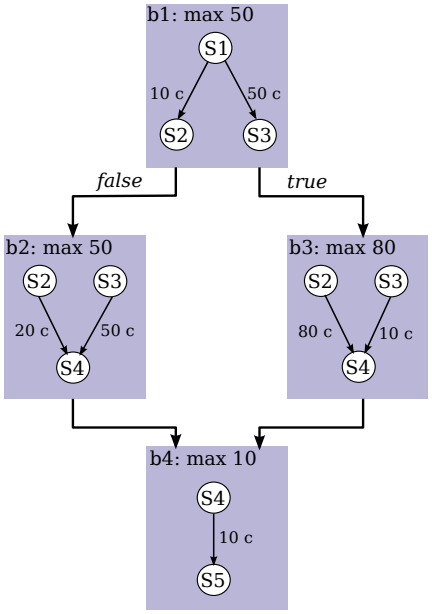

Figure 2 An example illustrating the differences between traditional ILP-based path analysis and prediction-file-based ILP path analysis.

on the maximum number of execution cycles for each basic block. The WCET is therefore 140 cycles in this case, and the worst-case exeuction path is `b1`→`b3`→`b4`. However, this result implies an architecturally-infeasible execution trace: `s1`→`s3`|||`s2`→`s4`→`s5`, where trace discontinuity is marked by |||.

On the other hand, the global state graph constructed in a prediction-file-based ILP path analysis excludes such paths and produces a WCET bound of 110 cycles, with the corresponding worst-case execution path: `b1`→`b2`→`b4`, and trace: `s1`→`s3`→`s4`→`s5`.

2.2 Motorola PowerPC 7448

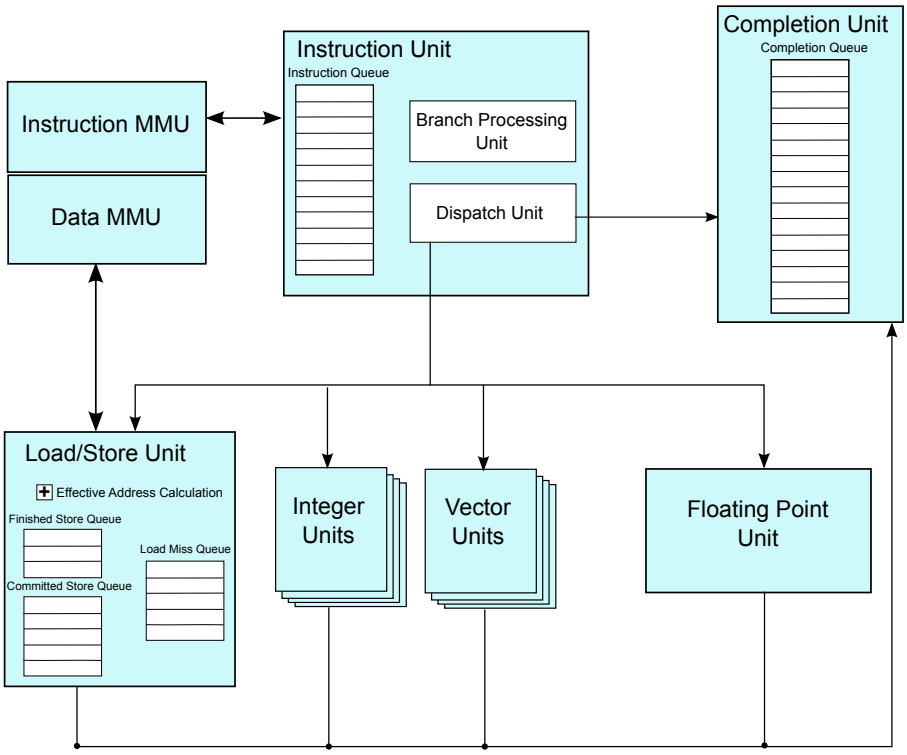

Figure 3 PowerPC 7448 Microprocessor Block Diagram.

The PowerPC 7448 is a reduced instruction set computer (RISC) superscalar processor that implements the 32-bit portion of the PowerPC architecture and the SIMD instruction set AltiVec architectural extension. It features a two-level memory hierarchy with separate L1 data and instruction caches (Harvard architecture), a unified L2 cache, four independent integer and four independent vector units for superscalar execution. It also features static and dynamic branch prediction, and a sophisticated load-store unit with long buffers.

"The PowerPC 7448 provides virtual memory support for up to 4 PB (2^{52}) of virtual memory and real memory support for up to 64 GB (2^{36}) of physical memory. It can dispatch and complete three instructions simultaneously" [11]. It consists of the following execution units, depicted in Figure 3:

- Instruction Unit (IU): the IU provides centralized control of instruction flow to the execution units. It contains an instruction queue (IQ), a dispatch unit (DU), and a branch processing unit (BPU). The IQ has 12 entries and loads up to 4 instructions

from the instruction cache in one cycle. The DU checks register dependencies and the availability of a position in the *completion queue* (described below), and issues or inhibits subsequent instruction dispatching accordingly. The BPU receives branch instructions from the IQ and executes them early in the pipeline. If a branch has a dependency that has not yet been resolved, the branch path is predicted using either architecture-defined static branch prediction or PowerPC 7448 -specific dynamic branch prediction.

- Completion Unit (CU): The CU retires an instruction from the 16-entry completion queue (CQ) when all instructions ahead of it have been completed. The CU operates closely with the IU to ensure that the instructions are retired in program order.
- Integer, Vector, and Floating-Point Units: the PowerPC 7448 provides nine execution units to support the execution of integer, fixed point, and AltiVec instructions.
- Cache/Memory Subsystem: The PowerPC 7448 microprocessor contains two separate 32-Kbyte, eight-way set-associative level 1 (L1) instruction and data caches (Harvard architecture). The caches implement a pseudo least-recently-used (PLRU) replacement policy. In addition, the PowerPC 7448 features an integrated 1 MB level 2 (L2) cache.
- Load-Store Unit (LSU): The LSU executes all load and store instructions and provides the data transfer interface between registers and the cache/memory subsystem. The LSU also calculates effective address and aligns data. This unit is described in detail in the following section.

Load-Store Unit

The LSU provides all the logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load-store string and load-store multiple operations [11]. The LSU contains a 5-entry load miss queue (LMQ) which maintains the load instructions that missed the L1 cache until they can be serviced. This allows the LSU to process subsequent loads. Unlike loads, stores cannot be executed speculatively: a store instruction is held in the 3-entry finished store queue (FSQ) until the completion unit signals that the store is committed, only then it moves to the 5-entry committed store queue (CSQ). In order to reduce the latency of loads dependent on stores, the LSU implements *data forwarding* from any entry in the CSQ before the data is actually written to the cache. When a load instruction misses, its address is compared to all entries in the CSQ. On a hit, the data is forwarded from the newest matching entry. If the address is also found in the FSQ, however, the LSU stalls since the newest data at this address could be updated should the store instruction in the FSQ be committed.

Analysis Model of the Load-Store Unit

During static analysis, crucial information on program execution such as register contents, cache contents and *bus clock offset* cannot be decided exactly. The bus clock offset is defined as the number of processor clock cycles until the next rising edge of the bus clock. When the analysis flow depends on such information, the analysis has to proceed in all possible paths to ensure a sound WCET bound in the presence of timing anomalies [18]. When the analysis is to proceed in more than one path, the analysis state has to be *split*, with the consequence of increasing the size of the state space during analysis and hence reducing analysis efficiency. The analysis model of the load-store unit reflects its structure in the concrete processor architecture, while accounting for non-determinism. In the load-store unit, the addresses of different memory accesses are represented in terms of intervals, rather than exact numbers. As we shall see in Section 4, the load-store unit is a significant source of splits in most cases,

and this is attributed to the long queues in this unit. As described in the previous section, data forwarding involves a number of comparisons for each missed load instruction. This number is proportional to the sizes of the load miss queue, committed store queue, and finished store queue. These comparisons are performed on imprecise addresses, resulting in potential splits when it cannot be decided whether addresses do alias or not. Moreover, having long queues in the LSU indicates more pending memory accesses in the core waiting to be served. Serving more accesses increases the possibility of querying non-exact bus clock offset, hence representing another source of splits. These observations motivate our experimental setup described in the following section.

3 Experimental Setup

To the end of reducing the number of splits, and thus improving analysis time, we modified the PowerPC 7448 by cutting the queue sizes in the load-store unit. The LMQ, CSQ, and FSQ sizes were reduced to 1, 2, and 1, respectively[1]. The benchmarks were selected from the Mälardalen benchmark suite [13]. These are the benchmarks for which the WCET analysis terminated successfully for both architectures. The analyzed programs are briefly described in Table 1. The benchmarks marked with * were not found in the official documentation although they are included in the test-suite distribution.

Table 1 List of benchmarks analyzed in the experiment.

Benchmark	Description	LOC
bs	Binary search for the array of 15 integer elements.	114
cnt	Counts non-negative numbers in a matrix.	267
expint	Series expansion for computing an exponential integral function.	157
fac	Computes the sum of factorials of a set of integers.	28
fibcall	Simple iterative Fibonacci calculation, used to calculate fib(30).	72
janne_complex	Nested loop program.	64
lcdnum	Read ten values, output half to LCD.	64
loop3*	Several loop patterns.	240
minmax*	Simple program with infeasible paths.	58
qurt	Root computation of quadratic equations.	166
sqrt	Square root function implemented by Taylor series.	77

The `aiT` analyzer was configured to use traditional ILP-based path analysis (with the CLP solver [1]) on all benchmarks and prediction-file based ILP path analysis only on some of them. Although the latter produces more precise WCET bounds, it is more computationally demanding as will be seen in the following section, and we were not able to finish the analysis of all of the benchmarks in time for this submission.

The experiment was performed on a 64-bit AMD Opteron machine with 16 processor cores at 2500 MHz and 64 GB of RAM. As the WCET analysis is not parallelized, we ran multiple analyses concurrently on this machine. As performance metrics, we use the micro-architectural-analysis time and the path-analysis time. On the analyzed benchmarks, these two metrics constitute on the average about 80% and 75% of the whole analysis time for the standard and reduced architectures, respectively.

[1] This is the strongest simplification we could apply without having to make significant changes to the micro-architectural analysis.

4 Experimental Results and Analysis

The analysis results of selected benchmarks using prediction-file-based ILP path analysis are shown in Table 2. We compute both the average of the relative changes (<average>) and the relative change of the sum of the respective values (<weighted average>).

Looking first at the analysis performance metrics, we see that the state space in the reduced architecture is significantly smaller than that of the standard architecture. This is manifested in the consistently lower number of splits in the micro-architectural analysis and path analysis time, cf. the `janne_complex` benchmark. For less memory-demanding benchmarks, such as `fac` and `fibcall`, we do not see significant improvement in the analysis performance.

Comparing the WCET bounds in both architectures yields a surprise: in half of the cases, the reduced architecture achieves a WCET bound that is *lower* than that of the standard architecture. A closer look at one of the benchmarks featuring this anomaly, `minmax`, reveals the following:

- There are no ambiguous memory accesses in this simple benchmark (i.e. all addresses are exact), the effect of data-forwarding on the analysis precision is therefore ruled out.
- The benchmark starts with several store instructions followed by a branch instruction.
- The standard architecture with its long store queues accomodates more pending stores and execute further instructions, including the branch. This results in more pending memory requests, both data and instruction accesses, and hence it is more likely to query non-exact bus clock offset.
- On the other hand, the reduced architecture accomodates fewer pending stores, hence it stalls on encountering more store instructions. This stalling is beneficial in that it leads to fewer pending memory accesses and hence reduces the loss of precision caused by the non-deterministic bus clock offset.

Using the less precise, yet significantly more efficient traditional ILP-based path analysis, more benchmarks were analyzed. The analysis results and performance metrics are shown in Table 3.

We observe an increased average speed-up of the micro-architectural analysis for the reduced architecture compared with the results for the prediction-file-based analysis. The increased average speed-up is attributed to analyzing more memory intensive benchmarks such as `bs` and `cnt`. The micro-architectural analysis time varies slightly from that in Table 2 for some benchmarks, likely due to interference on shared resources between multiple analyses running concurrently on the machine.

The path-analysis contribution to the total analysis performance is insignificant, compared to that of the micro-architectural analysis. In the traditional ILP-based approach, path-analysis time is expected to be independent of the complexity of the underlying micro-architecture. Unsurprisingly, we observe little differences between the standard and the reduced architecture.

The "WCET anomaly", i.e. lower WCET bounds for the reduced architecture, is more pronounced using this path-analysis method. This is not surprising since a larger number of paths with different timings through basic blocks, as is the case for the standard architecture, makes it more likely for the path analysis to compute an architecturally infeasible worst-case execution path. This adds up to the precision loss observed in the standard architecture.

Table 2 WCET bounds and performance metrics using prediction-file-based ILP path analysis.

Benchmark	WCET bound *in cycles*			μarch. analysis splits			μarch. analysis time *in seconds*			Path-analysis time *in seconds*			Total analysis time *in seconds*	
	std.	red.	Δ %	std.	red.	Δ %	std.	red.	Δ %	std.	red.	Δ %	std.	red.
fac	3,321	3,343	0.7	1,610	1,586	-1.5	0.6	0.6	-0.8	0.1	0.1	-7.1	1.1	1.1
fibcall	3,346	3,325	-0.6	844	789	-6.5	0.5	0.5	-0.2	0.1	0.1	1.9	0.9	0.9
janne_complex	20,005	19,846	-0.8	107,222	4,599	-95.7	21.4	1.1	-94.8	3445.4	0.9	-100.0	3470.1	2.5
lcdnum	1,969	1,996	1.4	20,533	8,506	-58.6	3.2	1.4	-55.5	1.5	0.6	-62.0	5.1	2.4
loop3	39,329	41,199	4.8	8,380	5,942	-29.1	4.5	3.8	-15.9	0.7	0.4	-38.6	8.2	7.2
minmax	1,629	1,500	-7.9	2,925	1,106	-62.2	1.7	0.8	-52.7	1.0	0.1	-91.0	3.1	1.2
qurt	17,817	17,953	0.8	98,859	25,459	-74.2	62.7	14.0	-77.7	54.5	10.8	-80.2	120.8	26.4
sqrt	5,096	4,976	-2.4	26,415	7,768	-70.6	15.6	3.4	-77.9	13.0	2.3	-81.9	29.8	6.5
<average>			-0.5			-46.9			-49.8			-57.3		
<weighted avg.>			1.8			-76.7			-79.1			-99.6		

$$\mu\text{arch. analysis splits} := \text{the total number of splits in the micro-architectural analysis,}$$

$$\Delta\%(b, \langle\text{measure}\rangle) := \frac{\langle\text{measure}\rangle_{\text{red.}}(b) - \langle\text{measure}\rangle_{\text{std.}}(b)}{\langle\text{measure}\rangle_{\text{std.}}(b)} \times 100,$$

$$\langle\text{average}\rangle(\langle\text{measure}\rangle) := \frac{\sum_{b \in \text{benchmarks}} \Delta\%(b, \langle\text{measure}\rangle)}{|\text{benchmarks}|}, \text{ and}$$

$$\langle\text{weighted avg.}\rangle(\langle\text{measure}\rangle) := \frac{\sum_{b \in \text{benchmarks}} \langle\text{measure}\rangle_{\text{red.}}(b) - \sum_{b \in \text{benchmarks}} \langle\text{measure}\rangle_{\text{std.}}(b)}{\sum_{b \in \text{benchmarks}} \langle\text{measure}\rangle_{\text{std.}}(b)} \times 100.$$

5 Related Work

While there is an abundance of work proposing more predictable or analyzable micro-architectures, there is not a lot of work that empirically studies the impact of simplifications of micro-architectures on WCET analysis time. Exceptions include the work of Grund et al. [12] and Burguière and Rochange [3].

Grund et al. [12] investigate several modifications of the branch target instruction cache of the PowerPC 56x. They observe that using LRU in place of FIFO replacement reduces analysis time drastically, as more memory accesses can be classified as hits or misses, thereby reducing the number of splits.

Burguière and Rochange [3] investigate the modeling complexity of various dynamic branch prediction schemes. Here, the modeling complexity is measured by the number of constraints, the number of variables, and the sizes of constraints in an ILP formulation of the behavior of the respective branch prediction schemes. This analysis is based on the assumption that the modeling complexity is strongly-correlated with the resulting analysis complexity. However, the actual analysis times are not analyzed.

Heckmann et al. [15] focus on the difficulty in modeling various architectural components, including caches and pipelines, and their influence on the precision of the resulting analyses. From their experience in modeling various processors they derive several recommendations regarding the design of processors for real-time systems. Later, Wilhelm et al. [26] describe properties of memory hierarchies, pipelines, and buses, which make timing analysis more complex and/or reduce its precision. Neither Heckmann et al. nor Wilhelm et al. provide an empirical evaluation of their recommendations.

Approaches aiming at improving predictability or analyzability include the EU projects Predator, Merasa [25], the PRET project [6], and the Java-Optimized Processor JOP [19]. These projects present entirely new processor designs. This makes it difficult to evaluate the impact of individual design choices on WCET analysis times. In the context of the JOP project, Huber et al. [16] analyze the influence of different object cache configurations on worst-case execution time estimates, varying several cache parameters and the background memory. They do not, however, analyze the impact of the design choices on analysis times.

6 Conclusions and Future Work

In this paper, we have investigated the influence of the design of the load-store unit on WCET analysis, in terms of analysis times and WCET bounds. Reducing the complexity of the LSU results in significantly shorter analysis times, and, surprisingly, sometimes even in slightly lower WCET bounds. We plan to investigate the influence of further components to get a more complete view of how strongly various components and their configurations influence WCET analysis.

Regarding the "WCET anomaly" found in some benchmarks analyzed in this paper, we are uncertain whether it is a product of analysis imprecision, or whether it corresponds to actual behaviors of the respective architectures. It will be future work to shed more light on this question.

Table 3 WCET bounds and performance metrics using traditional ILP-based path analysis.

Benchmark	WCET bound *in cycles*			μarch. analysis splits			μarch. analysis time *in seconds*			Path-analysis time *in seconds*			Total analysis time *in seconds*	
	std.	red.	Δ %	std.	red.	Δ %	std.	red.	Δ %	std.	red.	Δ %	std.	red.
bs	11,082	9,807	-11.5	166,466	18,458	-88.9	201.6	15.8	-92.1	0.0	0.0	7.7	201.8	16.1
cnt	44,285	38,399	-13.3	20,489,562	830,784	-95.9	16842.5	489.4	-97.1	0.1	0.1	-12.3	16843.8	490.7
expint	13,610	13,536	-0.5	4,127	2,873	-30.4	2.5	2.0	-20.6	0.1	0.1	3.7	3.6	3.1
fac	4,173	4,015	-3.8	1,610	1,586	-1.5	0.6	0.6	-1.7	0.0	0.0	0.0	1.0	1.0
fibcall	3,685	3,530	-4.2	844	789	-6.5	0.5	0.5	5.1	0.0	0.0	0.0	0.8	0.9
janne_complex	28,172	21,034	-25.3	107,222	4,599	-95.7	19.8	1.1	-94.7	0.0	0.0	0.0	20.1	1.3
lcdnum	2,538	2,506	-1.3	20,533	8,506	-58.6	3.0	1.3	-55.3	0.0	0.0	-10.5	3.3	1.7
loop3	53,986	53,879	-0.2	8,380	5,942	-29.1	4.2	3.6	-15.5	0.1	0.1	0.0	7.2	6.5
minmax	1,987	1,898	-4.5	2,925	1,106	-62.2	1.7	0.8	-51.4	0.0	0.0	8.3	1.9	1.1
qurt	26,363	21,742	-17.5	98,859	25,459	-74.2	60.5	13.5	-77.6	0.1	0.1	27.5	61.5	14.5
sqrt	7,120	5,576	-21.7	26,415	7,768	-70.6	14.6	3.2	-77.9	0.0	0.0	0.0	15.1	3.7
<average>			-9.4			-52.6			-55.8			2.2		
<weighted avg.>			-10.7			-96.9			-95.7			2.2		

References

1 COIN-OR Linear Programming: `http://www.coin-or.org/Clp`.

2 AbsInt Angewandte Informatik GmbH. *AbsInt Advanced Analyzer for PowerPC MPC7448 (Simple Memory Model): User Documentation*.

3 Claire Burguière and Christine Rochange. On the complexity of modeling dynamic branch predictors when computing worst-case execution time. In *Proceedings of the ERCIM/DECOS Workshop On Dependable Embedded Systems*, August 2007.

4 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 238–252, New York, NY, USA, 1977. ACM Press.

5 Christoph Cullmann. Cache persistence analysis: a novel approach theory and practice. In Jan Vitek and Bjorn De Sutter, editors, *LCTES*, pages 121–130. ACM, 2011.

6 Stephen A. Edwards and Edward A. Lee. The case for the precision timed (PRET) machine. In *DAC*, pages 264–265. IEEE, 2007.

7 Jakob Engblom. *Processor Pipelines and Static Worst-Case Execution Time Analysis*. PhD thesis, Dept. of Information Technology, Uppsala University, 2002.

8 Andreas Ermedahl and Jan Gustafsson. Deriving annotations for tight calculation of execution time. In *Euro-Par*, pages 1298–1307, 1997.

9 C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In *International Conference on Embedded Software*, volume 2211 of *LNCS*, pages 469–485, 2001.

10 Christian Ferdinand and Reinhard Wilhelm. Efficient and precise cache behavior prediction for real-time systems. *Real-Time Sys.*, 17(2-3):131–181, 1999.

11 Freescale Semiconductor. *MPC7450 RISC Microprocessor Family Reference Manual*.

12 Daniel Grund, Jan Reineke, and Gernot Gebhard. Branch target buffers: WCET analysis framework and timing predictability. *Journal of Systems Architecture*, 57(6):625–637, 2011.

13 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET benchmarks – past, present and future. pages 137–147, Brussels, Belgium, July 2010. OCG.

14 C. Healy, M. Sjödin, V. Rustagi, D. Whalley, and R. van Engelen. Supporting timing analysis by automatic bounding of loop iterations. *Real-Time Sys.*, pages 129–156, 2000.

15 Reinhold Heckmann, Marc Langenbach, Stephan Thesing, and Reinhard Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7):1038–1054, 2003.

16 Benedikt Huber, Wolfgang Puffitsch, and Martin Schoeberl. Worst-case execution time analysis-driven object cache design. *Concurrency and Computation: Practice and Experience*, 24(8):753–771, 2012.

17 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*, pages 456–461, 1995.

18 Jan Reineke, Björn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. A definition and classification of timing anomalies. In *Proceedings of 6th International Workshop on Worst-Case Execution Time (WCET) Analysis*, July 2006.

19 Martin Schoeberl. A java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54(1-2):265 – 286, 2008.

20 Ingmar Stein and Florian Martin. Analysis of path exclusion at the machine code level. In *Proceedings of the 7th Intl. Workshop on Worst-Case Execution-Time Analysis*, 2007.

21 Henrik Theiling. *Control-Flow Graphs For Real-Time Systems Analysis*. PhD thesis, Saarland University, Saarbrücken, Germany, 2002.

22 Henrik Theiling. ILP-based interprocedural path analysis. In *International Conference on Embedded Software*, volume 2491 of *LNCS*, pages 349–363. Springer, 2002.

23 Stephan Thesing. *Safe and Precise WCET Determinations by Abstract Interpretation of Pipeline Models*. PhD thesis, Saarland University, Saarbrücken, Germany, 2004.

24 Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, and Christian Ferdinand. An abstract interpretation-based timing validation of hard real-time avionics software systems. In *Proceedings of the 2003 Intl. Conference on Dependable Systems and Networks*, pages 625–632. IEEE Computer Society, 2003.

25 Theo Ungerer, Francisco J. Cazorla, Pascal Sainrat, Guillem Bernat, Zlatko Petrov, Christine Rochange, Eduardo Quiñones, Mike Gerdes, Marco Paolieri, Julian Wolf, Hugues Cassé, Sascha Uhrig, Irakli Guliashvili, Michael Houston, Florian Kluge, Stefan Metzlaff, and Jörg Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, 2010.

26 Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, and Christian Ferdinand. 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, 2009.