Performance Evaluation of Computer Systems

Instrumentation (Pin & DiSL)

Vojtěch Horký    Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University in Prague
Czech Republic

April 12, 2017
Task

Routine duration
Measure duration of a single routine, excluding called subroutines.

Histogram of memory writes
Record values written to memory as a histogram.

Implementation
Java/DiSL or C++/Pin.
Source Code & Submission

Harness and examples

http://d3s.mff.cuni.cz/teaching/peva/files/lab03.tar.gz
(Includes compilation and run scripts.)

Submission

By e-mail to horky@d3s.mff.cuni.cz.

Send TAR (or ZIP) with (commented) sources, run scripts and a brief README.
DiSL Framework (Java)
DiSL

- Dynamic Java instrumentation.
- Instrumentation of
  - methods (entry and exit points),
  - basic blocks,
  - individual byte-code instructions.
- Can instrument code of `java.lang` classes (reckless instrumentation can crash the JVM, though).
- Developed and maintained by Charles University, University of Lugano and Shanghai Jiao Tong University.
DiSL – Source Code Example

**Instrumentation.java**

```java
public class Instrumentation {
    @Before(marker = BodyMarker.class, scope = "my.pkg.MyClass.*")
    public static void onMethodEnter(MethodStaticContext msc) {
        System.out.println(msc.thisMethodFullName());
    }
}
```

**Manifest file**

Manifest-Version: 1.0
DiSL-Classes: Instrumentation
DiSL – Compilation & Launching

Instrumentation compiled separately

javac Instrumentation.java
jar cfm MANIFEST.MF instr.jar Instrumentation.class

Launcher script

disl.py -d $DISL_HOME/output/ -- \
    instr.jar \n    -cp . [JVM arguments] \n    your.pkg.Application [application arguments]
### DiSL – Provided Example

#### Example instrumentation ...

... collect list of called methods.

... counts number of synchronized sections.

#### Scripts

<table>
<thead>
<tr>
<th>Script</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prepare.sh</code></td>
<td>Downloads and compiles DiSL; downloads DaCapo.</td>
</tr>
<tr>
<td><code>example/*.java</code></td>
<td>Example application and DiSL instrumentation.</td>
</tr>
<tr>
<td><code>example/make.sh</code></td>
<td>Builds application, packs instrumentation JAR.</td>
</tr>
<tr>
<td><code>example/run.sh</code></td>
<td>Runs application with DiSL instrumentation.</td>
</tr>
</tbody>
</table>
Pin Framework (C, C++)
Pin

• Dynamic instrumentation of binaries.
• Instrumentation of
  • methods,
  • individual instructions.
• Developed by Intel.
Pin – Source Code Example

```c
void printRoutineName(string name) { cerr << name << endl; }

VOID instrumentation(RTN routine, VOID *ignored) {
    RTN_Open(routine);
    RTN_InsertCall(routine, IPOINT_BEFORE, (AFUNPTR) printRoutineName,
                   IARG_PTR, RTN_Name(routine), IARG_END);
    RTN_Close(routine);
}

int main(int argc, char *argv[]) {
    PIN_Init(argc, argv);
    PIN_InitSymbols();
    RTN_AddInstrumentFunction(instrumentation, NULL);
    PIN_StartProgram();
}
```
Pin – Compilation & Launching

**Use Makefiles from Pin distribution**

```make
make
```

**Launching**

```shell
../pin-3.0-76991-gcc-linux/pin \  
  -t obj-intel64/YourTool.so \  
  -- your-app [application arguments]
```
**Example instrumentation ...**

... prints called methods.
... counts number of system calls.
... counts number of NOPs.

**Scripts**

`prepare.sh` – Downloads Pin, compiles example application.
`Example/Example.cpp` – Pin instrumentation.
`example/make*` – Pin-specific Makefiles.
`example/run.sh` – Runs application with Pin instrumentation.
Tasks
Task I: Routine Duration

Measure duration of a single routine, excluding duration of any nested function calls.

```c
void inner() {
    // Something that takes 5s
}

void measured_routine() {
    // Something that takes 3s
    inner();
    // Something that takes 4s
}
```

Your instrumentation shall report duration of 7s, not 12s.

Note: `sleep()` is also a function call.
Task II: Histogram of Memory Writes

Create a histogram of values written to memory by your program. Estimate overhead of the instrumentation.

It is okay to record only lowest 4 bits (i.e. use array of 16 members to record counts).

For C++, record all memory writes.

For Java, record stores of all integer types.
Hints & Caveats
Stable Environment

Use aiya.ms.mff.cuni.cz for running your code.

Consider using SSHFS so you can edit files locally but execute the binaries on the remote machine.

On local machine, do:

```bash
mkdir aiya-mount
sshfs LOGIN@aiya.ms.mff.cuni.cz:/home/LOGIN/ aiya-mount/
```

Now, you can edit files in aiya-mount locally.

But **run your code on aiya** via SSH.
Other Hints

DiSL

• Limit your instrumentation to one class only at first (System.out.println may crash JVM on java.lang classes).
• Estimate overhead on a reasonable benchmark, e.g. on DaCapo (java -jar dacapo-9.12-bach.jar -n 30 xalan).

Pin

• IPOINT_AFTER on routines do not work reliably (see FAQ).
• Use OS_Time() to query current time.
• Estimate overhead on a reasonable benchmark, use dd at least (dd if=/dev/urandom of=/dev/null bs=8 count=1000).

Histogram of memory writes

• Start with one routine to test your instrumentation.