SableSpMT: A Software Framework for Analysing Speculative Multithreading in Java

Christopher J.F. Pickett and Clark Verbrugge
School of Computer Science, McGill University
Montréal, Québec, Canada H3A 2A7
{cpicke,clump}@sable.mcgill.ca

September 6th, 2005

Outline

Motivation

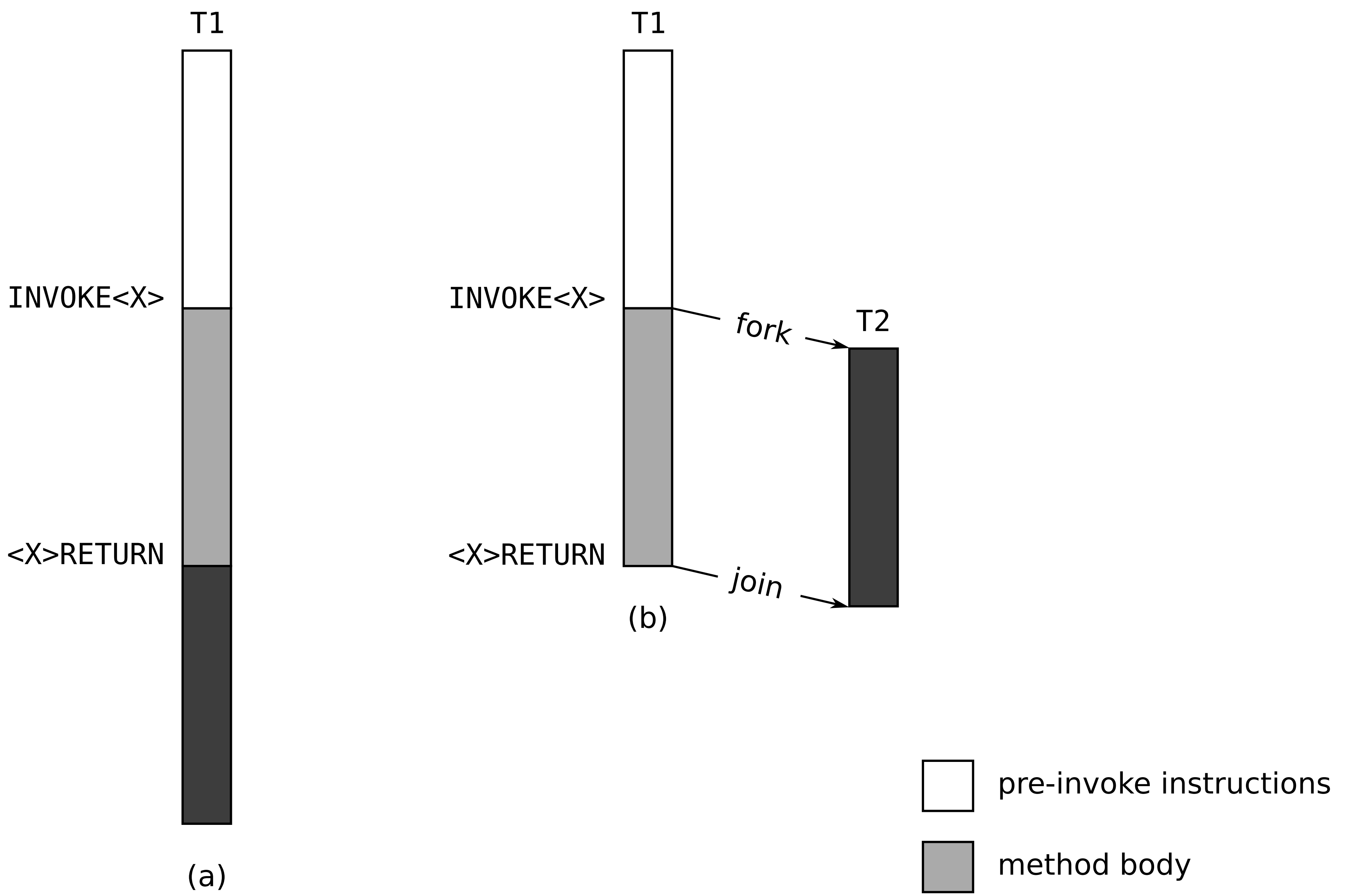
- Speculative Multithreading (SpMT) is a dynamic parallelisation technique that shows good potential speedup.
- Current status: SpMT hardware does not exist, and software SpMT has focused on loops in numeric programs.
  - How do we know what features to incorporate?
  - Can generic SpMT be done entirely in software?
  - Is it really worth building this hardware?
- Many different studies, with many variables:
  - Source language, thread partitioning scheme, compiler framework, hardware simulator, simulation parameters, software architecture.
- Difficult to analyse and compare proposals.

Contributions

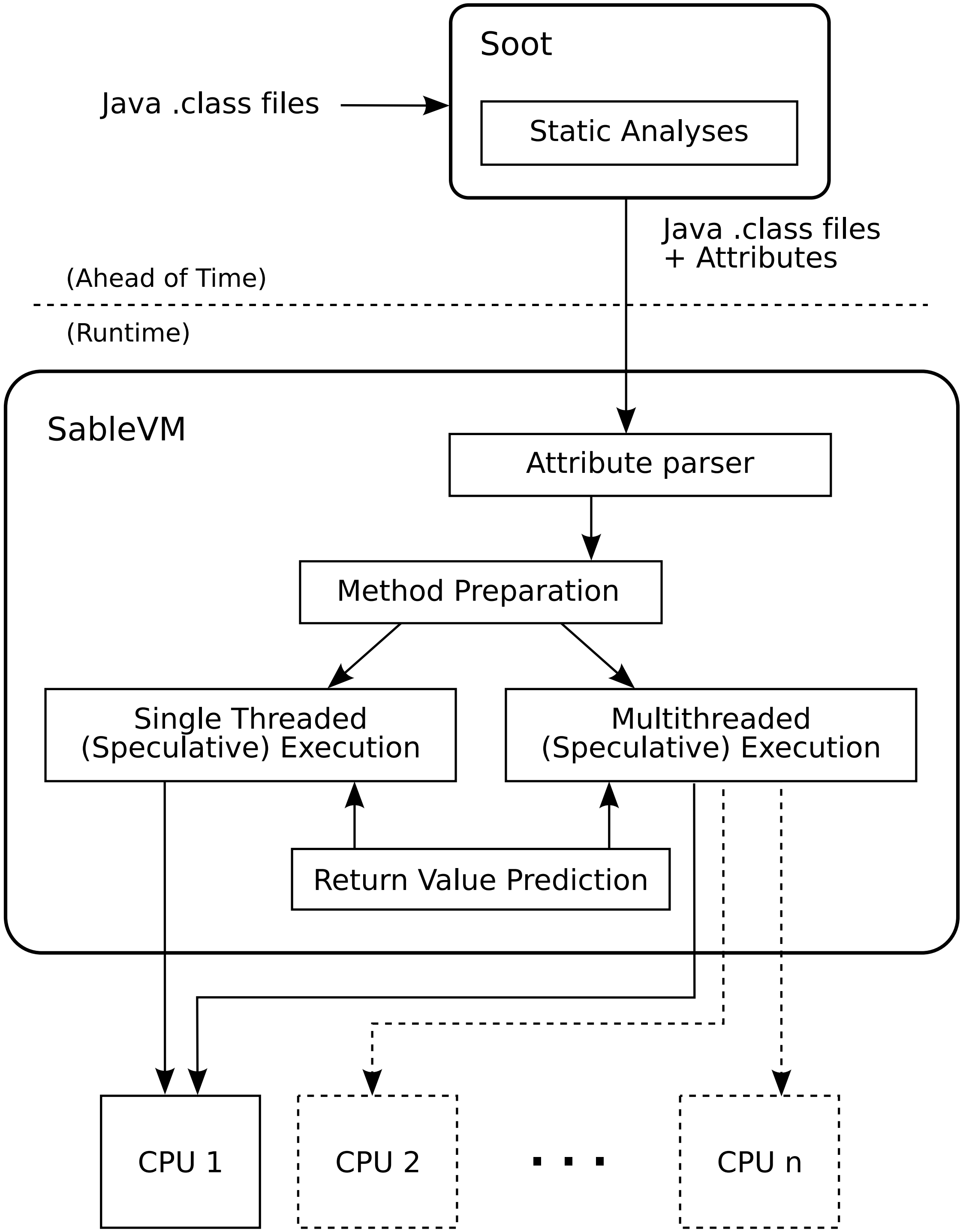
- SableSpMT: software SpMT implementation in JVM
  - Runs on real multiprocessors
  - Suitable as an analysis framework
  - First complete such work, handles SPECjvm98 at S100
- Provide several debugging and analysis features.
- Demonstrate exploitation of static and dynamic info.
- Runtime evaluation:
  - Overhead costs
  - Two parallelism metrics
  - Performance

Outline

Speculative Method Level Parallelism (SMLP)



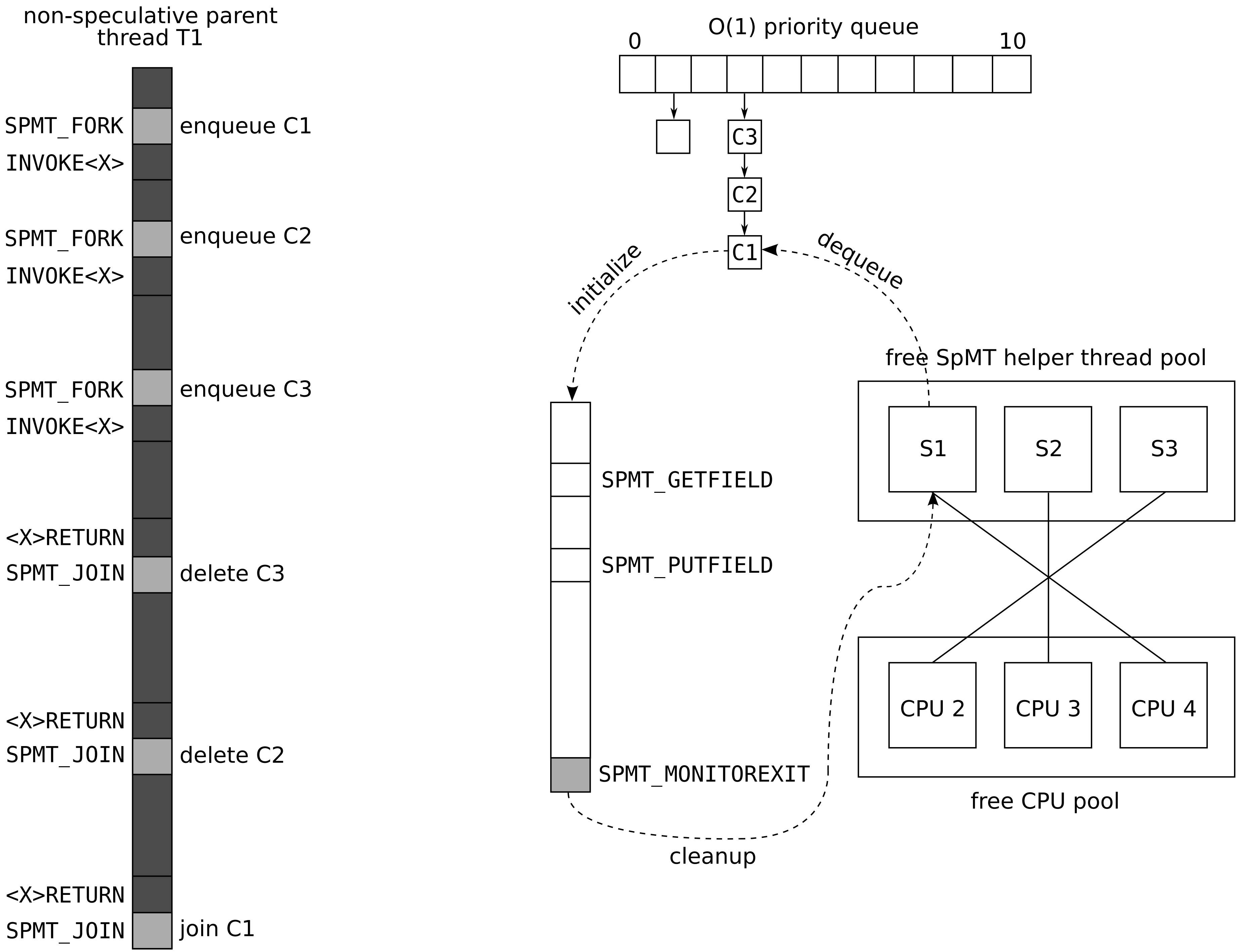
SableSpMT Execution Environment



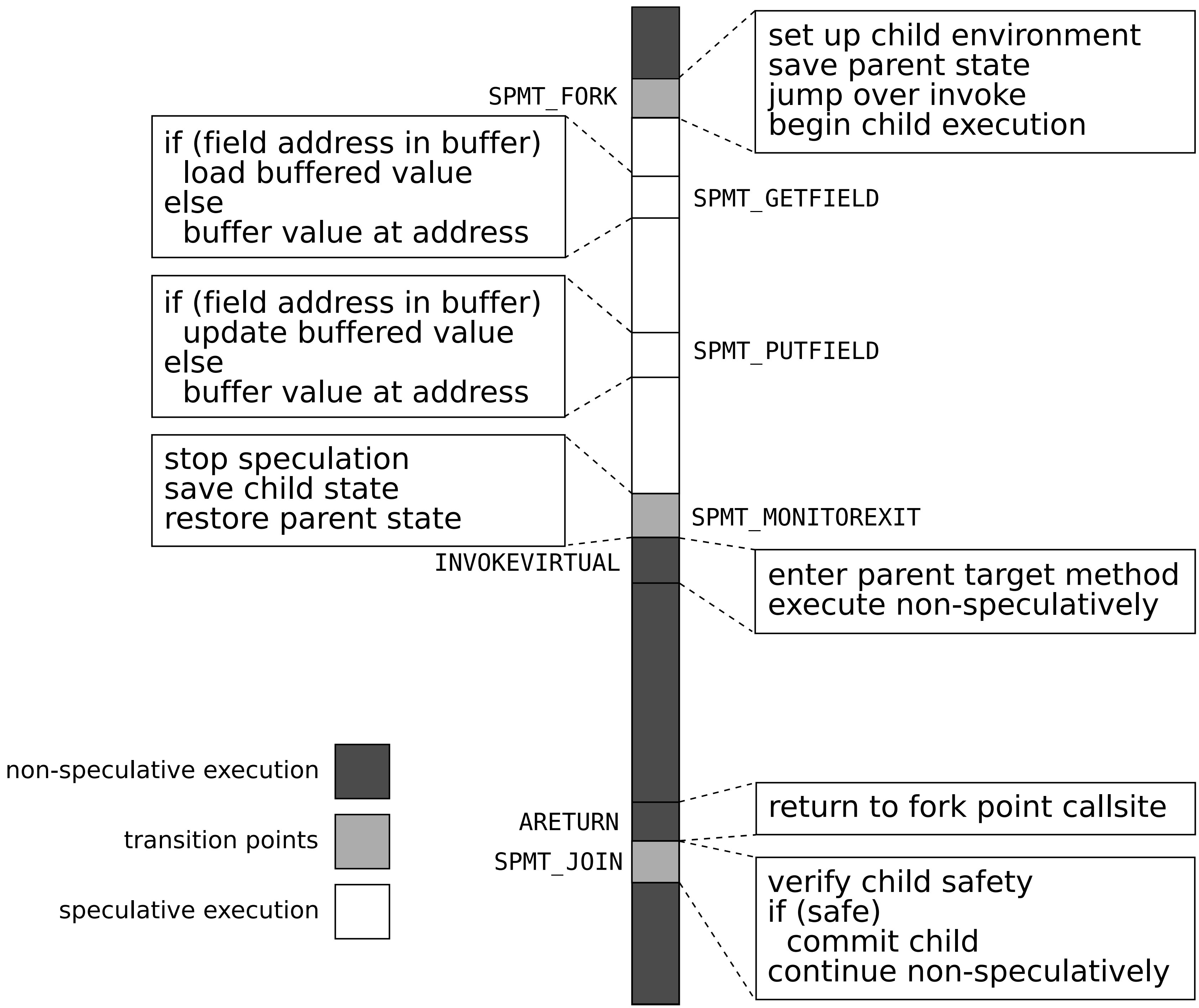
SpMT Execution Components

- Numerous software SpMT components needed:
  - Dependence buffer
  - Stack buffer
  - Return value predictors
  - Helper threads
  - Priority queue
  - Modified bytecodes
- Interaction with existing VM services:
  - Class loading
  - Object allocation
  - Garbage collection
  - Exception handling
  - Native methods
  - Synchronization
  - Java memory model

Multithreaded Mode



Single-threaded Simulation Mode



Outline

Example Component Analysis: RVP

- Framework components:
  - Analyse individually and in detail
  - Instrument and extend to accomodate new analyses
- *Return value prediction* (RVP) is critical for SMLP.
- We implemented software versions of many hardware predictors.
  - Existing stride, context predictors in hybrid: 72% accuracy
  - New memoization predictor added to hybrid: 81% accuracy
- Many RVP configuration properties can be varied: e.g. per-callsite (min, max) hashtable sizes, load factors, enabled predictors.
- Easy to introduce new predictors.

Example Component Analysis: RVP

Two neat analysis results:

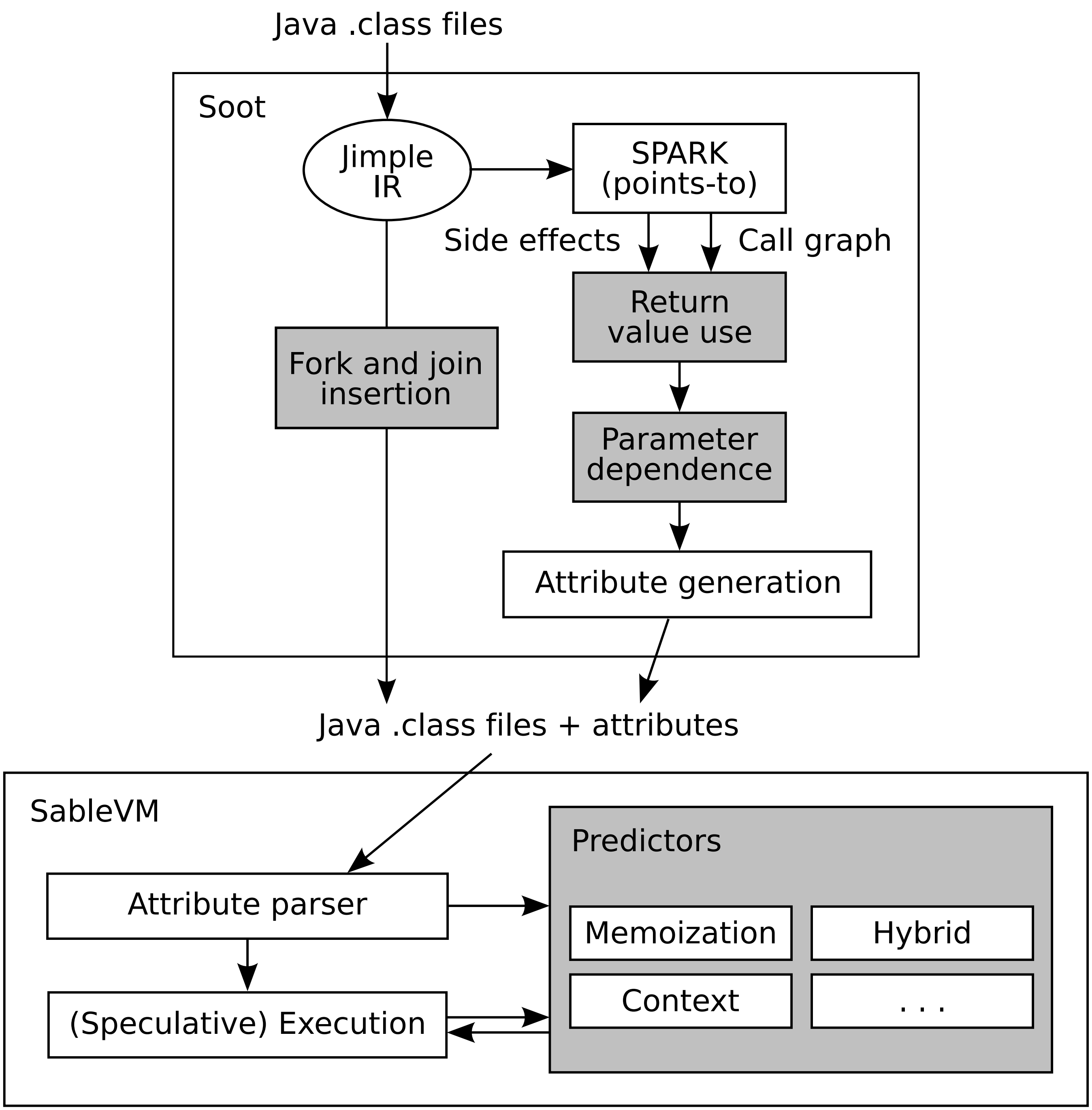
1. Context and memoization predictors behave quite differently, but hybrid allows them to complement each other.

2. Memory requirements of table-based predictors:
   - Large context table: callsite *produces* highly variable data
   - Large memoization table: callsite *consumes* highly variable data

Finally, runtime profiling is used to improve accuracy and reduce memory requirements.

C.J.F. Pickett and C. Verbrugge. Return value prediction in a Java virtual machine. *Second Value-Prediction and Value-Based Optimization Workshop (VPW2) at ASPLOS XI*, Boston, MA, Oct. 2004.

Static Analysis Integration



Static Analysis Integration

- Return Value Use (RVU):

	unconsumed	inaccurate
static	10%	21%
dynamic	3%	14%

  - predictor accuracy: gain up to 7%
  - predictor memory: save 3%

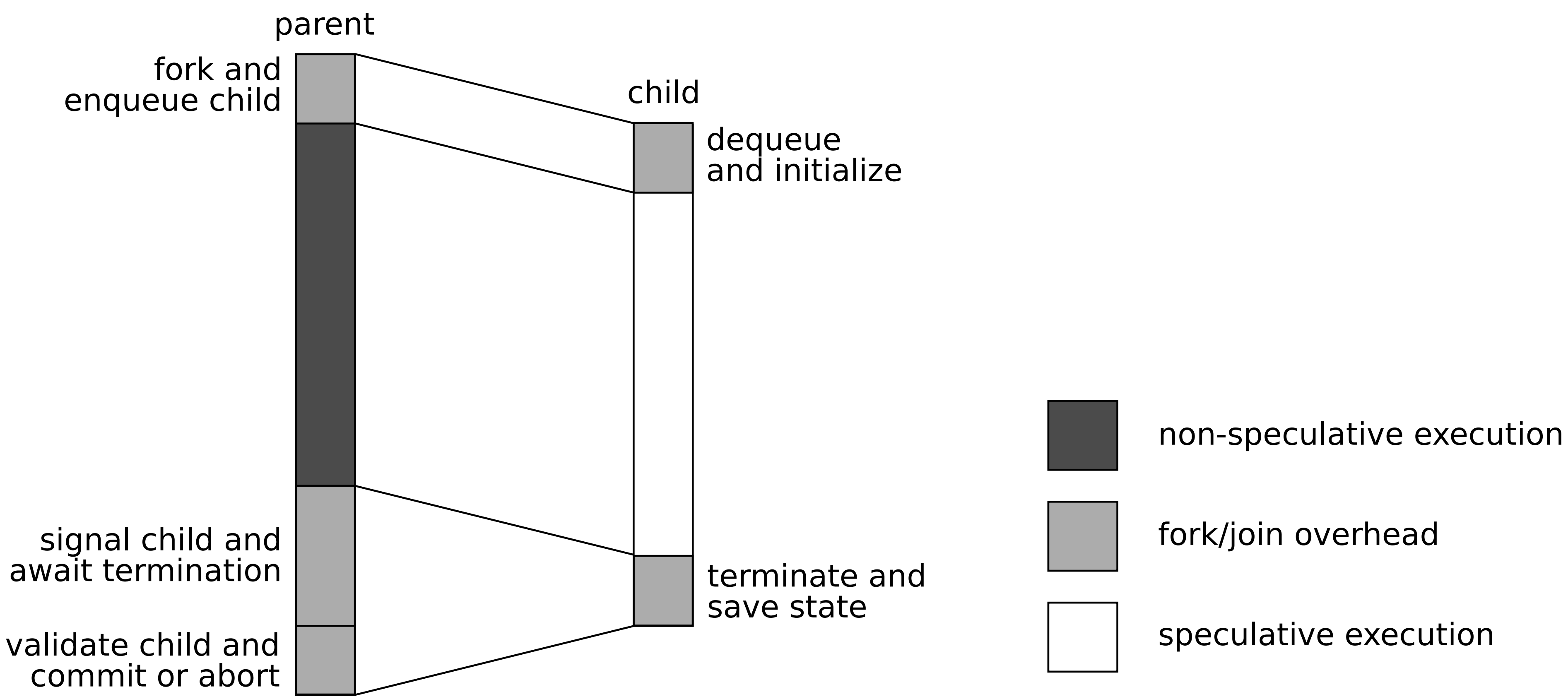
- Parameter Dependence (PD):

	zero dependences	partial dependences
static	25%	23%
dynamic	7%	3%

  - memoization accuracy: gain up to 13%
  - predictor memory: save 2%

Overall System Behaviour

Speculation overhead:



Non-speculative Thread Overhead

parent execution	comp	db	jack	javac	jess	mpeg	mtrt	rt
USEFUL WORK	39%	24%	29%	30%	21%	59%	49%	58%
initialize child	2%	5%	3%	4%	4%	2%	1%	2%
enqueue child	4%	10%	10%	9%	7%	3%	2%	2%
TOTAL FORK	6%	15%	13%	13%	11%	5%	3%	4%
update predictor	7%	13%	12%	11%	12%	6%	7%	7%
delete child	5%	5%	5%	4%	5%	2%	2%	2%
signal and wait	15%	14%	11%	11%	19%	8%	26%	11%
validate prediction	4%	4%	4%	5%	7%	3%	2%	3%
validate buffer	4%	6%	6%	5%	5%	3%	1%	2%
commit child	5%	5%	7%	6%	6%	3%	2%	3%
abort child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
clean up child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
profiling	11%	10%	10%	12%	11%	7%	5%	6%
TOTAL JOIN	53%	59%	57%	56%	67%	34%	47%	36%
PROFILING	2%	2%	1%	1%	1%	2%	1%	2%

Non-speculative Thread Overhead

parent execution	comp	db	jack	javac	jess	mpeg	mtrt	rt
USEFUL WORK	39%	24%	29%	30%	21%	59%	49%	58%
initialize child	2%	5%	3%	4%	4%	2%	1%	2%
enqueue child	4%	10%	10%	9%	7%	3%	2%	2%
TOTAL FORK	6%	15%	13%	13%	11%	5%	3%	4%
update predictor	7%	13%	12%	11%	12%	6%	7%	7%
delete child	5%	5%	5%	4%	5%	2%	2%	2%
signal and wait	15%	14%	11%	11%	19%	8%	26%	11%
validate prediction	4%	4%	4%	5%	7%	3%	2%	3%
validate buffer	4%	6%	6%	5%	5%	3%	1%	2%
commit child	5%	5%	7%	6%	6%	3%	2%	3%
abort child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
clean up child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
profiling	11%	10%	10%	12%	11%	7%	5%	6%
TOTAL JOIN	53%	59%	57%	56%	67%	34%	47%	36%
PROFILING	2%	2%	1%	1%	1%	2%	1%	2%

Non-speculative Thread Overhead

parent execution	comp	db	jack	javac	jess	mpeg	mtrt	rt
USEFUL WORK	39%	24%	29%	30%	21%	59%	49%	58%
initialize child	2%	5%	3%	4%	4%	2%	1%	2%
enqueue child	4%	10%	10%	9%	7%	3%	2%	2%
TOTAL FORK	6%	15%	13%	13%	11%	5%	3%	4%
update predictor	7%	13%	12%	11%	12%	6%	7%	7%
delete child	5%	5%	5%	4%	5%	2%	2%	2%
signal and wait	15%	14%	11%	11%	19%	8%	26%	11%
validate prediction	4%	4%	4%	5%	7%	3%	2%	3%
validate buffer	4%	6%	6%	5%	5%	3%	1%	2%
commit child	5%	5%	7%	6%	6%	3%	2%	3%
abort child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
clean up child	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
profiling	11%	10%	10%	12%	11%	7%	5%	6%
TOTAL JOIN	53%	59%	57%	56%	67%	34%	47%	36%
PROFILING	2%	2%	1%	1%	1%	2%	1%	2%

Speculative Thread Overhead

helper execution	comp	db	jack	javac	jess	mpeg	mtrt	rt
IDLE	86%	82%	78%	78%	78%	55%	53%	71%
INITIALIZE CHILD	3%	4%	4%	4%	4%	2%	5%	4%
startup	<1%	<1%	<1%	<1%	<1%	<1%	1%	<1%
query predictor	3%	5%	4%	4%	6%	5%	15%	8%
useful work	5%	6%	10%	10%	10%	34%	20%	13%
shutdown	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
profiling	<1%	<1%	<1%	<1%	<1%	1%	2%	1%
EXECUTE CHILD	9%	12%	16%	16%	17%	41%	40%	24%
CLEAN UP CHILD	<1%	<1%	<1%	<1%	<1%	< 1%	<1%	<1%
PROFILING	1%	1%	1%	1%	<1%	1%	1%	<1%

Speculative Thread Overhead

helper execution	comp	db	jack	javac	jess	mpeg	mtrt	rt
IDLE	86%	82%	78%	78%	78%	55%	53%	71%
INITIALIZE CHILD	3%	4%	4%	4%	4%	2%	5%	4%
startup	<1%	<1%	<1%	<1%	<1%	<1%	1%	<1%
query predictor	3%	5%	4%	4%	6%	5%	15%	8%
useful work	5%	6%	10%	10%	10%	34%	20%	13%
shutdown	<1%	<1%	<1%	<1%	<1%	<1%	<1%	<1%
profiling	<1%	<1%	<1%	<1%	<1%	1%	2%	1%
EXECUTE CHILD	9%	12%	16%	16%	17%	41%	40%	24%
CLEAN UP CHILD	<1%	<1%	<1%	<1%	<1%	< 1%	<1%	<1%
PROFILING	1%	1%	1%	1%	<1%	1%	1%	<1%

Parallelism Metrics

Speculative thread lengths:

- In hardware simulations, max 40 *machine* instructions is great
- In software, we can get 100s of *bytecode* instructions

	$<$ 10 bytecodes committed	$>$ 90 bytecodes committed
ST mode children	30%	15%
MT mode children	80%	2%

Speculative coverage:

- Percentage of entire program executed successfully in parallel.
- 4 processors, MT mode, $-$RVP: 19%
- 4 processors, MT mode, $+$RVP: 33%

Execution Times and Relative Speedup

experiment	comp	db	jack	javac	jess	mpeg	mtrt	rt	mean
SpMT must fail	1297s	931s	293s	641s	665s	669s	1017s	1530s	722s
SpMT may pass	1224s	733s	211s	468s	405s	662s	559s	736s	539s
relative speedup	1.06x	1.27x	1.39x	1.37x	1.64x	1.01x	1.82x	2.08x	1.34x
vanilla SableVM	368s	144s	43s	108s	77s	347s	55s	67s	120s
actual slowdown	3.33x	5.09x	4.91x	4.33x	5.26x	1.91x	10.16x	10.99x	4.49x

Execution Times and Relative Speedup

experiment	comp	db	jack	javac	jess	mpeg	mtrt	rt	mean
SpMT must fail	1297s	931s	293s	641s	665s	669s	1017s	1530s	722s
SpMT may pass	1224s	733s	211s	468s	405s	662s	559s	736s	539s
relative speedup	1.06x	1.27x	1.39x	1.37x	1.64x	1.01x	1.82x	2.08x	1.34x
vanilla SableVM	368s	144s	43s	108s	77s	347s	55s	67s	120s
actual slowdown	3.33x	5.09x	4.91x	4.33x	5.26x	1.91x	10.16x	10.99x	4.49x

Outline

Conclusions & Future Work

Conclusions:

- New software SpMT framework for Java
- Facilitates experimental analysis, profiling, and development of new techniques

Future Work:

- Different speculation modes: loop, lock, basic block
- Move speculation components into language-independent library
- Performance improvements, actual speedup
- IBM Testarossa JIT and J9 VM integration