Karlsruhe Institute of Technology

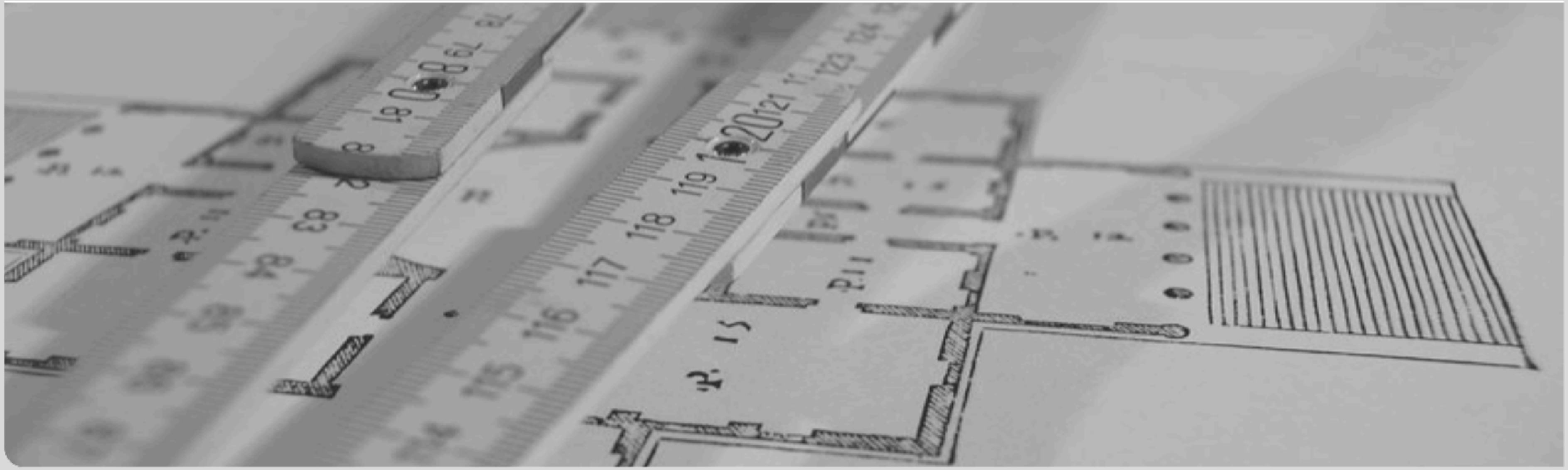
Using Invariant Analysis for Improving Instrumentation-based Performance Evaluation of SPECjvm2008 Benchmarks

Michael Kuperberg, Martin Krogmann, Ralf Reussner

Karlsruhe Institute of Technology

SOFTWARE DESIGN AND QUALITY GROUP

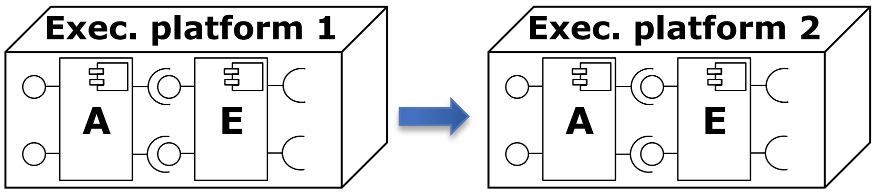
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS



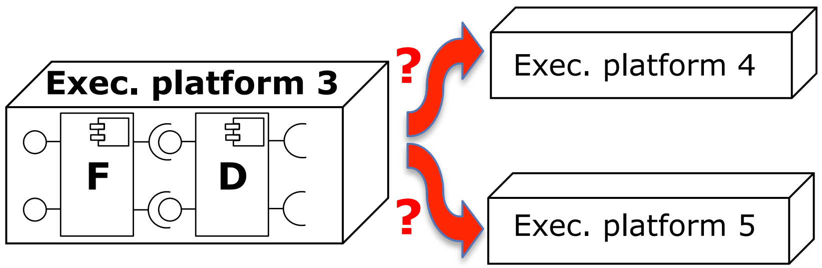
Motivation

- **Cross-platform performance prediction** [KKR2008a] for systematic engineering of component-based software
  - Performance in our case: execution duration of component services

- Performance prediction e.g. for following scenarios:
  - **Relocation** of an application to another execution platform



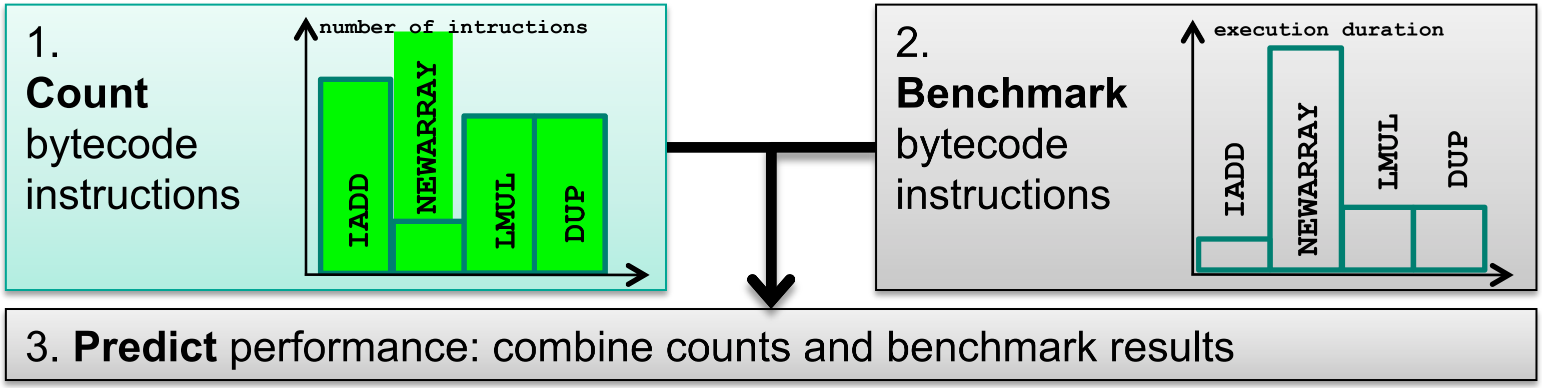
  - **Sizing**: choosing appropriate execution platform to fulfil changed perf. requirements



Software Design and Quality Group
Institute for Program Structures and Data Organization

Bytecode-based Performance Prediction

- Context of presented work: bytecode-based **performance prediction** [KKR2008a] for existing components:
  - Performance of a component on other execution platform
  - Bytecode instructions counts as a performance metric

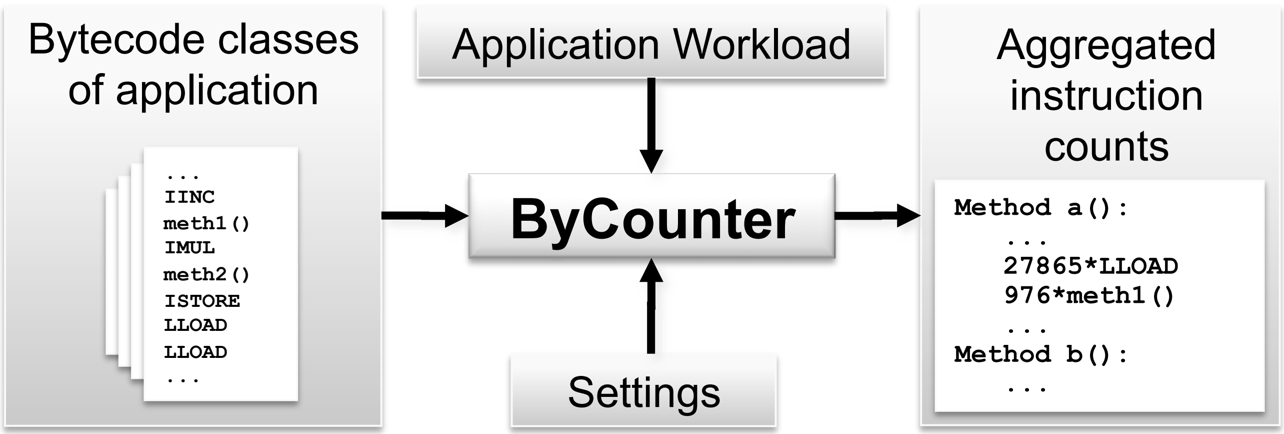


- Counting must be performed at runtime, since static analysis or symbolic execution not sufficient
- Must be applicable to sourceless and legacy components

Software Design and Quality Group
Institute for Program Structures and Data Organization

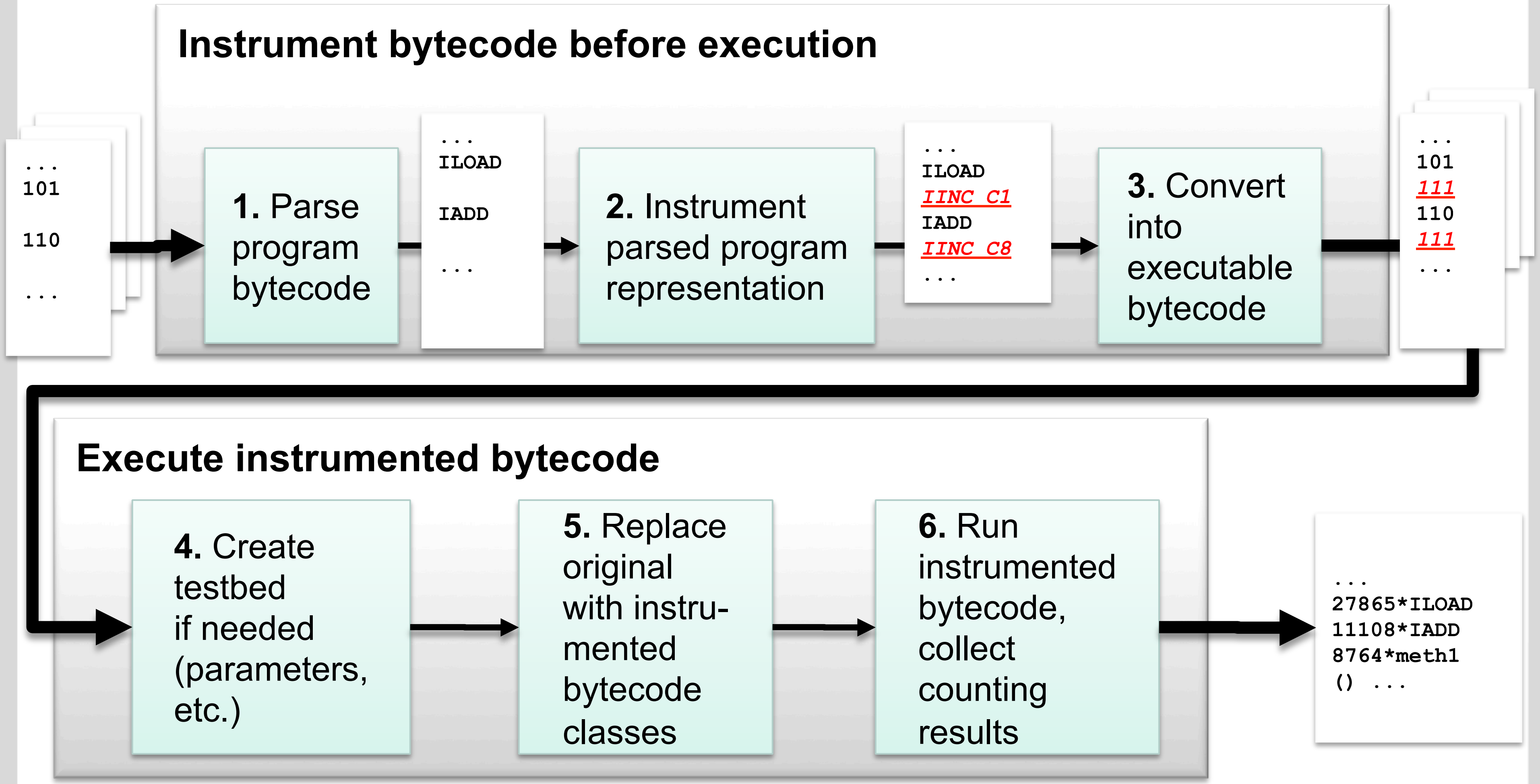
ByCounter: Runtime Bytecode Instruction Counting using Application Instrumentation

- ByCounter collects runtime counts of Java bytecode instructions and method invocations



  - Counts different instruction types individually
  - Configurable parameter recording for array-related instructions
  - Not constrained by timer accuracies and costs (cf. short methods)
  - **Based on JVM-independent application instrumentation**

Overview over the ByCounter Process



Idea and Advantages of ByCounter

- **Idea**: instrument the application, not the virtual machine
  - Insert counters into existing bytecode, preserve method signatures
- **Advantages**:
  - Instrumentation **transparent** to the application: no functional side-effects (but: runtime overhead)
  - Method invocations by the bytecode of the instrumented method: configurable and extendable treatment
  - No dependence on native interfaces, works on any JVM
  - Idea applicable to Dalvik, CLR etc.
- Previous approaches: use modified JVMs or JVMTI etc.
  - Insufficient portability; not desirable in production environments

Example: SOR Part of the Scimark Benchmark in SPECjvm2008

```
public final double num_flops(int M,
int N, int num_iterations) {
    long a=0;
    a++;
    double Md = (double) M;
    double Nd = (double) N;
    double num_iterD =
        (double) num_iterations;
    return Md-1)*
        ((Nd-1)*num_iterD*6.0;
    }
```

```
LCONST_0   LSTORE 4
LLOAD 4    LCONST_1   LADD   LSTORE 4
ILOAD 1    I2D        DSTORE 6
ILOAD 2    I2D        DSTORE 8
ILOAD 3    I2D        DSTORE 10
DLOAD 6    DCONST_1   DSUB   DLOAD 8
DCONST_1   DSUB       DMUL   DLOAD 10
DMUL       LDC 6.0    DMUL   DRETURN
```

- No jumps, loops, method invocations or other control flow

➔ The number of executed bytecode instructions...

  - ... is independent of the input parameter values of **`num_flops`**
  - ... is independent of the state of the invocation target
  - ... **can be determined statically**

Software Design and Quality Group
Institute for Program Structures and Data Organization

Switching to Bytecode Instruction Sequences

Karlsruhe Institute of Technology

-
  - ... is costly in terms of runtime overhead (CPU, memory)
  - ... limits scalability, offers room for improvement
- **performance-invariant bytecode instruction sequences**
  - Decreases amount of inserted instrumentation
  - Maintains existing precision of counting results
  - Similar to basic blocks (and dictionaries in data compression)
- workloads of the SPECjvm2008 benchmark

PIBISes: Treatment in ByCounter

- PIBISes are not identical to *basic blocks:*
  - As with basic blocks: no jumps etc. allowed
  - Additionally: a PIBIS may not contain instructions with parameter-dependent performance (which can change between executions: cf. **`size`** parameter of **`newarray`**)
- Extended ByCounter: identifies PIBISes
  - Instead of 1 counter incrementation for every single executed instruction: 1 incrementation per PIBIS exec.
  - Note that some PIBISes still contain just one instruction

Implementation of ByCounter for Java

Karlsruhe Institute of Technology

1. Parse program bytecode

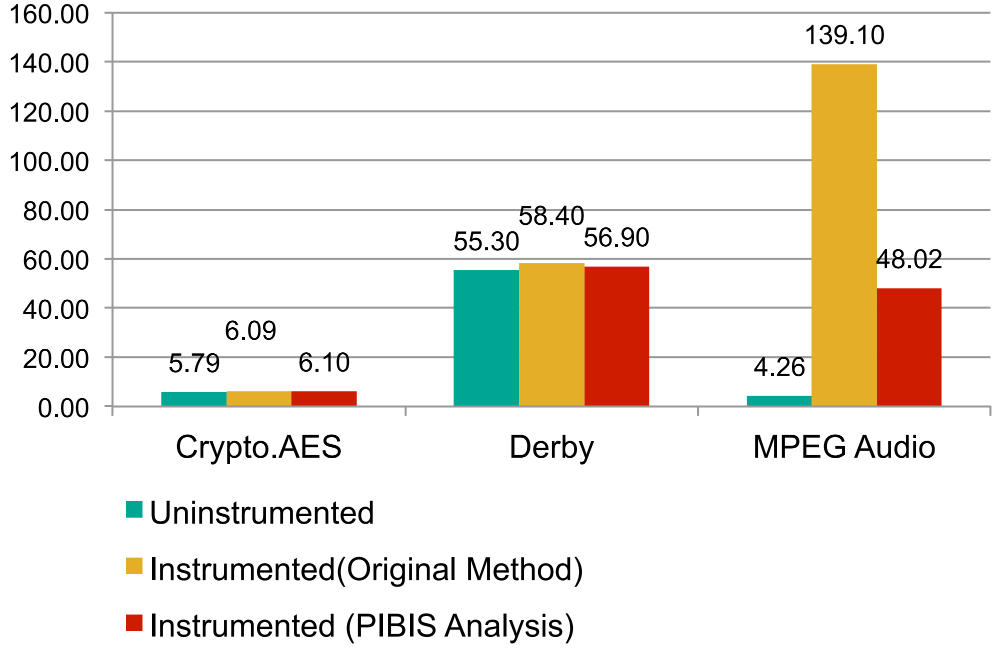
- Analysable, easily modifiable representation
  - Obtained using ASM framework

2. Instrument parsed program representation and run resulting bytecode

- **Insert counting instrumentation into application**
  - Counters are `long`-typed bytecode local variables (invisible outside the instrumented method),
  - Counters initialised when method execution starts
  - **Each execution of instruction/PIBIS: counter is also incremented**
  - Report counters at method exit points (write to a log file or report to a central „collector“ daemon)

- Instrumented .class files: persistable, usable by any `ClassLoader`
- Existing workloads, harnesses, scripts and configurations can be used

Preliminary Results



- Durations in seconds
- Median values based on 21 measurements using **`java.lang.System.nanoTime()`**
- Durations include result aggregation and storage
- JITting takes place (proof: **`-XX:+PrintCompilation`** JVM flag to enable logging)

Evaluation platform (runs Mac OS X 10.6.4, 64 bit):

- 2.8 GHz Intel Core 2 Duo, 4 GB of 1067 MHz DDR3 main memory
- JVM 1.6.0_20 provided by Apple (default mode, equals **`-server`**)
- **`-Xmx768M`** JVM flag to allocate 768 MB of heap memory

Related Work

- Concerning SPECjvm98:
  - [Gregg et al., 2002] *modified JVM to* benchmarking methods and bytecode instructions, no research on counting overhead
  - [Lambert and Power, 2005] static/dynamic frequencies of *basic blocks*
  - [Li et al., 2000] complete system simulation: not addressing bytecode-level basic blocks or precise bytecode counts
- SPECjvm2008
  - [Oi, 2009], [Oi, 2010] compared other performance metrics, different JVMs
  - [Shiv et al., 2009] impact of hardware architecture details on SPECjvm2008 performance in comparison to other SPEC benchmarks
- JVM-internal basic block analysis for Just-in-Time compilation etc.
  - Analysis results not available to platform-independent counting tools
  - Program optimisers, escape analysis and control flow graph analysis of basic blocks have different objectives

Assumptions and Limitations

- Subsequences (i.e. Sub-PIBISes) irrelevant: PIBISes should be as large as possible

- Bytecode supplied to ByCounter must be „final“
  - Complex classloading in application servers: to test
  - ByCounter works as JVM „instrumentation agent“, too

- JIT impact to be considered
- Further evaluation needed (e.g. SPECjbb2005)
- Instrumenting Java Platform API methods: t.b.d.

Future Work

Karlsruhe Institute of Technology

- Further potential for decreasing runtime overhead
  - Identify performance-invariant methods:
    no need for result reporting each time (counts constant)
  - Parallelise evaluation and aggregation of results on multi-core execution platforms
- Combine with purity analysis
  - To prevent counting code that otherwise is „dead code“
- Study the shape/contents of different PIBISes
  - Also: their static/dynamic frequency
- Compare overhead to JVMTI-based tools

Bibliography

[BLC2002a] Bruneton, E., Lenglet, R., and Coupaye, T. (2002). ASM: a code manipulation tool to implement adaptable systems. *Adaptable and Extensible Component Systems*. http://asm.ow2.org.

[GPW2002a] Gregg, D., Power, J., and Waldron, J. (2002). Benchmarking the Java virtual architecture - the specjvm98 benchmark suite. *Java Microarchitectures*, pages 1–18.

[HKRR2009a] Hauck, M., Kuperberg, M., Krogmann, K., and Reussner, R. (2009). Modelling Layered Component Execution Environments for Performance Prediction. Springer LNCS, 2009

[KB2007a] Kuperberg, M. and Becker, S. (2007). Predicting Software Component Performance: On the Relevance of Parameters for Benchmarking Bytecode and APIs. *Proceedings of the 12th International Workshop on Component Oriented Programming (WCOP 2007)*.

[KKR2008a] Kuperberg, M., Krogmann, K., and Reussner, R. (2008). Performance Prediction for Black-Box Components using Reengineered Parametric Behaviour. Springer LNCS, 2008.

[KKR2009a] Kuperberg, M., Krogmann, M., and Reussner, R. (2009). TimerMeter: Quantifying Properties of Software Timers for System Analysis. Proceedings of QEST2009.

[KKR2010a] Krogmann, K., Kuperberg, M., and Reussner, R. (2010). Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction. *IEEE Transactions on Software Engineering*. Accepted for publication, to appear 2010.

[SPECjvm2008] SPECjvm2008 Benchmarks. SPEC Corporation. http://www.spec.org/jvm2008/

Conclusions

- Runtime bytecode instruction counts using ByCounter: platform-independent dynamic performance metric
  - Successful usage in cross-platform perf. prediction [KKR2008a]
  - Uses transparent instrumentation of application bytecode
  - Neither profilers nor JVM monitoring tools are instruction-precise
- New: to decrease overhead in ByCounter: identify and use **performance-invariant bytecode instruction sequences**
- Evaluation shows significant overhead decrease, e.g. for SPECjvm2008 MPEGaudio: 2.9x lesser runtime overhead