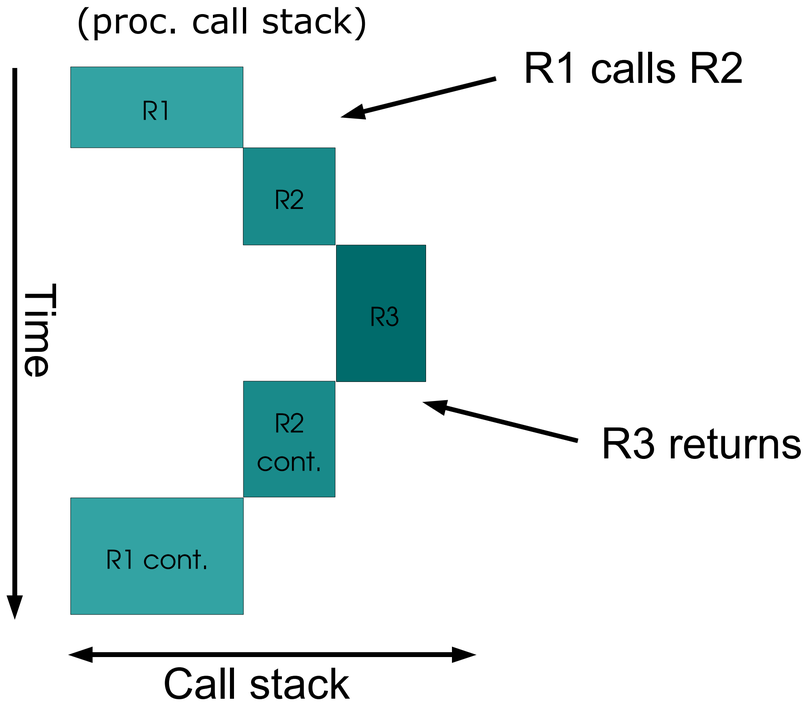
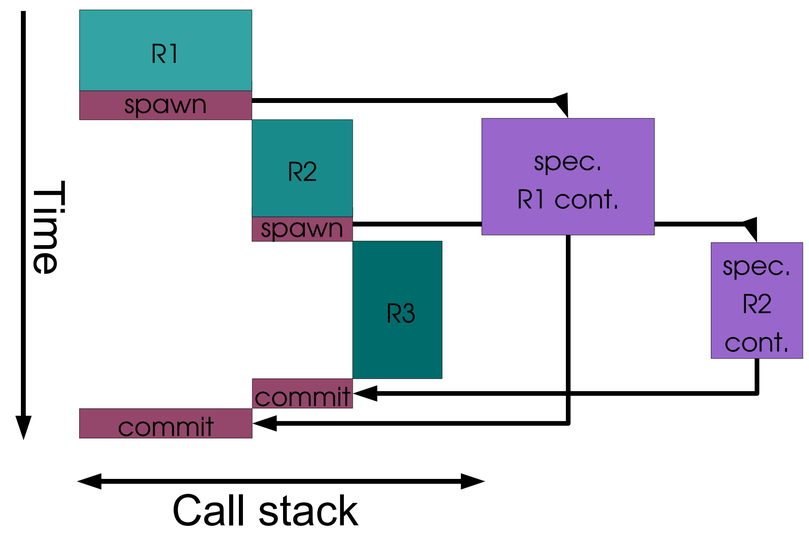
Virtualization Support for the Execution of Speculative Threads

Mathias Payer
Laboratory for Software Technology, ETH Zürich

Normal execution:
(proc. call stack)



Speculative execution:
(on multiple cores)



Motivation

- Not all available resources are used (SMP, CMP, SMT)
- Limits of Instruction Level Parallelism and automatic (non-speculative) parallelization reached
- Call for new, speculative optimization techniques

Introduction

- Parts of code executed speculatively
- Code runs separated from normal control flow
  - Abort speculation on exceptions and synchronization
- State-changes must be validated for hazards
  - (non spec.) Writer after (spec) Read
  - Write after Write
- Increases Thread Level Parallelism

Speculative method calls

- Promising method calls replaced by spec. spawning points
  - normal thread runs the method call
  - speculative thread runs speculatively from return point on
- Return value prediction in the spec. section
- Spec. thread stops as normal thread returns from method

Possible problems / challenges

- Low overhead for non-speculative thread
  - Move as much work as possible to speculative thread
  - Keep synchronization (notify) overhead low
- Memory allocation (low overhead needed)
  - Speculation should not increase memory pressure
- Integration into Opt-Compiler and collaboration with MMTk

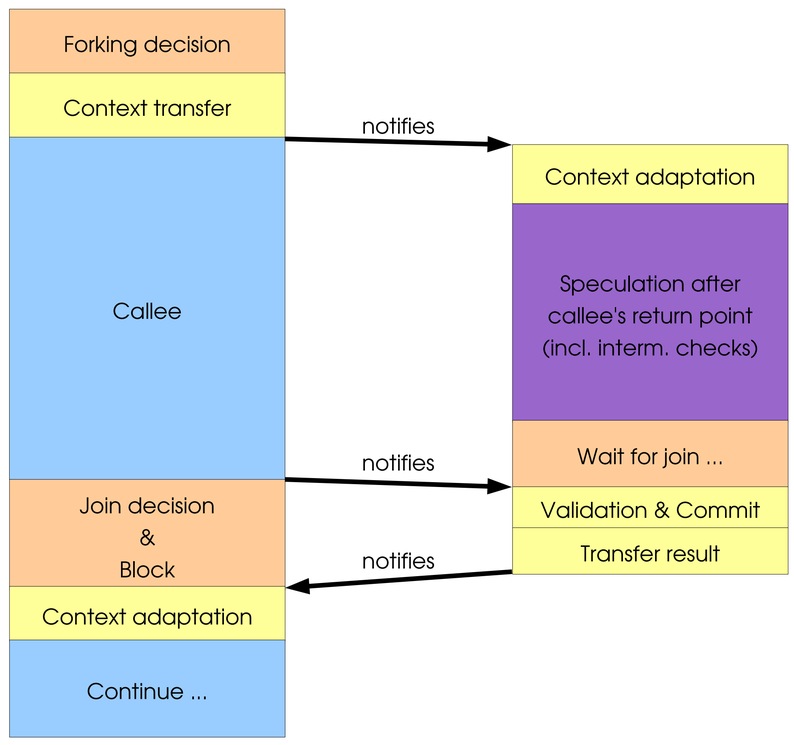
Implementation

- Prototype in Jikes RVM
  - Opt-Compiler extended to generate speculative code
  - Simple buffer structure for speculative objects
- Analysis using ASM bytecode modification toolkit and Spec Benchmarks
- Target architecture: IA32 (Core 2 Duo)

Related work

- JavaSpMt uses src-src transformations for loop speculation, does not change JVM
- SableSpMt is a speculation systems for interpreting JVMs, uses method level speculation

Speculative control transfer and coordination



First results: Low overhead thread synchronization

- Thread library (pthread)
  - High level abstraction (barriers)
  - High overhead (1810 cycles, σ=436 cycles)
- Custom tailored
  - Based on CAS
  - Overhead
    - 115 cycles (σ=4 cycles) on success
    - 440 cycles (σ=87 cycles) on conflict

Hazard frequency for speculations

	compress	raytrace	javac	mpeg
RAW	0.33530%	4.36%	11.81%	32.42%
WAR	0.33548%	8.50%	22.00%	67.38%
WAW	0.33529%	2.86%	11.26%	31.58%

Upper bound for spec. success rate

Aborts	0.22350%	1.21%	7.52%	25.19%

Quality of prediction and end to end overhead not yet evaluated.

References

B. Alpern, et al., *The Jikes Research Virtual Machine project: Building an open-source research community*, IBM Systems Journal, Vol 44, No 2, 2005.

S. Blackburn, et al., *Oil and Water: High Performance Garbage Collection in Java with MMTk*, ICSE 2004

E. Bruneton, et al., *ASM: a code manipulation tool to implement adaptable systems*, Composants 2002

I. Kazi, et al., *JavaSpMt: A speculative thread pipelining parallelization model for java programs*, IDPS 2000

C. Picket, et al., *SableSpMt: A software framework for analyzing speculative multithreading in java*, PASTE 2005

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Supported, in part, by a gift from Intel Corp.