

Precise Simulation of Interrupts using a Rollback Mechanism

Florian Brandner

Vienna University of Technology
Institute of Computer Languages
Christian Doppler Laboratory -
Compilation Techniques for Embedded Systems



Outline

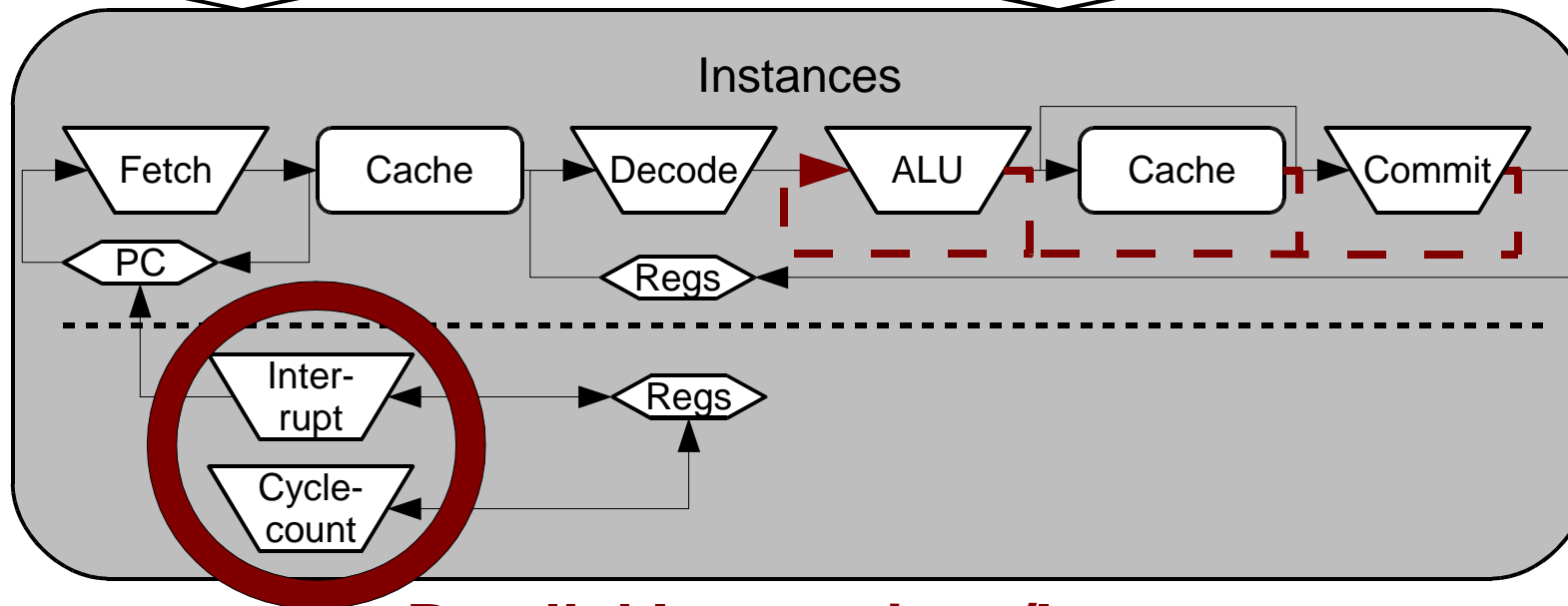
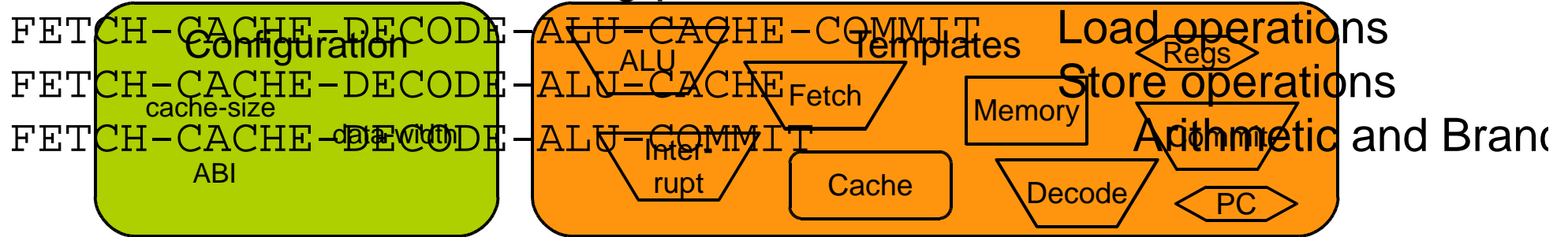
- Introduction
 - Architecture description language
 - Instruction set simulation
- Simulation of interrupts
 - Simple approaches
 - Optimized by a rollback mechanism
- Evaluation
 - Simulation performance
 - Rollback behavior
- Conclusion

Introduction

- Architecture description language
 - Meta-Information
 - Structural processor models
 - Implicit definition of instruction set
 - Parallel instructions/Interrupts
- Generator backends
 - Compiler backend
 - Instruction set simulator
 - XML report
 - VHDL model (prototype)

Example: MIPS Model

Instructions discovered along paths:



Parallel Instructions/Interrupts

Instruction Set Simulation

- Cycle-accurate emulation
- Mixed-mode simulation
 - Simple interpreter
 - Simulate each pipeline stage separately
 - Hot linear code
 - Compiled into basic block functions using LLVM
 - Executed as native code
 - Hot basic blocks
 - Compiled into region functions
 - May contain arbitrary control flow
 - Including loops
- Automatically derived from architecture models

Example: Simulation

BB1:	xor	a0, a1, a0	
	andi	a0, a0, 0xff	WB
	sll	a0, a0, 0x2	MEM
	lui	v0, 0x8003	EX
	beq	v0, BB3	REGION DE
	nop		FE
	srl	a1, a1, 0x8	BB2
BB3:	bne	a1, BB1	
	xor	v0, a1, v0	BB3

• Region Blocks:

- Breaks machine code into blocks
- Simulate complete basic block instructions one by one
- Simulate regions one by one

Simple Interrupt Handling

- Emit code for interrupt check and dispatch on every cycle
- Problems
 - Interrupts are rare events!
 - Increased compile time and code size
 - Due to the interrupt check and dispatch
 - Most of the code is useless though
 - Low simulation speed
 - Interrupt check always needs to be executed
 - Optimizations are impeded by interrupt code

Simple Interrupt Handling

Cycle Simulation-Action

N (a) increment time

Missed
Optimizations

(b) simulate instruction

(c) if (interrupt) jump exit

N+1 (a) increment time

(b) simulate instruction

(c) if (interrupt) jump exit

Rarely taken

...

exit: store state

- Cycle-accurate interrupt simulation is thus avoided
 - Instead only done at certain points
 - Example: basic block boundaries

Using Rollbacks

- Assume interrupts do not occur while executing compiled code
 - On exit verify this assumption
 - Rollback to previous *restore-point* when interrupt missed
 - Restart simulation using interpreter
 - Interpreter models interrupts faithfully
- Advantages
 - Reduced code size
 - Reduced compilation time
 - Improved simulation speed

Using Rollbacks (2)

Cycle Simulation-Action

N simulate instruction

N+1 simulate instruction

N+2 simulate instruction

...

N+n simulate instruction

increment time by n

if (interrupt-missed) jump rollback

exit: store state

rollback: revert state

Optimizations possible

Still rarely taken

Restore-point

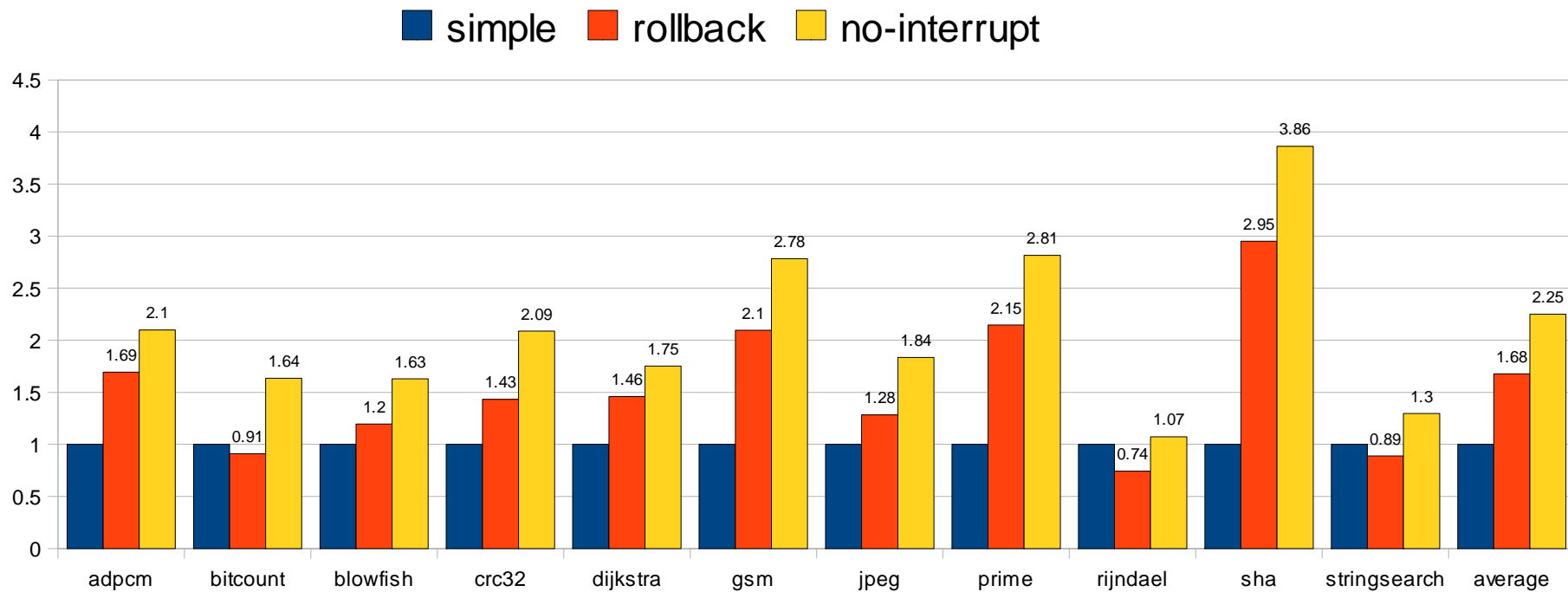
Implementing Rollbacks

- Shadow architecture state in local variables
 - Restore-points
 - Program point that last updated the global state
 - Usually at end of compiled code
 - Special care within regions
 - On rollback simply discard values and return
 - Otherwise copy the local to the global state
- Memories can not be shadowed!
 - Establish restore-points around store operations

Evaluation

- 3GHz Xeon CPU, 24 GB RAM, Linux 2.6.18
- Large subset of MiBench
 - Compiled using GCC 4.2 for mips-elf target
 - With standard optimizations
- 3 Configurations of MIPS-r2000
 - Simple approach (check for interrupts on every cycle)
 - Optimized using rollback mechanism
 - No interrupt support (no overhead)
- Periodic interrupt every 20,000 cycles
- Initially determine optimal thresholds for each configuration

Relative Speedup



Speedups up to 2.95x, only two cases show slowdowns!
Similar results for code size and compilation time.

Cycles reverted per rollback

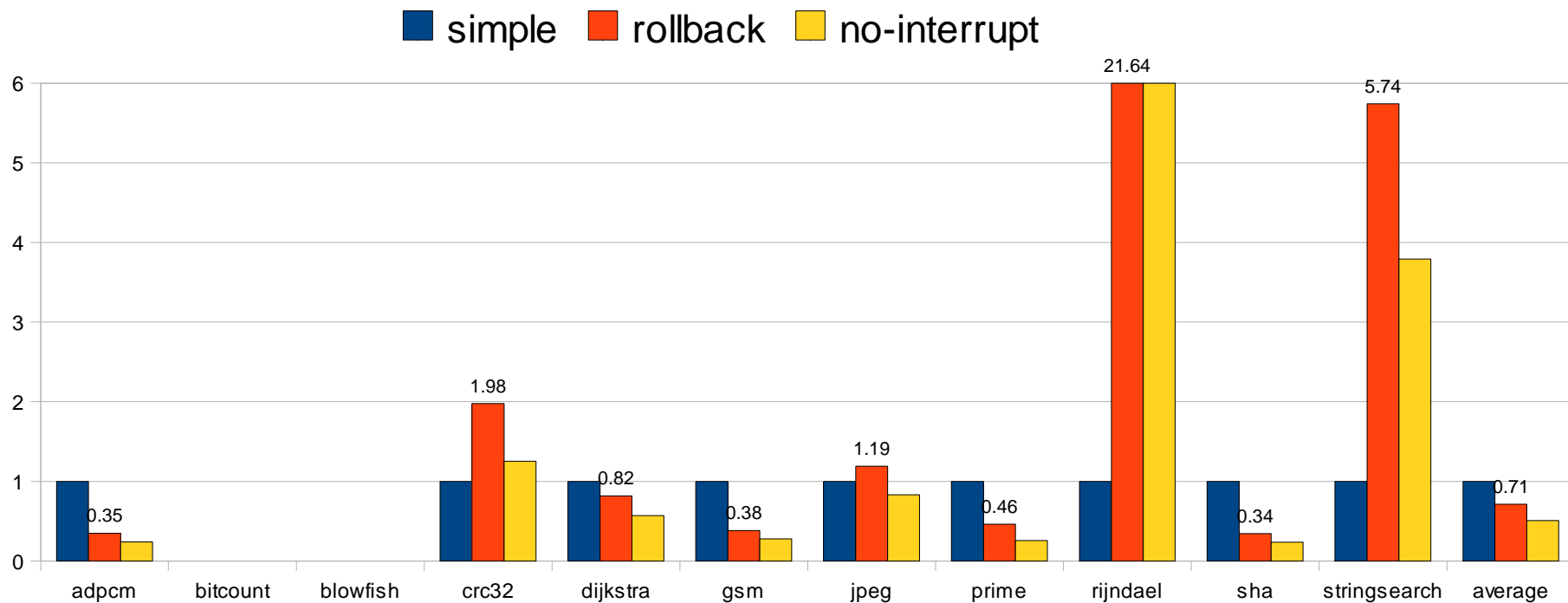
benchmark	avg	min	max	Std. dev.
adpcm	5	1	11	2.74
bitcount	--	--	--	--
blowfish	396	1	422	100.41
crc32	8	1	12	3.7
dijkstra	7	1	16	4.17
gsm	275	1	477	228.64
jpeg	17	1	160	27.3
prime	4	2	5	1.33
rijndael	185	1	538	235.47
sha	19	1	32	7.21
stringsearch	4	1	9	1.84

The number of cycles reverted by a rollback is small. The fraction of reverted cycles is very low and never exceeds 1.67%.

Conclusion

- Rollbacks are very efficient for interrupt simulation
 - Reduced code size
 - Reduced compilation time
 - Even for lower thresholds
 - Improved simulation speed, up to 2.95x
 - Due to reduced overhead
 - More effective optimizations
- May also be beneficial for other rare events
 - Examples: Branch predictors, Caches

Compilation Time



Compile time reduced to 34% in the best case, despite lower compilation thresholds.