#### Modular WCET Analysis of ARM Processors

Andreas Engelbredt Dalsgaard **Mads Christian Olesen** Martin Toft René Rydhof Hansen Kim Guldstrand Larsen

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
●○○	000	0000	000000	O	00
The Prol	olem				

#### Problem

Given a program in executable form, for an ARM9 processor, determine a safe and tight worst-case execution time (WCET)

Goals:

- $\bullet\,$  Model the pipeline and cache(s) of the ARM9 in a precise manner
- Make the model modular, such that other ARM9 processors can easily be modelled

Introduction	Challenges	<b>Tool-Chain</b>	Value Analysis	Demo	Conclusion
○●○	000	0000	000000	0	
Real-Time	Systems				

- Real-time systems (RTS) are systems that need to respond to real-life events in a timely manner
- A number of processes with associated WCETs and deadlines
- Tasks are periodic or sporadic



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
○○●	000	0000	000000	0	00

#### WCET Distribution



- Estimates should be on the safe side!
- $\bullet\,$  However, too much on the safe side  $\Rightarrow$  inefficient system

 Introduction
 Challenges
 Tool-Chain
 Value Analysis
 Demo
 Conclusion

 000
 ●00
 0000
 000000
 0
 00

#### Challenge I: Modern Processors

Modern processors optimise for the average case, using:

Caching: allowing quick access to recently used memory items Pipelining: executing instructions in parallel



Size: ~128 Mb Access-time: ~33 cycles

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	○●○	0000	000000	0	00
Can we be	ignorant?				

#### No!

- Some processors have "timing anomalies", i.e. local worst-case ≠ global worst-case
- Even without "timing anomalies" assuming the local worst-case can give an over-approximation by a **factor 30**

The ARM9 processor does not exhibit "timing anomalies"

- Quicker analysis, less overapproximation
- Processors without "timing anomalies" are sufficient for most real-time systems

Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000				

## Challenge II: Making the Analysis Modular

A modular analysis allows **more flexibility**, e.g. how would this program perform:

- ... if the cache was larger?
- ... with an extra processor core?
- ... on an entirely different processor?

#### And different accuracy/performance tradeoffs:

- Abstract interpretation for (abstract) cache analysis
- Model checking for (concrete) cache analysis
- Use simple always-miss cache, if no need to do more precise analysis

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	●○○○	000000	0	



Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
	0000			

## Overview of Our Model



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	○○●○	000000	O	00
Path Ana	alvsis				

- Timed automaton for every function
- Transitions emulate instruction execution



• Functions handled flow-sensitively

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	○○○●	000000	0	
Cache Ar	nalysis				

- ARM9: Separate data and instruction caches
  - 16 kB in size, 64-way associative, 8 words (32 byte) per line
  - Write-through and write-back policies
  - Pseudo-random and round-robin replacement policies
- Modelled concretely as timed automata in UPPAAL



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	●○○○○○	0	00
	alveic				

- The cache analysis needs concrete memory addresses
- Registers are used as base and offset in all memory accesses
- Value analysis:



- Find an over-approximation of possible register values at all execution points of a process
- Weighted push-down systems (WPDSs) used for inter-procedural, control-flow sensitive value analysis
- Presented by Reps et al. in **Program Analysis using Weighted Push-Down Systems**

Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
		00000		

## Weighted Push-Down Systems

Use the PDS-stack as call-stack:

• Sequential: 
$$\langle p, n_{main} \rangle \hookrightarrow \langle p, n_2 \rangle$$

- Function call:  $\langle p, n_4 \rangle \hookrightarrow \langle p, n_8 n_5 \rangle$
- Function return:  $\langle p, n_{12} \rangle \hookrightarrow \langle p, \epsilon \rangle$

Each rule has an associated weight, describing the effect of the transition. Weights can be:

- Combined ("join"):  $w_1 \oplus w_2 = w_3$
- Extended (sequential progression):  $w_1 \otimes w_2 = w_3$

The effect of executing a program to a set of configurations (T) ("Meet over all paths"):

 $\bigoplus \{w_1 \otimes \ldots \otimes w_n | w_1, \ldots, w_n \text{ is the weights associated with a path of rules leading to a configuration in T }$ 

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	○○●○○○	O	
Our Value	Analysis				

Implemented simple value analysis, using:

- Loop unrolling
- Simple (syntactical) register-value tracking
- No tracking of values in memory
- Finds good amount of values for some programs, but could be much better

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	○○○●○○	O	
Our Value	Analysis				

• Weights = functions representing the effect of an instruction or a sequence of instructions, e.g.:

$$\mathbf{w}_{1} \begin{pmatrix} r_{0} \\ r_{1} \end{pmatrix} = \begin{pmatrix} "r_{1} + 2" \\ \mathbf{id} \end{pmatrix}, \quad \mathbf{w}_{2} \begin{pmatrix} r_{0} \\ r_{1} \end{pmatrix} = \begin{pmatrix} \mathbf{id} \\ "r_{0} * 2 + r_{1} <<3" \end{pmatrix}$$

- $\bullet$  Special values: id,  $\perp$  and  $\top$
- Combine and extend handled syntactically (string equality, and string replacement)

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	○○○○●○	0	00

#### Implementation = WALi + Python

- The open source Weighted Automata Library (WALi) implements a number of WPDS algorithms
- Easy to extend with e.g. new weight domains
- Our weights are, very conveniently, valid Python expressions
- Process automata are annotated with the results



Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	○○○○○●	0	

# Disassembler — Dissy

D		Dissy - /home/mchr	o/DAT6/svn/wcet_b	ench/web/	matrr	ult/m	atmult.o		
Eile Navigation Opt	ions <u>T</u> ools	Help							
Lookup			~	Highlight					
Address Size L	abel								
0x00008340 84 F	landomInte	ger							
0x00006394 60 I	nitialize								
0x000083d0 120 M	ultiply								
0x00008448 48 T	est								
					14 14				Information how
Address   b0   b1   b	2 Instructio	n .			10 1	1 12	Target		o fed efil
0x00008394	push	{r4, r5, r6, lr}						Store	Register
0x00008398	mov	r5, r0						Corr	abined A
0x0000839c	mov	r6, #0						Reg	Values (hex.)
0x0000e3a0	> mov	r4, #U						rO	N/A
0x000083a4	ы	8340					Randominteger	r2	N/A N/A
0x00008388	str	r0, [r4, r5]						r3	N/A 0x0 0x4 0x8 0xc 0x10 0x14
0X000083ac	add	r4, r4, #4							0x18, 0x1c, 0x20, 0x24, 0x28.
OXOCOURADO	cmp	r4, #80							0x40, 0x44, 0x48, 0x4c, 0x0,
0x00008364	bne	83a4 ;@Loop_bound 20							0x4, 0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c
0x00008368	add	r6, r6, #1							0x30, 0x34, 0x38, 0x3c, 0x40,
OXOOOOB3DC	cmp	r6, #20							0xc, 0x10, 0x14, 0x18, 0x1c,
OXOCOURSCU	add	r5, r5, #80						r4	0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38, 0x3c, 0x40, 0x44,
0x000083c4	bne	BGaU ;@toop_bound 20							0x48, 0x4c, 0x0, 0x4, 0x8, 0xc, 0x10, 0x14, 0x18, 0x1c, 0x20
0X00008368	pop	(r4, r5, r6, tr)							0x24, 0x28, 0x2c, 0x30, 0x34.
OXOOODBJCC	DX	tr							0x36, 0x36, 0x40, 0x44, 0x48, 0x4c, 0x0, 0x4, 0x8, 0xc, 0x10,
									0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38
									0x3c, 0x40, 0x44, 0x48, 0x4c
								rs	0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
									0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
									0x0, 0x0, 0x1, 0x1, 0x1, 0x1, 0x1,
									0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1, 0x1,
									0×1, 0×1, 0×1, 0×1, 0×2, 0×2,
								-6	0x2, 0x2, 0x2, 0x2, 0x2, 0x2, 0x2, 0x2,
									0x2, 0x2, 0x2, 0x2, 0x2, 0x2, 0x2,
									10x3.0x3.0x3.0x3.0x3.0x3.0x3.

16/20

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	000000	●	00

WCET Guarantee in Three Easy Steps

# Demo

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	000000	O	●○
Experime	ents				

#### • Evaluated on the Mälardalen WCET benchmarks

Introduction 000	Challenges 000	Tool-Chain 0000	Value Analysis 000000	Demo O	Conclusion ●○
Evenering	opto				
Experime	enus				

- Evaluated on the Mälardalen WCET benchmarks
- The most interesting findings:
  - Taking the **instruction cache** into account yields WCETs that are up to 97% sharper (78% on average at -02)
  - Taking the **data cache** into account yields WCETs that are up to 68% sharper (31% on average at -02)
  - Almost all results are obtained within five minutes

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	000000	0	●○
Evnerimer	ite				

- Evaluated on the Mälardalen WCET benchmarks
- The most interesting findings:
  - Taking the **instruction cache** into account yields WCETs that are up to 97% sharper (78% on average at -02)
  - Taking the data cache into account yields WCETs that are up to 68% sharper (31% on average at -02)
  - Almost all results are obtained within five minutes
- Some programs fail due to
  - State space explosion (6)
  - Write to program counter (2)
  - Floating point operations <sup>1</sup>
  - Value analysis problems

<sup>&</sup>lt;sup>1</sup>need to manually find good loop-bounds for very optimised assembler

Introduction	Challenges	Tool-Chain	Value Analysis	Demo	Conclusion
000	000	0000	000000	O	●○
Experime	ents				

- Evaluated on the Mälardalen WCET benchmarks
- The most interesting findings:
  - Taking the **instruction cache** into account yields WCETs that are up to 97% sharper (78% on average at -02)
  - Taking the data cache into account yields WCETs that are up to 68% sharper (31% on average at -02)
  - Almost all results are obtained within five minutes
- Some programs fail due to
  - State space explosion (6)
  - Write to program counter (2)
  - Floating point operations <sup>1</sup>
  - Value analysis problems

# • We are able to analyse 17 out of the 25 non-floating point benchmarks!

<sup>&</sup>lt;sup>1</sup>need to manually find good loop-bounds for very optimised assembler

Introduction	Challenges	<b>Tool-Chain</b>	Value Analysis	Demo	Conclusion
000	000	0000	000000	0	○●
Conclusion	- Future V	Vork			

- Prototype implementation successful
- UPPAAL provides a good framework for modularising the models
- The analysis times seem acceptable
- Better path analysis
- Precise value analysis essential for tight bounds (work in progress)
  - Modelling the stack
  - Modelling memory regions
- Support other typical embedded processors:
  - ARM7 (3-stage pipeline, cache)
  - Atmel AVR 8bit (3-stage pipeline, no cache, 1-2 cycle instructions)
  - H8/300 (old Lego Mindstorms)
- Modelling the cache abstractly



• Our master's thesis, the accompanying source code, and these slides are available at

http://metamoc.martintoft.dk

Thank you for your attention!



- The Problem
- Real-Time Systems
- WCET Distribution

#### 2 Challenges

- Challenge I: Modern Processors
- Can we be ignorant?
- Challenge II: Making the Analysis Modular

## 3 Tool-Chain

- Tool-Chain Overview
- Overview of Our Model
- Path Analysis
- Cache Analysis
- Value Analysis
  - Value Analysis
  - Weighted Push-Down Systems
  - Our Value Analysis

- Implementation = WALi + Python
- Disassembler Dissy

#### 5 Demo

• WCET Guarantee in Three Easy Steps

#### 6 Conclusion

- Experiments
- Conclusion Future Work