Why Aren’t More Users More Happy With Our VMs?

Laurence Tratt

Warmup work in collaboration with:
Edd Barrett, Carl Friedrich Bolz, Rebecca Killick, and Sarah Mount

Software Development Team
2018-02-05
JVMs bring "gcc -O2" to the masses

–Cliff Click: A JVM does that?
What do VM claims pertain to?

iteration time

in-process iteration
What do VM claims pertain to?
What do VM claims pertain to?
What do VM claims pertain to?

![Graph showing iteration time vs. in-process iteration]

- Compilation
- Profiling Interpreter
What do VM claims pertain to?

![Graph showing iteration time vs. in-process iteration with labels for Compilation, Profiling Interpreter, and Peak Performance.](http://soft-dev.org/)
What do VM claims pertain to?
Users *always* perceive warmup
Users *always* perceive warmup

Maybe we should know how long it is?
Measure warmup of modern language implementations
Measure warmup of modern language implementations

*Hypothesis:* Small, deterministic programs reach a steady state of peak performance.
The language benchmark games are perfect for us (unusually)
The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism
The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism

We added checksums to all benchmarks
Method 2: How long to run?

2000 in-process iterations
Method 2: How long to run?

2000 in-process iterations

30 process executions
Method 3: VMs

- Graal-0.22
- HHVM-3.19.1
- JRuby/Truffle (git #6e9d5d381777)
- Hotspot-8u121b13
- LuaJit-2.0.4
- PyPy-5.7.1
- V8-5.8.283.32
- GCC-4.9.4

Note: same GCC (4.9.4) used for all compilation
Method 4: Machines

- Linux\textsubscript{4790}, Debian 8, 24GiB RAM
- Linux\textsubscript{E3-1240v5}, Debian 8, 32GiB RAM
- OpenBSD\textsubscript{4790}, OpenBSD 6.0, 32GiB RAM
Method 4: Machines

- Linux\textsubscript{4790}, Debian 8, 24GiB RAM
- Linux\textsubscript{E3-1240v5}, Debian 8, 32GiB RAM
- OpenBSD\textsubscript{4790}, OpenBSD 6.0, 32GiB RAM

- Turbo boost and hyper-threading disabled
- Network card turned off.
- Daemons disabled (cron, smtpd)
Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible
Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible e.g.:

- Minimises I/O
- Sets fixed heap and stack ulimits
- Drops privileges to a ‘clean’ user account
- Automatically reboots the system prior to each proc. exec
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)

HTTP://SOFT-DEV.ORG/
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #12 (warmup)

In-process iteration

Time (secs)

0.56523
0.56703
0.56882
0.57061
0.57241
0.57420
0.57600
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #12 (warmup)

In-process iteration

Time (secs)

1 201 401 601 801 1001 1201 1401 1601 1801 2000

0.56523
0.56703
0.56882
0.57061
0.57241
0.57420
0.57600

0.57600
0.57420
0.57241
0.57061
0.56882
0.56703
0.56523

11/49 HTTP://SOFT-DEV.ORG/
Fannkuch Redux, LuaJIT, OpenBSD\textsubscript{4790}, Proc. exec. #12 (warmup)

Changepoint
Fannkuch Redux, LuaJIT, OpenBSD\textsuperscript{4790}, Proc. exec. #12 (warmup)

Changepoint

Changepoint segment
Method 7: Classification

Classification algorithm (steps in order):
All segs are equivalent: *flat*
Method 7: Classification

Classification algorithm (steps in order):

All segs are equivalent: \textit{flat}

Final seg is in fastest set: \textit{warmup}
Fasta, V8, Linux, Proc. exec. #15 (warmup)
Warmup & flat (2)
Richards, Hotspot, Linux\textsuperscript{E3 – 1240v5}, Proc. exec. #3 (slowdown)
Method 7: Classification

Classification algorithm (steps in order):

- All segs are equivalent: *flat*
- Final seg is in fastest set: *warmup*
Method 7: Classification

Classification algorithm (steps in order):

All segs are equivalent: flat

Final seg is in fastest set: warmup

Final seg is not in fastest set: slowdown
In-process iteration

1.14063
1.14575
1.15087
1.15599
1.16111
1.16623
1.17135

Fasta, V8, Linux_{4790}, Proc. exec. #14 (slowdown)

Time (secs)

1.14086
1.15142
1.16197

16/49 HTTP://SOFT-DEV.ORG/
No steady state (1)

Binary Trees, V8, Linux\textsuperscript{4790}, Proc. exec. #24 (no steady state)
Classification algorithm (steps in order):

All segs are equivalent: *flat*

Final seg is in fastest set: *warmup*

Final seg is not in fastest set: *slowdown*
Classification algorithm (steps in order):

All segs are equivalent: \textit{flat}

Final seg is in fastest set: \textit{warmup}

Final seg is not in fastest set: \textit{slowdown}

Else: \textit{no steady state}
Classification algorithm, in order:

All segs are equivalent: *flat*

Final seg is in fastest set: *warmup*

Final seg is not in fastest set: *slowdown*

Else: *no steady state*

**Good**
Classification algorithm, in order:

All segs are equivalent: \textit{flat}

Final seg is in fastest set: \textit{warmup}

Final seg is not in fastest set: \textit{slowdown}

Else: \textit{no steady state}

\textbf{Bad}
Warmup or no steady state?

Fannkuch Redux, Hotspot, Linux, Proc. exec. #1 (warmup)

Time (secs) vs In-process iteration

0.38753
0.37986
0.37218
0.36451
0.35683
0.34916
0.34148

In-process iteration

1 201 401 601 801 1001 1201 1401 1601 1801 2000

0.34357
0.34269
0.34181

601 801 1000

Fannkuch Redux, Hotspot, Linux, Proc. exec. #1 (warmup)
Inconsistent Process-executions

(Binary Trees, V8, Linux\textsuperscript{E3−124v5}, Proc. exec. #7 (warmup))

(Binary Trees, V8, Linux\textsuperscript{E3−124v5}, Proc. exec. #8 (slowdown))

(Same machine)
Inconsistent Process-executions

(Different machines. Bouncing ball Linux-specific)
## Individual benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
<th>Class</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>(27L, 3J)</td>
<td>32.0 (70, 139.8)</td>
<td>6.60 (3.728, 54.406)</td>
<td>0.18594</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td>(24L, 4J, 2w)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>(25L, 5J)</td>
<td>7.0 (70, 53.5)</td>
<td>1.19 (1.182, 9.705)</td>
<td>0.18279</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>(1082.0)</td>
<td>2219.59</td>
<td>2.05150</td>
<td></td>
<td>(999.0, 1283.5)</td>
<td>(2039.394, 2516.02)</td>
<td>(0.017738)</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(23L, 4J, 2w, 1e)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>(27J, 3w)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>(15L, 6J)</td>
<td>1.25 (0.089, 0.265)</td>
<td>0.49237</td>
<td></td>
<td>(0.089, 0.265)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Results

- **C**:
  - Steady iter: 32.0 (70, 139.8)
  - Steady perf: 0.18594

- **Graal**: (27L, 3J)
  - Steady iter: 6.60 (3.728, 54.406)

- **HHVM**: (24L, 4J, 2w)
  - Steady perf: 0.18279

- **HotSpot**: (25L, 5J)
  - Steady perf: 0.18279

- **JRuby+Truffle**: (1082.0)
  - Steady perf: 0.18279

- **LuaJIT**: (23L, 4J, 2w, 1e)
  - Steady perf: 0.18279

- **PyPy**: (27J, 3w)
  - Steady perf: 0.18279

- **V8**: (15L, 6J)
  - Steady perf: 0.49237

### Additional Results

- **Richards**:
  - Steady perf: 0.95909

- **Fastas**:
  - Steady perf: 0.95909

- **Spectrals**:
  - Steady perf: 0.95909

---

Software Development Team

http://soft-dev.org/
# Individual benchmark stats

<table>
<thead>
<tr>
<th>Class.</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>(27, 3)</td>
<td>(17.0, 193.8)</td>
<td>(3.729, 36.608)</td>
</tr>
<tr>
<td>HHVM</td>
<td>(24, 4, 2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>(25, 5)</td>
<td>(7.0, 53.5)</td>
<td>(1.182, 9.703)</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(23, 4, 2, 1)</td>
<td>(999.0, 1232.5)</td>
<td>(2039.304, 2516.021)</td>
</tr>
<tr>
<td>PyPy</td>
<td>(27, 3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>(15, 9, 6)</td>
<td>(1.0, 794.0)</td>
<td>(0.000, 391.026)</td>
</tr>
</tbody>
</table>
## Overall benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux\textsubscript{4790}</th>
<th>Linux\textsubscript{1240v5}</th>
<th>OpenBSD\textsubscript{4790} †</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>8.7%</td>
<td>13.0%</td>
<td>6.7%</td>
</tr>
<tr>
<td>(\parallel)</td>
<td>28.3%</td>
<td>23.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>(\triangleright)</td>
<td>6.5%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>4.3%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>6.5%</td>
<td>6.5%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\neq)</td>
<td>45.7%</td>
<td>43.5%</td>
<td>70.0%</td>
</tr>
</tbody>
</table>

### ⟨VM, benchmark⟩ pairs

### Process executions

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux\textsubscript{4790}</th>
<th>Linux\textsubscript{1240v5}</th>
<th>OpenBSD\textsubscript{4790} †</th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>26.4%</td>
<td>20.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>(\parallel)</td>
<td>48.3%</td>
<td>51.5%</td>
<td>52.1%</td>
</tr>
<tr>
<td>(\triangleright)</td>
<td>16.7%</td>
<td>17.9%</td>
<td>11.1%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>8.7%</td>
<td>9.6%</td>
<td>2.8%</td>
</tr>
</tbody>
</table>
### Overall benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux\textsubscript{4790}</th>
<th>Linux\textsubscript{1240v5}</th>
<th>OpenBSD\textsubscript{4790}\textsuperscript{†}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(VM, benchmark) pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>8.7%</td>
<td>13.0%</td>
<td>6.7%</td>
</tr>
<tr>
<td>↓</td>
<td>28.3%</td>
<td>23.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>∫</td>
<td>6.5%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>≈</td>
<td>4.3%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>=</td>
<td>6.5%</td>
<td>6.5%</td>
<td>13.3%</td>
</tr>
<tr>
<td>≠</td>
<td>45.7%</td>
<td>43.5%</td>
<td>70.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Process executions</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>−</td>
<td>26.4%</td>
<td>20.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>↓</td>
<td>48.3%</td>
<td>51.5%</td>
<td>52.1%</td>
</tr>
<tr>
<td>∫</td>
<td>16.7%</td>
<td>17.9%</td>
<td>11.1%</td>
</tr>
<tr>
<td>≈</td>
<td>8.7%</td>
<td>9.6%</td>
<td>2.8%</td>
</tr>
</tbody>
</table>
Classical warmup occurs for only:
Classical warmup occurs for only:

72.4%–74.7% of process executions
Summary

Classical warmup occurs for only:

72.4%–74.7% of process executions

43.4%–43.5% of (VM, benchmark) pairs
Summary

Classical warmup occurs for only:

72.4%–74.7% of process executions

43.4%–43.5% of (VM, benchmark) pairs

0% of benchmarks for (VM, benchmark, machine) triples
Are odd effects correlated with compilation and GC?

In-process iteration

Fasta, PyPy, Linux\textsubscript{E3–1240v5}, Proc. exec. #5 (no steady state)

Time (secs)

JIT

GC

Software Development Team

http://soft-dev.org/
Are odd effects correlated with compilation and GC?

Richards, Hotspot, Linux$_{E3-1240v5}$, Proc. exec. #3 (slowdown)

In-process iteration

Richards, Hotspot, Linux$_{E3-1240v5}$, Proc. exec. #3 (slowdown)

JIT (secs)

GC (secs)

Time (secs)

In-process iteration
Are odd effects correlated with compilation and GC?

Fannkuch Redux, Hotspot, Linux\textsubscript{E3 – 1240v5}, Proc. exec. #4 (slowdown)

![Graph showing in-process iteration, GC time, JIT time, and total time over iterations.](image-url)
Benchmark suites

Benchmarks guide our optimisations
Benchmarks guide our optimisations

Are they complete guides?
A war story
Symptom: poor performance of a Pyston benchmark on PyPy
Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion
A war story

Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion

Fix: Check for recursion before tracing
A war story: the basis of a fix

diff --git a/rpython/jit/metainterpy.py b/rpython/jit/metainterpy.py
--- a/rpython/jit/metainterpy.py
+++ b/rpython/jit/metainterpy.py
@@ -951,9 +951,31 @@
if warmrunnerstate.inlining:
    if warmrunnerstate.can_inline_callable(greenboxes):
        # We’ve found a potentially inlinable function; now we need to
        + # see if it’s already on the stack. In other words: are we about
        + # to enter recursion? If so, we don’t want to inline the
        + # recursion, which would be equivalent to unrolling a while
        + # loop.
        portal_code = targetjitdriver_sd.mainjitcode
-    return self.metainterpy.perform_call(portal_code, allboxes,
-                                          greenkey=greenboxes)
+    inline = True
+    if self.metainterpy.is_main_jitcode(portal_code):
+        for gk, _ in self.metainterpy.portal_trace_positions:
+            if gk is None:
+                continue
+            assert len(gk) == len(greenboxes)
+            i = 0
+            for i in range(len(gk)):
+                if not gk[i].same_constant(greenboxes[i]):
+                    break
+            else:
+                # The greenkey of a trace position on the stack
+                # matches what we have, which means we’re definitely
+                # about to recurse.
+                inline = False
+                break
+    if inline:
+        return self.metainterpy.perform_call(portal_code, allboxes,
+                                          greenkey=greenboxes)
Success: slow benchmark now 13.5x faster
A war story: mixed fortunes

Success: slow benchmark now 13.5x faster

Failure: some PyPy benchmarks slow down
A war story: mixed fortunes

Success: slow benchmark now 13.5x faster

Failure: some PyPy benchmarks slow down

Solution: allow some tracing into recursion
# A war story: data

<table>
<thead>
<tr>
<th>#unrollings</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>hexiom2</td>
<td>1.3</td>
<td>1.4</td>
<td>1.1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>raytrace-simple</td>
<td>3.3</td>
<td>3.1</td>
<td>2.8</td>
<td>1.4</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>spectral-norm</td>
<td>3.3</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>sympy_str</td>
<td>1.5</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>telco</td>
<td>4</td>
<td>2.5</td>
<td>2.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>polymorphism</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.08</td>
<td>0.09</td>
</tr>
</tbody>
</table>

http://marc.info/?l=pypy-dev&m=141587744128967&w=2
The benchmark suite said 7 levels, so that’s what I suggested
The benchmark suite said 7 levels, so that’s what I suggested

Even though I doubted it was the right global value
Benchmark suites (2)
Benchmark suites (2)

Benchmarks guide our optimisations
Benchmarks guide our optimisations

Are they correct guides?
17 JavaScript benchmarks from V8
17 JavaScript benchmarks from V8

Let’s make each benchmark run for 2000 iterations
Octane: pdf.js explodes

$ d8 run.js
Richards
DeltaBlue
Encrypt
Decrypt
RayTrace
Earley
Boyer
RegExp
Splay
NavierStokes
PdfJS

<--- Last few GCs --->

14907865 ms: Mark-sweep 1093.9 (1434.4) -> 1093.4 (1434.4) MB, 274.8 / 0.0 ms [allocation failure] [GC in old space
14908140 ms: Mark-sweep 1093.4 (1434.4) -> 1093.3 (1434.4) MB, 274.4 / 0.0 ms [allocation failure] [GC in old space
14908421 ms: Mark-sweep 1093.3 (1434.4) -> 1100.5 (1418.4) MB, 280.9 / 0.0 ms [last resort gc].
14908703 ms: Mark-sweep 1100.5 (1418.4) -> 1107.8 (1418.4) MB, 282.1 / 0.0 ms [last resort gc].

<--- JS stacktrace --->

==== JS stack trace =========================================
Security context: 0x20d333ad3ba9 <JS Object>
2: extractFontProgram(aka Type1Parser_extractFontProgram) [pdfjs.js:17004] [pc=0x3a13b275421b] (this=0x3de358283581 <a type1Parser with map 0x1f822131a411>,stream=0x4603fbdc4e1 <an Uint8Array with map 0x393de2707fe1>)
3: new Type1Font [pdfjs.js:17216] [pc=0x3a13b2752078] (this=0x4603fbdaea9 <a Type1Font with map 0x1f822134f7e1>,

# Fatal error in CALL_AND_RETRY_LAST
# Allocation failed - process out of memory
#
zsh: illegal hardware instruction  d8 run.js
Octane: analysing pdf.js

In-process iteration
0.0126
5.9004
11.7882
17.6760
23.5638
29.4516
35.3394

Process execution #1

Time (secs)
Octane: analysing pdf.js

Process execution #1

In-process iteration

-0.2442
2.0974
4.4389
6.7805
9.1221
11.4636
13.8052

Time (secs)
Octane: debugging

```
var pdf_file = "test.pdf";
var canvas_logs = [];

var PdfJS = new BenchmarkSuite("PdfJS", [10124921], [
    new Benchmark("PdfJS", false, false, 24,
        runPdfJS, setupPdfJS, tearDownPdfJS, null, 4)
]);

function runPdfJS() {
    PDFJS.getDocument(pdf_file).then(function(pdf) {
        var canvas = PdfJS_window.document.getElementById('canvas');
        var context = canvas.getContext('2d');
        var renderContext = {canvasContext: context};
        canvas_logs.push(context.__log__);

        // Cycle through all pages.
        function renderPages(i, j) {
            if (i > j) return;
            context.clearRect(0, 0, canvas.width, canvas.height);
            pdf.getPage(i).then(function(page) {
                renderContext.viewport = page.getViewport(i);
                canvas.height = renderContext.viewport.height;
                canvas.width = renderContext.viewport.width;
                page.render(renderContext).then(renderPages.bind(null, i + 1, j));
            });
        }
        renderPages(1, pdf.numPages);
    });
}

// Wait for everything to complete.
PdfJS_window.flushTimeouts();
```
Fix memory leak in pdfjs.js. #42

Open

ltratt wants to merge 2 commits into chromium:master from ltratt:master

Conversation 5  Commits 2  Files changed 1

Changes from all commits ▼  1 file ▼  +1 −0

1  pdfjs.js

```javascript
@@ -43,6 +43,7 @@ function setupPdfJS() {

    + canvas_logs.length = 0;

-    PDFJS.getDocument(pdf_file).then(function(pdf) {
+    var canvas = PdfJS_window.document.getElementById('canvas');
            var context = canvas.getContext('2d');
```
pdfjs isn’t the only problem
pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak
Octane: other issues

pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak

zlib complains that Cannot enlarge memory arrays in asm.js (a memory leak? I don’t know)
Octane: other issues

pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak

zlib complains that Cannot enlarge memory arrays in asm.js (a memory leak? I don’t know)

Timings are made with a non-monotonic microsecond timer
Summary

MORE USERS MORE HAPPY WITH OUR 6-S?

THESIS: BENCHMARKING AND BENCHMARKS ARE PERFORMANCE DESTINY.

HTTP://SOFT-DEV.ORG/
Summary

Why aren’t more users more happy with our VMs?
Why aren’t more users more happy with our VMs?

My thesis: benchmarking and benchmarks are performance destiny.
Why aren’t more users more happy with our VMs?

My thesis: benchmarking *and* benchmarks are performance destiny.

Ours have misled us.
How to benchmark a bit better

1. Unbenchmark for longer to uncover issues.
2. Accept that peak performance may not occur.
3. Always report warm-up time.
4. Avoid over-training on small benchmark suites.
5. Collect more benchmarks.
6. Focus on predictable performance.
1 Run benchmarks for longer to uncover issues.
1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. Always report warmup time.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. Always report warmup time.
4. Stop over-training on small benchmark suites.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. Always report warmup time.
4. Stop over-training on small benchmark suites.
5. Collect more benchmarks.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. Always report warmup time.
4. Stop over-training on small benchmark suites.
5. Collect more benchmarks.
6. Focus on predictable performance.
The big question

...BUT A LOT?

I CAN'T, I HAVE AN IDEA...

40/49 HTTP://SOFT-DEV.ORG/
Can we fix existing VMs?
Can we fix existing VMs?

At least a bit... but a lot? Unclear.
Can we fix existing VMs?

At least a bit... but a lot? Unclear.

In case we can’t, I have an idea...
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
instr = load_instruction(program_counter)
if instr == INSTR_VAR_GET:
    stack.push(
        vars[read_var_name_from_instruction()])
    program_counter += 1
elif instr == INSTR_VAR_SET:
    vars[read_var_name_from_instruction()]
    = stack.pop()
    program_counter += 1
elif instr == INSTR_INT:
    stack.push(read_int_from_instruction())
    program_counter += 1
elif instr == INSTR_LESS_THAN:
    rhs = stack.pop()
    lhs = stack.pop()
    if isinstance(lhs, int) and isinstance(rhs, int):
        if lhs < rhs:
            stack.push(True)
        else:
            stack.push(False)
    else: ...
    program_counter += 1
elif instr == INSTR_IF:
    result = stack.pop()
    if result == True:
        program_counter += 1
    else:
        program_counter +=
        read_jump_if_instruction()
elif instr == INSTR_ADD:
    lhs = stack.pop()
    rhs = stack.pop()
    if isinstance(lhs, int)
    and isinstance(rhs, int):
        stack.push(lhs + rhs)
    else: ...
    program_counter += 1
Meta-tracing JITs

**FL Interpreter**

```python
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()]
        )
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
        = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
        program_counter += 1
```

41 / 49
**Meta-tracing JITs**

---

**FL Interpreter**

```python
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
        = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
        program_counter += 1
```

**User program (lang FL)**

```python
if x < 0:
    x = x + 1
else:
    x = x + 2
x = x + 3
```

---

**Software Development Team**

41 / 49

http://soft-dev.org/
**Meta-tracing JITs**

**FL Interpreter**

```python
program_counter = 0; stack = []
vars = {...}
while True:
    jit_merge_point(program_counter)
    instr = load_instruction(program_counter)
    if instr == INSTR_VAR_GET:
        stack.push(
            vars[read_var_name_from_instruction()])
        program_counter += 1
    elif instr == INSTR_VAR_SET:
        vars[read_var_name_from_instruction()]
        = stack.pop()
        program_counter += 1
    elif instr == INSTR_INT:
        stack.push(read_int_from_instruction())
        program_counter += 1
    elif instr == INSTR_LESS_THAN:
        rhs = stack.pop()
        lhs = stack.pop()
        if isinstance(lhs, int) and isinstance(rhs, int):
            if lhs < rhs:
                stack.push(True)
            else:
                stack.push(False)
        else: ...
        program_counter += 1
```

**Initial trace**

```python
v0 = <program_counter>
v1 = <stack>
v2 = <vars>
v3 = load_instruction(v0)
guard_eq(v3, INSTR_VAR_GET)
v4 = dict_get(v2, "x")
list_append(v1, v4)
v5 = add(v0, 1)
v6 = load_instruction(v5)
guard_eq(v6, INSTR_INT)
list_append(v1, 0)
v7 = add(v5, 1)
v8 = load_instruction(v7)
guard_eq(v8, INSTR_LESS_THAN)
v9 = list_pop(v1)
v10 = list_pop(v1)
guard_type(v9, int)
guard_type(v10, int)
guard_not_less_than(v9, v10)
list_append(v1, False)
v11 = add(v7, 1)
v12 = load_instruction(v11)
guard_eq(v12, INSTR_IF)
v13 = list_pop(v1)
guard_false(v13)
...```
Meta-tracer states

- **Interpreter**
- **Tracer**
- **Machine code**
- **Blackhole interpreter**

States:
- **Hot**
- **Compile**
- **Safepoint**
- **Guard failure**
Meta-tracer states

Interpreter → Tracer → Machine code → Blackhole interpreter → Safepoint → Interpreter

Hot → Compile → Guard failure
Meta-tracer states

Interpreter

Tracer

Machine code

Blackhole interpreter

Safepoint

Guard failure

Compile

Hot
Meta-tracer states

Interpreter → Tracer → Machine code

1. Hot
2. Compile
3. Safepoint
4. Guard failure

Blackhole interpreter
Meta-tracer states

Interpreter | Tracer | Machine code

Hot | Compile

Safepoint | Guard failure

Blackhole interpreter

Guard failure

Interpreter Tracer Machine code

Blackhole interpreter

42 / 49

http://soft-dev.org/
Meta-tracer states

Interpreter

Tracer

Machine code

Hot

Compile

Safepoint

Guard failure

Blackhole interpreter
Meta-tracer states

Interpreter -> Tracer

Compiled code

Guard failure

Safepoint

Blackhole interpreter

Hot

Compile
Meta-tracer performance (now)

Diagram:
- Interpreter
- Tracer
- Machine code
- Blackhole interpreter

Arrows:
- Hot
- Compile
- Safepoint
- Guard failure

Note: 1x
Meta-tracer performance (now)

Interpreter  Tracer  Machine code

1x  0.1x

Hot  Compile

Guard failure  Safepoint

Blackhole interpreter

http://soft-dev.org/
Meta-tracer performance (now)

Interpreter ➔ Hot ➔ Tracer ➔ Compile ➔ Machine code ➔ Guard failure ➔ Blackhole interpreter ➔ Safepoint ➔ Interpreter

1x ➔ 200x ➔ 0.1x

43 / 49
Meta-tracer performance (our aim)

Interpreter → Tracer → Machine code

- **Interpreter**: Hot (1x)
- **Tracer**: Hot (2x)
- **Machine code**: Compile (0.1x)

- **Blackhole interpreter**: Safepoint
- **Guard failure**:
VM Warmup Blows Hot and Cold
E. Barrett, C. F. Bolz, R. Killick, V. Knight, S. Mount and L. Tratt.

Rigorous Benchmarking in Reasonable Time
T. Kalibera and R. Jones

Specialising Dynamic Techniques for Implementing the Ruby Programming Language
C. Seaton (Chapter 4)

Quantifying performance changes with effect size confidence intervals
T. Kalibera and R. Jones
Thanks

- EPSRC: COOLER and Lecture.
- Oracle.
- Cloudflare.
Thanks for listening
How long to run things for

% similarity to n=2000

Classifications
Steady iteration (# or s)
Steady performance (s)
Overall

In-process iterations

Software Development Team

48 / 49

http://soft-dev.org/
How long to run things for
## Comparing Results

<table>
<thead>
<tr>
<th>Class</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarytrees</td>
<td>(13, 2)</td>
<td>1.0</td>
<td>0.00</td>
</tr>
<tr>
<td>capnproto_decode</td>
<td></td>
<td>4.0</td>
<td>0.11</td>
</tr>
<tr>
<td>capnproto_encode</td>
<td>⊥</td>
<td>2.0</td>
<td>0.13</td>
</tr>
<tr>
<td>famkuch Redux</td>
<td>⊥</td>
<td>2.0</td>
<td>0.13</td>
</tr>
<tr>
<td>fasta</td>
<td></td>
<td>6.36</td>
<td>0.15</td>
</tr>
<tr>
<td>jsonlua_decode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jsonlua_encode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>luacheck</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>luacheck parser</td>
<td>⊗</td>
<td>42.0</td>
<td></td>
</tr>
<tr>
<td>luafun</td>
<td>(14, 1)</td>
<td>6.36</td>
<td>0.15</td>
</tr>
<tr>
<td>md5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nbody</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>richards</td>
<td>⊥</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>series</td>
<td>⊥</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>spectralnorm</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>