On the Use of Context Information for Precise Measurement-Based Execution Time Estimation

Stefan Stattelmann

FZI Forschungszentrum Informatik
Karlsruhe

Florian Martin

AbsInt Angewandte Informatik GmbH
Saarbrücken

Introduction

- Nontrivial interaction of performance enhancing features
  - Caches
  - Speculative Execution
- Execution time depends on execution history

- Static WCET analysis
  - Based on safe/pessimistic abstract models
  - Incorporating context information increases precision

- Dynamic WCET analysis
  - How to get accurate traces *on the real hardware*?
  - How to represent the execution history of measurements?

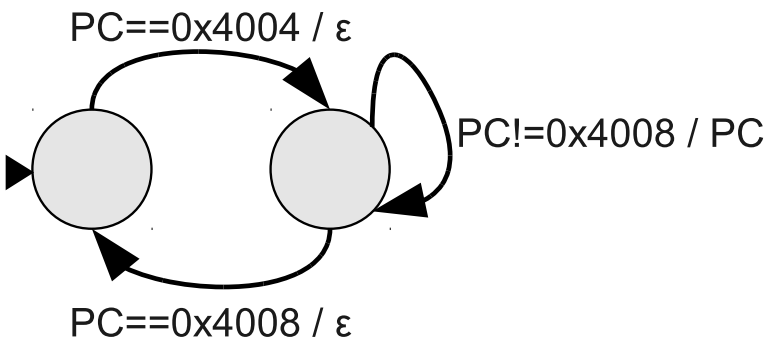
Trace Data Generation

- Off-Chip Trace Memory (e.g. Nexus)
  - Large off-chip memory allows very long traces
  - Executed instructions can be missing in trace
  - Timestamps created upon transport to off-chip buffer
- On-Chip Trace Memory with Programmable Event Logic
  - Vendors: ARM (CoreSight), Infineon (Multi-Core Debug Solution)
  - Cycle-accurate traces with complex trigger conditions
  - Programmable triggers with access to processor state
  - Limited trace memory $\Rightarrow$ no complete program traces

Trace Data Generation

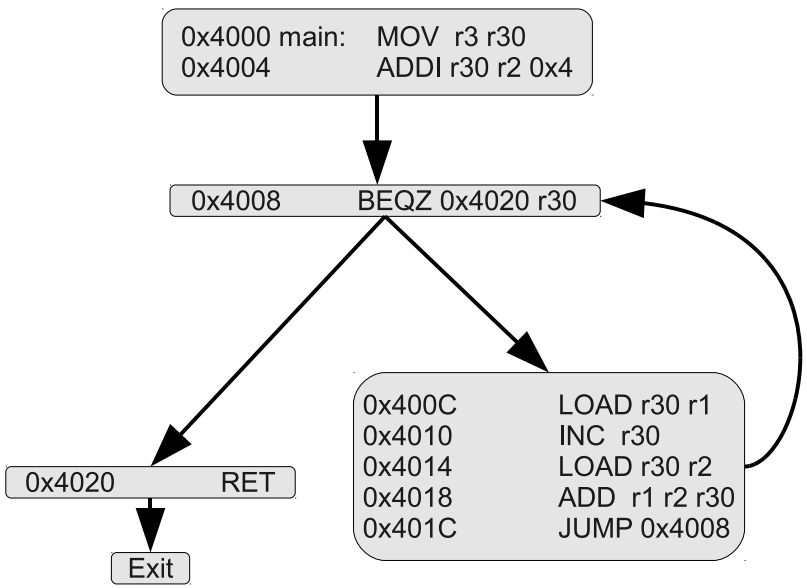
- Off-Chip Trace Memory (e.g. Nexus)
  - Large off-chip memory allows very long traces
  - Executed instructions can be missing in trace
  - Timestamps created upon transport to off-chip buffer

- On-Chip Trace Memory with Programmable Event Logic
  - Vendors: ARM (CoreSight), Infineon (Multi-Core Debug Solution)
  - Cycle-accurate traces with complex trigger conditions
  - Programmable triggers with access to processor state
  - Limited trace memory $\Rightarrow$ no complete program traces
  - Idea: Use state machines to encode execution history of traces



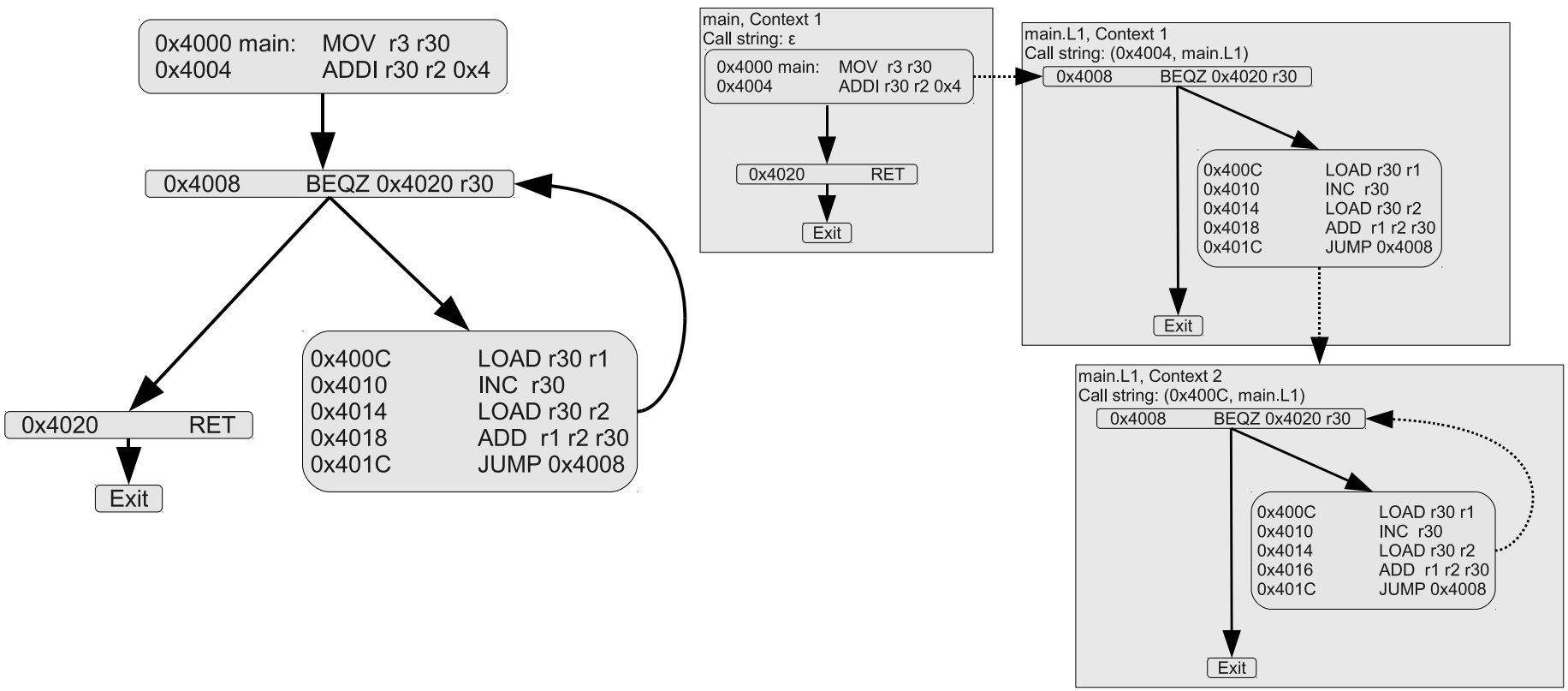
Context Representation

- Control flow extracted from binary executable
- Transform loops into recursive routines
- Extend interprocedural control flow graph by "duplicating" nodes: virtual inlining, virtual unrolling (VIVU)
- Execution history for different instances of a basic block can be described by a call string

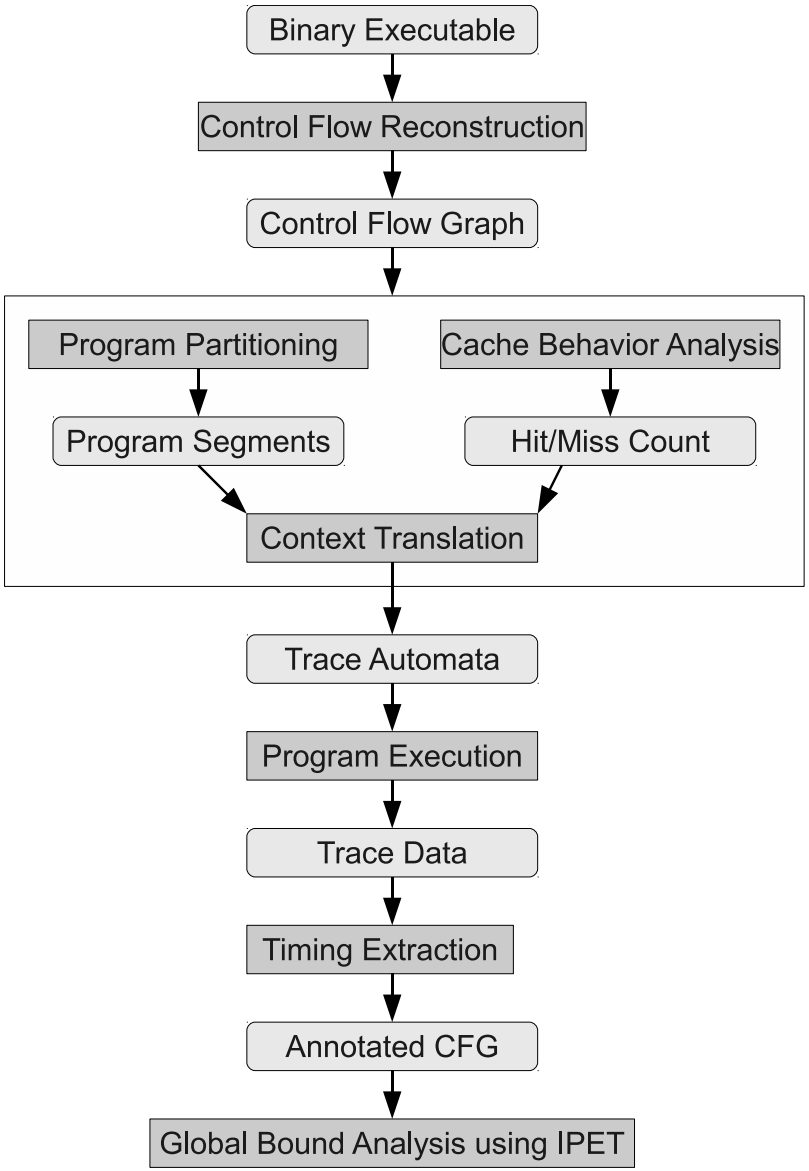


Context Representation

- Control flow extracted from binary executable
- Transform loops into recursive routines
- Extend interprocedural control flow graph by "duplicating" nodes: virtual inlining, virtual unrolling (VIVU)
- Execution history for different instances of a basic block can be described by a call string

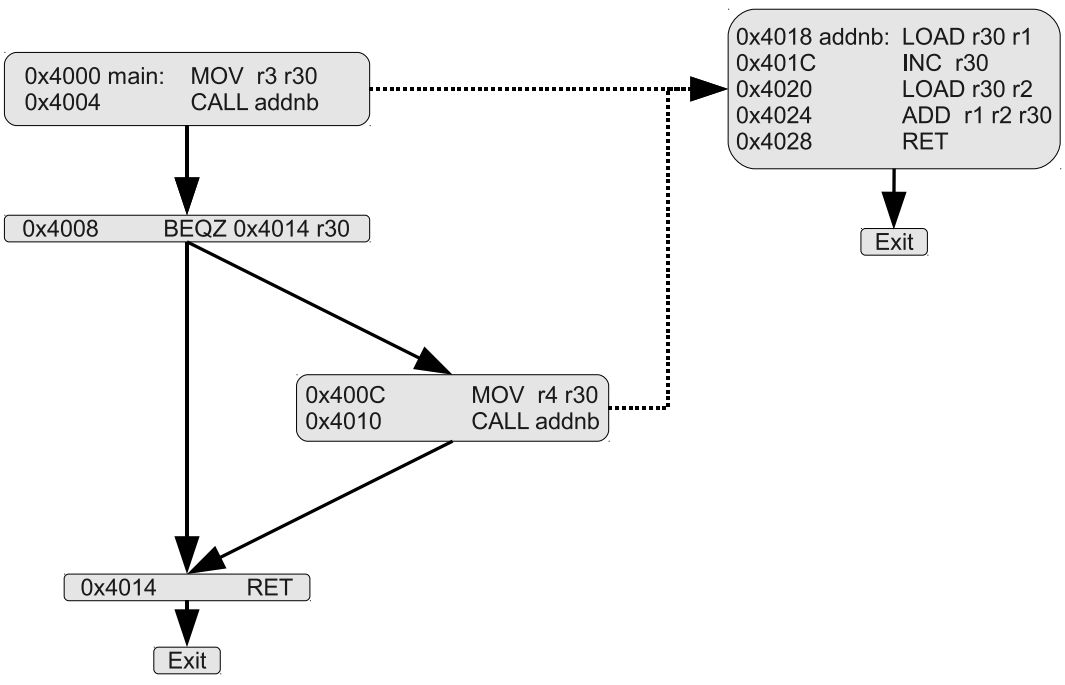


Proposed Tracing Method

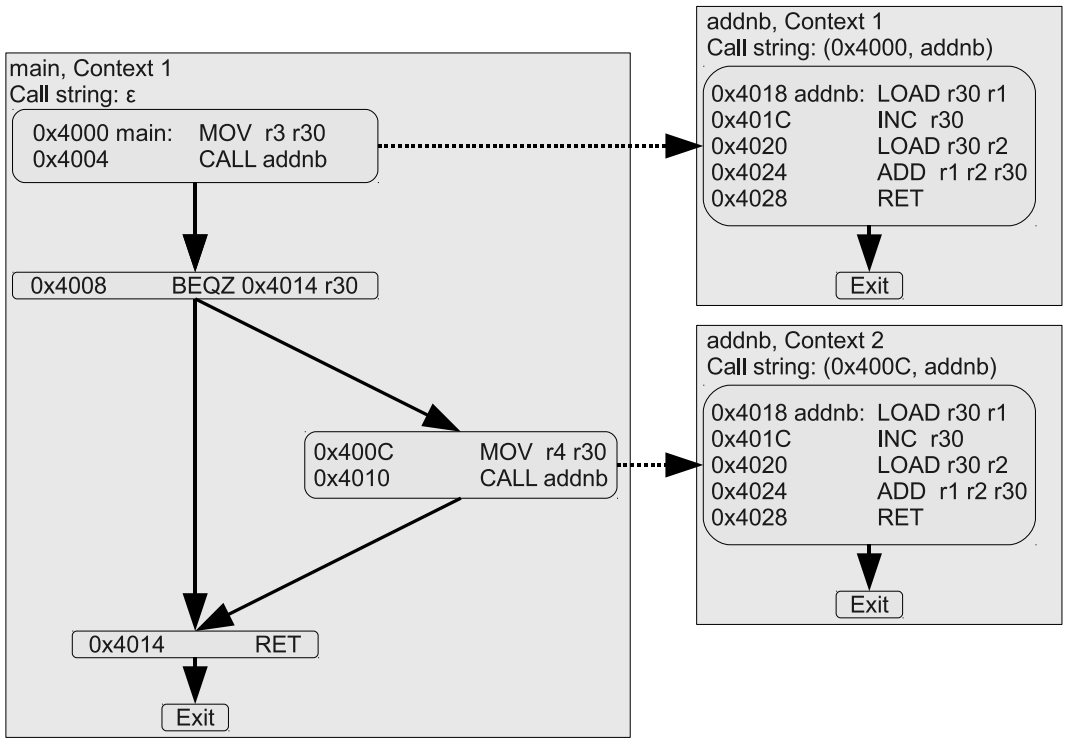


- Program Partitioning
  - Divide CFG into traceable segments
- Cache Behavior Analysis
  - Estimate cache hits and misses
- Context Translation
  - Trigger conditions per segment
  - Merge contexts with similar cache behavior $\Rightarrow$ reduce #measurements
- Program Execution
  - Measure often to cover local worst-case
  - Preserve context information
- Timing Extraction
  - Process traces
  - Annotate execution times to CFG

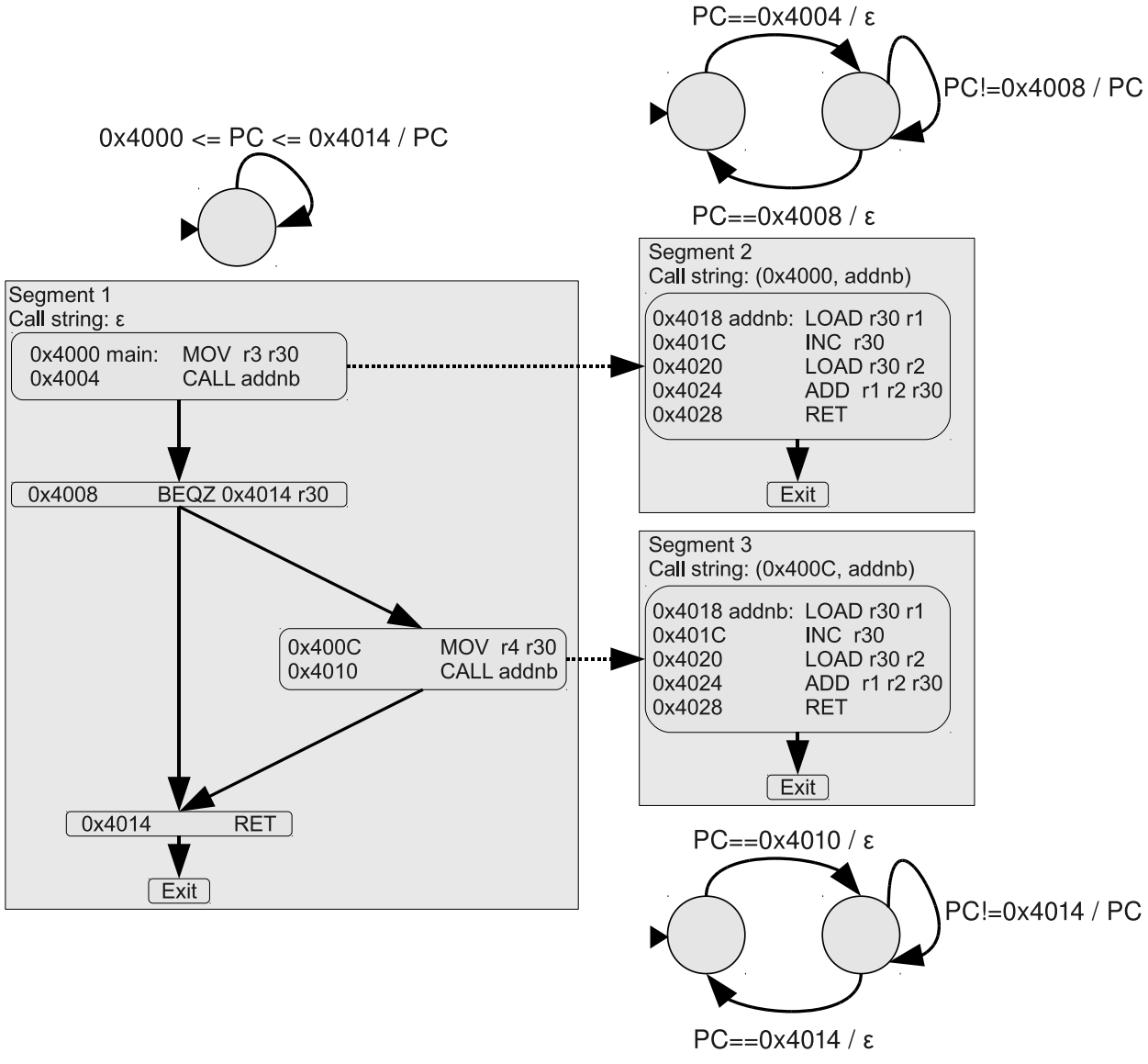
Example


0x4000 main: MOV r3 r30
0x4004 CALL addnb
0x4008 BEQZ 0x4014 r30
0x400C MOV r4 r30
0x4010 CALL addnb
0x4014 RET
Exit
0x4018 addnb: LOAD r30 r1
0x401C INC r30
0x4020 LOAD r30 r2
0x4024 ADD r1 r2 r30
0x4028 RET
Exit

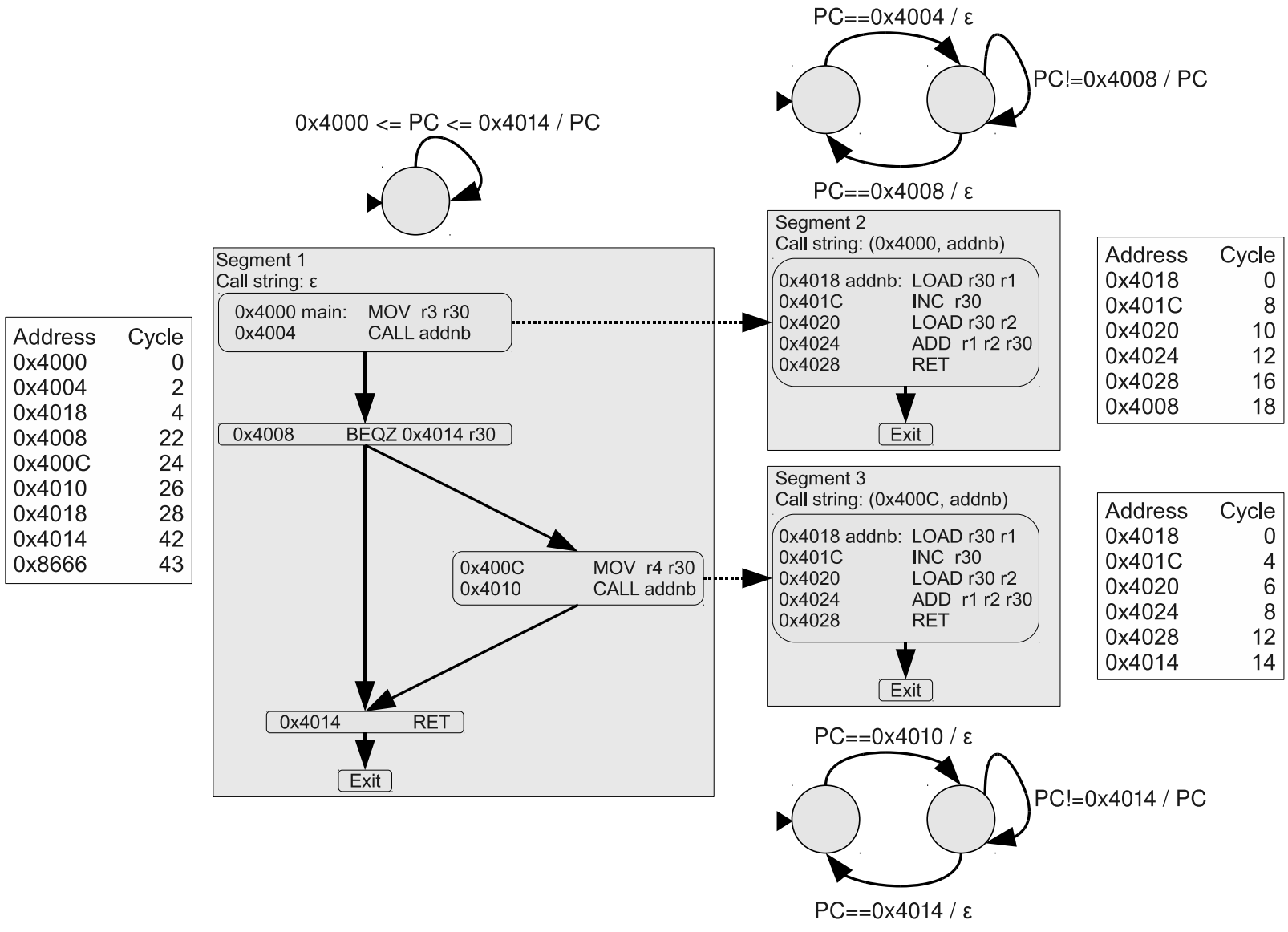
Example


main, Context 1
Call string: ε
0x4000 main: MOV r3 r30
0x4004 CALL addnb
0x4008 BEQZ 0x4014 r30
0x400C MOV r4 r30
0x4010 CALL addnb
0x4014 RET
Exit
addnb, Context 1
Call string: (0x4000, addnb)
0x4018 addnb: LOAD r30 r1
0x401C INC r30
0x4020 LOAD r30 r2
0x4024 ADD r1 r2 r30
0x4028 RET
Exit
addnb, Context 2
Call string: (0x400C, addnb)
0x4018 addnb: LOAD r30 r1
0x401C INC r30
0x4020 LOAD r30 r2
0x4024 ADD r1 r2 r30
0x4028 RET
Exit

Example


PC==0x4004 / ε
PC!=0x4008 / PC
0x4000 <= PC <= 0x4014 / PC
PC==0x4008 / ε
Segment 1
Call string: ε
0x4000 main: MOV r3 r30
0x4004 CALL addnb
0x4008 BEQZ 0x4014 r30
0x400C MOV r4 r30
0x4010 CALL addnb
0x4014 RET
Exit
Segment 2
Call string: (0x4000, addnb)
0x4018 addnb: LOAD r30 r1
0x401C INC r30
0x4020 LOAD r30 r2
0x4024 ADD r1 r2 r30
0x4028 RET
Exit
Segment 3
Call string: (0x400C, addnb)
0x4018 addnb: LOAD r30 r1
0x401C INC r30
0x4020 LOAD r30 r2
0x4024 ADD r1 r2 r30
0x4028 RET
Exit
PC==0x4010 / ε
PC!=0x4014 / PC
PC==0x4014 / ε

Example



Address	Cycle
0x4000	0
0x4004	2
0x4018	4
0x4008	22
0x400C	24
0x4010	26
0x4018	28
0x4014	42
0x8666	43

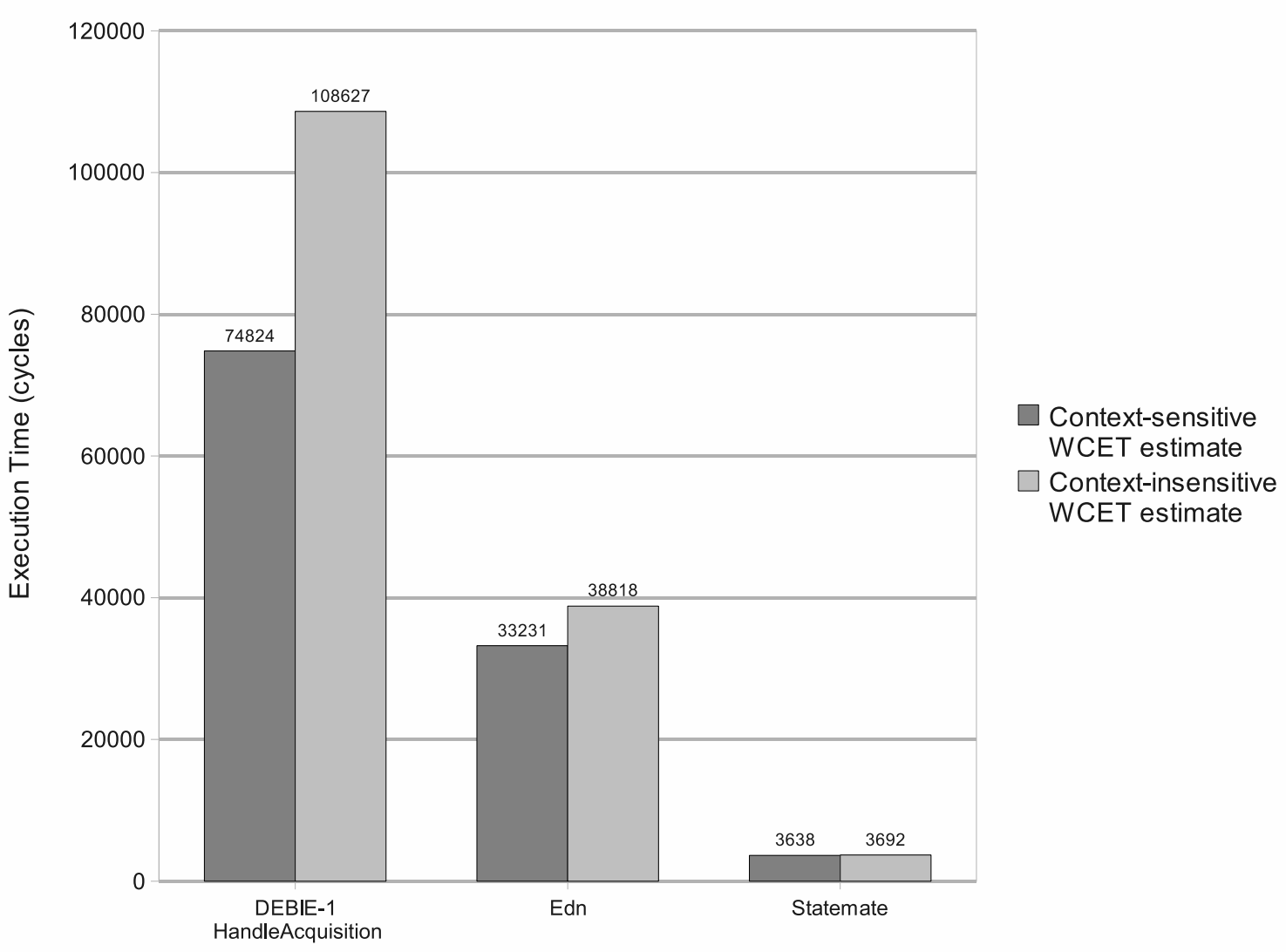
Address	Cycle
0x4018	0
0x401C	8
0x4020	10
0x4024	12
0x4028	16
0x4008	18

Address	Cycle
0x4018	0
0x401C	4
0x4020	6
0x4024	8
0x4028	12
0x4014	14

Experiments

- Integration into the AbsInt aiT WCET analyzer
- Target processor: Infineon TriCore TC1797
- Applied to several embedded applications successfully
  - DEBIE-1 benchmark
  - Examples from Mälardalen WCET benchmark suite
  - MatLab and SCADE generated code
- Cache behavior analysis reduces measurements without affecting WCET estimate
- Contexts can have significant effect on WCET estimates

Results


120000
100000
80000
60000
40000
20000
0
Execution Time (cycles)
108627
74824
38818
33231
3638
3692
DEBIE-1
HandleAcquisition
Edn
Statemate
Context-sensitive
WCET estimate
Context-insensitive
WCET estimate

Conclusion

- Ignoring the execution history can add severe pessimism to measurement-based WCET estimates
  - First iteration of a loop is likely to be slowest one
  - Parameter-dependent execution time of routines
- Context-preserving traces are possible with off-the-shelf hardware
- Future work
  - Parallelize trace generation and timing extraction
  - Reconstruct execution context from arbitrary traces

Thank you for your attention!

Stefan Stattelmann
FZI Research Center for Information Technology, Karlsruhe, Germany

Email: stefan.stattelmann@fzi.de

Florian Martin
AbsInt Angewandte Informatik GmbH, Saarbrücken, Germany

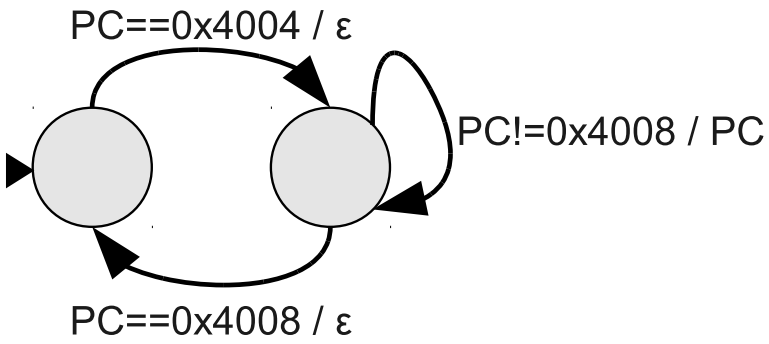
Email: florian.martin@absint.com

Backup Slides

Trace Automata

Call String: $(0x4004, addnb)$

Trace Automaton:



TQL Program:

```
// global configuration
config.memorysize = 0x3ffff;
config.trigger = 0x0;
config.absmode = 0x1;

// define trigger conditions for the TriCore PC
pob_tc.ptu_trig[0].bound = 0xd4004;
pob_tc.ptu_trig[0].range = 0x2;
pob_tc.ptu_trig[1].bound = 0xd4008;
pob_tc.ptu_trig[1].range = 0x2;
pob_tc.tc_act[0] = pob_tc.ptu_trig[0];
pob_tc.tc_act[1] = pob_tc.ptu_trig[1];

// automata states and transitions
mcx.cnt_trig[0].limit = 0x0;
mcx.cnt_trig[0].inc = mcx.tc_act[0];
mcx.cnt_trig[0].clear = mcx.tc_act[1];
mcx.tc_trig[0] = mcx.cnt_trig[0];

// define when to store trace data
pob_tc.ptu_enable[0] = pob_tc.tc_trig[0];
pob_tc.ptu_sync[0] = pob_tc.tc_trig[0];
mcx.trace_done[0] = rise mcx.lmb_act[0];
mcx.tick_enable[0] = true;
```

Cache Behavior Metrics

Goal: detect contexts with similar cache behavior

$$dist(v, w) := \sqrt{(ah_v - ah_w)^2 + (am_v - am_w)^2 + (nc_v + nc_w)^2}$$

$$\delta(s, s') := \{dist(v, w) \mid v \in s,\ w \in s',\ address(v) = address(w)\}$$

Average Distance Metric: $m_{avg}(s, s') := \sum_{d \in \delta(s,s')} \frac{d}{|\delta(s,s')|}$

Maximum Distance Metric: $m_{max}(s, s') := max\{d \mid d \in \delta(s, s')\}$

Cache Behavior Analysis & Context Merging

Program segments s,s' are similar iff

- identical start and end block
- call strings share common suffix
- $m(s, s') < C$

$\Rightarrow$ similar segments can use same trace automaton

$\Rightarrow$ reduces number of measurements