Making SPL practical via Java annotations
[work in progress report]

Vojtěch Horký

3rd January, 2012
Outline

- SPL
- Prototype implementation of measuring framework
  - Features & limitations
  - Demonstration
- Future work & other ideas*

* Gathered from the authors of the *Capturing Performance Assumptions using SPL* paper
Stochastic performance logic
SPL in a nutshell

- Capturing function performance...
  - platform independent
  - automatic evaluation
- “performance assertions”

\[ \forall n \in \{500, 1000\} : P_{MD5}(n) \leq p(id, \lambda x.100x) P_{memcpy}(n) \]
SPL in a nutshell – annotations

```java
@SPL(
    generators = "org.pkg.gen.Generator(arg)",
    formula = "for n {1,2} "+ "SELF(n) <= org.otherpkg.class#f(n)"
)
void measuredFunction(int[] parameter) {
    ...
}
```
Annotation crash-course
@SPL annotation

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface cz.cuni.mff.d3s.spl.SPL {

    /* Method aliases. */
    String[] methods() default {};

    /* Generator aliases. */
    String[] generators() default {};

    /* Actual formula. */
    String[] formula() default {};

}
Annotating methods

- Devise the formula...
- ...and annotate it ;-)
Creating generators

- provide parameters for the measured method
- `Iterable<Object[]>`
  - `Object[]` used by `Method.invoke()`
- `int` parameters (for n 100, 500 ...)
Prototype implementation
Scope of the prototype

- Java annotations
- "Simpler" formulas
  - lambda functions – constant multiplication (x \cdot 0.95x)
- Measurement results processed by R
- No implicit generators
Output

- OKAY/FAIL for each @SPL-ed method
- Histograms
- Computed statistics
Java & statistics strike back

Expected / ideal distribution
Java & statistics strike back

Typical measurement
Java & statistics strike back

Better cache usage?
Java & statistics strike back

JIT turned off...
Java & statistics strike back (moral)

- Fresh JVM for each measurement
- CPU governor does matter
- JIT does matter
- Warm-up phase needed (badly)
- Removing outliers
Demonstration

Comparing own implementation of quicksort with java.util.Arrays.sort()

Examples from the paper.
Behind the scene

• Get @SPL annotated methods
• Build a formula tree
• Expand argument values
• Gather measured methods with actual parameters
• Measure the performance
• Compute results with R, generate XML
• Convert XML to HTML
Integrating into Ant
Caveats

- Deployment
- @SPL is intrusive
  - separate source tree
- Unattractive output
Demonstration
Future work
Near future

- Deployment
- Nicer interface for user
Regression testing

- Implicit
- Explicit
  - Scenario
    1. Profiling reveals problem in `func()`
    2. Improvement implemented in revision 5147
    3. Goal: ensure that performance would be always better than in 5146
  - Idea: extend the alias specification with revision number
    - `OLD:org.pkg.SomeClass(ctorArg)func@5146`
    - for `n` ... `SELF(n) <= OLD(n)`
More precise measuring

- `System.nanoTime()`
- Garbage collector
- Just-in-time compilation & co.
- Caches
- Instrumenting
Assorted thoughts & more distant future

- Multi threaded
- Integration with other tools (Eclipse, Hudson, …)
- Integration with JUnit
  - Use existing data generators
- Use in real project
- Port to other languages
Conclusion
Conclusion

• Usable for collecting data
• Needs better interface. . .
• . . .to allow adding to existing projects
Thank you.

Questions?