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

KING’S College
LONDON

Software Development Team
2018-09-26
JVMs bring "gcc -O2" to the masses

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

![Diagram showing iteration time vs. in-process iteration with peak performance and compilation points highlighted.](http://soft-dev.org/)
What do VM claims pertain to?

![Graph showing iteration time versus in-process iteration with a warmup phase.]
Users *always* perceive warmup
Users *always* perceive warmup

Maybe we should know how long it is?
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)
Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

We removed any CFG non-determinism
Method 1: Which benchmarks?

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
- TruffleRuby 20170502
- 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, smtprd)
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
- Reruns any proc. exec where the CPU was throttled
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)
The experiment has gone through many versions.
The experiment has gone through many versions.

The following data is from the 1.5 run.
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD\textsuperscript{4790}, Proc. exec. #14 (warmup)

\begin{figure}
\centering
\includegraphics[width=\textwidth]{chart.png}
\end{figure}
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD

Time (secs)

In-process iteration

0.56745
0.56825
0.56905
0.56984
0.57064
0.57143
0.57223

12/51 HTTP://SOFT-DEV.ORG/
Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #14 (warmup)
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD4790, Proc. exec. #14 (warmup)

Changepoint
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD_{4790}, Proc. exec. #14 (warmup)

Changepoint segment

Changepoint
Warmup & flat (1)

N-Body, PyPy, Linux, Proc. exec. #24 (flat)

Time (secs)

In-process iteration

1.79372
1.80244
1.81116
1.81988
1.82860
1.83731
1.84603
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: flat
- Final seg is in fastest set: warmup
Warmup & flat (2)

Fasta, V8, Linux, Proc. exec. #15 (warmup)

In-process iteration

Time (secs)

1.12811
1.13248
1.13685
1.14121
1.14558
1.14995
1.15432

14/51 HTTP://SOFT-DEV.ORG/
Warmup & flat (2)

Spectral Norm, PyPy, Linux\textsubscript{E3−1240v5}, Proc. exec. #5 (warmup)

In-process iteration

Time (secs)

0.46470
0.46618
0.46766
0.46915
0.47063
0.47211
0.47359

0.46503
0.46939
0.47374

1 201 401 601 801 1001 1201 1401 1601 1801 2000
Richards, HotSpot, Linux_{E3-1240v5}, Proc. exec. #8 (slowdown)
Classification algorithm (steps in order):

All segs are equivalent: flat

Final seg is in fastest set: warmup
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

Fasta, V8, Linux

Time (secs)

Proc. exec. #26 (slowdown)

17/51 HTTP://SOFT-DEV.ORG/
Binary Trees, V8, Linux$_{4790}$, Proc. exec. #6 (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: *flat*

Final seg is in fastest set: *warmup*

Final seg is not in fastest set: *slowdown*

Else: *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 (3)

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}

Bad
Warmup or no steady state?

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

Time (secs)

In-process iteration
Inconsistent Process-executions

Binary Trees, V8, Linux4790, Proc. exec. #15 (warmup)

Binary Trees, V8, Linux4790, Proc. exec. #6 (no steady state)

(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 perf (s)</th>
<th>Class</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>1063.0 (283, 8, 11)</td>
<td>2.61136</td>
<td>Graal</td>
<td>0.057777</td>
</tr>
<tr>
<td>Graal</td>
<td>228.0 (1, 5)</td>
<td>0.46227</td>
<td>HIIVM</td>
<td>1.0314</td>
</tr>
<tr>
<td>HIIVM</td>
<td>2621.96 (749, 853, 554)</td>
<td>4.80364</td>
<td>HotSpot</td>
<td>1.0</td>
</tr>
<tr>
<td>HotSpot</td>
<td>106.34 (0.600, 290, 406)</td>
<td>0.003335</td>
<td>LuaJIT</td>
<td>0.00</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>1003.0 (283, 8, 11)</td>
<td>2.61136</td>
<td>PyPy</td>
<td>0.00</td>
</tr>
<tr>
<td>PyPy</td>
<td>228.0 (1, 5)</td>
<td>0.46227</td>
<td>TruffleRuby</td>
<td>0.00</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>2621.96 (749, 853, 554)</td>
<td>4.80364</td>
<td>V8</td>
<td>0.00</td>
</tr>
<tr>
<td>V8</td>
<td>106.34 (0.600, 290, 406)</td>
<td>0.003335</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 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>C</td>
<td>≈ (27lasses, 2−, 1Γ)</td>
<td>775.0 (1.5,780.0)</td>
<td>425.16 (0.246,426.809)</td>
</tr>
<tr>
<td>Graal</td>
<td>⊥</td>
<td>14.0 (2.0,94.6)</td>
<td>13.60 (0.830,98.737)</td>
</tr>
<tr>
<td>HHVM</td>
<td>≈ (29lasses, 1ω)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>⊥</td>
<td>7.0 (7.0,7.5)</td>
<td>1.91 (1.902,3.645)</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>−</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>≈ (27−, 3lasses)</td>
<td>1.0 (1.0,45.2)</td>
<td>0.00 (0.000,20.597)</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>≈ (25Γ, 5ω)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>⊥</td>
<td>3.0 (3.0,3.0)</td>
<td>0.52 (0.523,0.526)</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>\langle VM, benchmark \rangle pairs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>⊴</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>⊤</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>⋆</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>=</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>≠</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
<tr>
<td></td>
<td>Process executions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>−</td>
<td>22.0%</td>
<td>23.3%</td>
<td>37.7%</td>
</tr>
<tr>
<td>⊴</td>
<td>48.3%</td>
<td>43.9%</td>
<td>35.2%</td>
</tr>
<tr>
<td>⊤</td>
<td>20.1%</td>
<td>22.1%</td>
<td>12.1%</td>
</tr>
<tr>
<td>⋆</td>
<td>9.6%</td>
<td>10.8%</td>
<td>15.0%</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>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\downarrow)</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>(\uparrow)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>(\sim)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(=)</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\neq)</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

### \(\langle\text{VM, benchmark}\rangle\) pairs

### Process executions

<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>22.0%</td>
<td>23.3%</td>
<td>37.7%</td>
</tr>
<tr>
<td>(\downarrow)</td>
<td>48.3%</td>
<td>43.9%</td>
<td>35.2%</td>
</tr>
<tr>
<td>(\uparrow)</td>
<td>20.1%</td>
<td>22.1%</td>
<td>12.1%</td>
</tr>
<tr>
<td>(\sim)</td>
<td>9.6%</td>
<td>10.8%</td>
<td>15.0%</td>
</tr>
</tbody>
</table>
Summary

Classical warmup occurs for only:
Classical warmup occurs for only:

67.2%–70.3% of process executions
Classical warmup occurs for only:

67.2%–70.3% of process executions

37.8%–40.0% of (VM, benchmark) pairs
Summary

Classical warmup occurs for only:

67.2%–70.3% of process executions

37.8%–40.0% of (VM, benchmark) pairs

12.5% of benchmarks for (VM, benchmark, machine) triples
Are odd effects correlated with compilation and GC?

Fasta, PyPy, Linux\textsubscript{E3 − 1240v5}, Proc. exec. #2 (no steady state)

- GC
- JIT
- Time (secs)

In-process iteration

In-process iteration

1 201 401 601 801 1001 1201 1401 1601 1801 2000

0.00000
0.00723
0.01446

0.759 \times 10^9
1.491 \times 10^9
2.222 \times 10^9

2.17580
2.25557
2.33534
2.41511
2.49487
2.57464
2.65441

0.00000
0.00723
0.01446

JIT

1 201 401 601 801 1001 1201 1401 1601 1801 2000

0.00000
0.00723
0.01446

0.759 \times 10^9
1.491 \times 10^9
2.222 \times 10^9

2.17580
2.25557
2.33534
2.41511
2.49487
2.57464
2.65441

25 / 51

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

Richards, HotSpot, Linux$_{4790}$, Proc. exec. #4 (slowdown)

In-process iteration

Time (secs)

JIT (secs)

GC (secs)

Richards, HotSpot, Linux$_{4790}$, Proc. exec. #4 (slowdown)

0.00000
0.01703
0.03407

JIT (secs)

Richards, HotSpot, Linux$_{4790}$, Proc. exec. #4 (slowdown)

0.00000
0.01703
0.03407

JIT (secs)

Richards, HotSpot, Linux$_{4790}$, Proc. exec. #4 (slowdown)

0.00000
0.01703
0.03407

JIT (secs)
Are odd effects correlated with compilation and GC?

Fannkuch Redux, HotSpot, Linux4790, Proc. exec. #3 (no steady state)
Benchmark suites
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
A war story

Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion
Symptom: poor performance of a Pyston benchmark on PyPy

Cause: RPython traces recursion

Fix: Check for recursion before tracing
diff --git a/rpython/jit/metainterp/pyjitpl.py b/rpython/jit/metainterp/pyjitpl.py
--- a/rpython/jit/metainterp/pyjitpl.py
+++ b/rpython/jit/metainterp/pyjitpl.py
@@ -951,9 +951,31 @@

  + # 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.metainterp.perform_call(portal_code, allboxes,
- greenkey=greenboxes)
+  + inline = True
+  + if self.metainterp.is_main_jitcode(portal_code):
+  +     for gk, _ in self.metainterp.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.metainterp.perform_call(portal_code, allboxes,
+  + greenkey=greenboxes)
A war story: mixed fortunes

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
Benchmarks guide our optimisations
Benchmark suites (2)

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
$ 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>,name=0x4603fbd9c09 <String[12]: JTYMKN+CMR17>,file=0x4603fb...)

# 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

0.0205
0.0679
0.1152
0.1625
0.2098
0.2571
0.3045

Time (secs)

Process execution #1

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(1);
                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

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

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

```
1 pdfjs.js

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

+    function runPdfJS() {
+        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
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)
pdfjs isn’t the only problem

CodeLoadClosure also has a memory leak

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

Timings are made with a non-monotonic microsecond timer
Summary

HyAREN MORE USERS MORE HAPPY WITH OUR 6-S?

THESIS: BENCHMARKING and BENCHMARKS ARE PERFORMANCE DESTINY.
Why aren’t more users more happy with our VMs?
Why aren’t more users more happy with our VMs?

My thesis: benchmarking \textit{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 neither peak performance nor steady state may occur.
3. Always report warmup 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 neither peak performance or a steady state may occur.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that neither peak performance or a steady state may occur.
3. Always report warmup time.
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
2. Accept that neither peak performance or a steady state may 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 neither peak performance or a steady state may 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 neither peak performance or a steady state may occur.
3. Always report warmup time.
4. Stop over-training on small benchmark suites.
5. Collect more benchmarks.
6. Focus on predictable performance.
Can we fix existing VMs?
Can we fix existing VMs?

At least a bit... but a lot? Unclear.
The big question

Can we fix existing VMs?

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

In case we can’t, I have an idea...
**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
    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
```

---

42 / 51 [HTTP://SOFT-DEV.ORG/]
Meta-tracing JIts

**FL Interpreter**

```
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
```
### FL Interpreter

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>program_counter = 0; stack = []</code></td>
<td>Initialize program counter and stack</td>
</tr>
<tr>
<td><code>vars = {...}</code></td>
<td>Initialize variables</td>
</tr>
<tr>
<td><code>while True:</code></td>
<td>Start infinite loop</td>
</tr>
<tr>
<td><code>jit_merge_point(program_counter)</code></td>
<td>Function to merge points</td>
</tr>
<tr>
<td><code>instr = load_instruction(program_counter)</code></td>
<td>Load instruction</td>
</tr>
<tr>
<td><code>if instr == INSTR_VAR_GET:</code></td>
<td>Get variable from instruction</td>
</tr>
<tr>
<td><code>    stack.push(vars[read_var_name_from_instruction()])</code></td>
<td>Push variable to stack</td>
</tr>
<tr>
<td><code>    program_counter += 1</code></td>
<td>Increment program counter</td>
</tr>
<tr>
<td><code>elif instr == INSTR_VAR_SET:</code></td>
<td>Set variable from instruction</td>
</tr>
<tr>
<td><code>    vars[read_var_name_from_instruction()] = stack.pop()</code></td>
<td>Set variable to value</td>
</tr>
<tr>
<td><code>    program_counter += 1</code></td>
<td>Increment program counter</td>
</tr>
<tr>
<td><code>elif instr == INSTR_INT:</code></td>
<td>Load integer from instruction</td>
</tr>
<tr>
<td><code>    stack.push(read_int_from_instruction())</code></td>
<td>Push integer to stack</td>
</tr>
<tr>
<td><code>    program_counter += 1</code></td>
<td>Increment program counter</td>
</tr>
<tr>
<td><code>elif instr == INSTR_LESS_THAN:</code></td>
<td>Less than comparison</td>
</tr>
<tr>
<td><code>    rhs = stack.pop()</code></td>
<td>Pop right-hand side value</td>
</tr>
<tr>
<td><code>    lhs = stack.pop()</code></td>
<td>Pop left-hand side value</td>
</tr>
<tr>
<td><code>    if isinstance(lhs, int) and isinstance(rhs, int):</code></td>
<td>Check if both values are integers</td>
</tr>
<tr>
<td><code>        if lhs &lt; rhs:</code></td>
<td>Compare values</td>
</tr>
<tr>
<td><code>            stack.push(True)</code></td>
<td>Push True to stack</td>
</tr>
<tr>
<td><code>        else:</code></td>
<td>Compare values</td>
</tr>
<tr>
<td><code>            stack.push(False)</code></td>
<td>Push False to stack</td>
</tr>
<tr>
<td><code>    else:</code></td>
<td>Handle non-integer values</td>
</tr>
<tr>
<td><code>        ...</code></td>
<td>Handle non-integer values</td>
</tr>
<tr>
<td><code>    program_counter += 1</code></td>
<td>Increment program counter</td>
</tr>
</tbody>
</table>

### User program (lang FL)

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

```python
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

URL: HTTP://SOFT-DEV.ORG/
Meta-tracer states

Interpreter

Tracer

Machine code

Blackhole interpreter

Hot

Compile

Safepoint

Guard failure

43 / 51

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

- Interpreter
- Tracer
- Machine code
- Blackhole interpreter
- Hot
- Compile
- Guard failure
- Safepoint
Meta-tracer states

Interpreter → Tracer → Machine code

Interpreted code → Blackhole interpreter

Guard failure → Safepoint

Hot → Compile
Meta-tracer states

Interpreter → Tracer → Machine code

Hot → Compile

Safepoint → Guard failure

Blackhole interpreter
Meta-tracer states

Interpreter

Tracer

Machine code

Hot

Compile

Guard failure

Safepoint

Blackhole interpreter

Interpreter Tracer Machine
code
Blackhole

 Interpreter
Tracer
Machine code

Hot
Compile
Guard failure
Safepoint
Blackhole interpreter

Interpreter Tracer Machine
code
Blackhole

Interpreter Tracer Machine
code
Blackhole

Interpreter Tracer Machine
code
Blackhole
Meta-tracer states

Interpreter -> Tracer -> Machine code

- Hot
- Compile
- Guard failure
- Safepoint

Blackhole interpreter
Meta-tracer states

Interpreter

Tracer

Machine code

Blackhole interpreter

Hot

Compile

Safepoint

Guard failure

43 / 51 HTTP://SOFT-DEV.ORG/
Meta-tracer performance (now)

Interpreter → Tracer → Machine code

Blackhole interpreter

1x

Hot

Compile

Safepoint

Guard failure

Software Development Team

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

Interpreter → Tracer (200x) → Machine code (0.1x) → Blackhole interpreter (1x) → Interpreter

- Hot
- Compile
- Safepoint
- Guard failure

HTTP://SOFT-DEV.ORG/
Meta-tracer performance (our aim)

Interpreter ➔ Tracer ➔ Machine code

1x ➔ Hot ➔ 2x ➔ Compile ➔ 0.1x

Safepoint ➔ Blackhole interpreter ➔ Guard failure

http://soft-dev.org/
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
| **warmup_stats** | Use our statistical method on your VMs  
|                  | [http://soft-dev.org/src/warmup_stats/](http://soft-dev.org/src/warmup_stats/) |
| **Krun**         | Run experiments in a controlled manner  
|                  | [http://soft-dev.org/src/krun/](http://soft-dev.org/src/krun/) |
• EPSRC: COOLER and Lecture.
• Oracle.
• Cloudflare.
Thanks for listening

Richards, HotSpot, Linux, Proc. exec. #8 (slowdown)

Binary Trees, V8, Linux, Proc. exec. #6 (no steady state)
How long to run things for (0.8)

In-process iterations

% similarity to n=2000

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

50/51 HTTP://SOFT-DEV.ORG/
How long to run things for (0.8)
## Diffing results (0.8 → 1.5)

<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>0</td>
<td>0</td>
<td>0.16188</td>
</tr>
<tr>
<td>fannkuch Redux</td>
<td>(27μ, 2μ, 1ω)</td>
<td>(3.0, 0.7, 0.4)</td>
<td>(0.535, 5.873)</td>
</tr>
<tr>
<td>fasta</td>
<td>4.0</td>
<td>0.75</td>
<td>0.16188</td>
</tr>
<tr>
<td></td>
<td>(3.0, 0.7, 0.4)</td>
<td>(0.535, 5.873)</td>
<td>±0.000738</td>
</tr>
<tr>
<td>nbody</td>
<td>6.0</td>
<td>0.86</td>
<td>0.13677</td>
</tr>
<tr>
<td></td>
<td>(5.0, 0.7, 0.0)</td>
<td>(0.704, 1.090)</td>
<td>±0.000343</td>
</tr>
<tr>
<td>richards</td>
<td>2.0</td>
<td>0.95</td>
<td>0.26465</td>
</tr>
<tr>
<td></td>
<td>(2.0, 0.353)</td>
<td>(0.879, 0.745)</td>
<td>±0.007761</td>
</tr>
<tr>
<td>spectralnorm</td>
<td>14.0</td>
<td>13.60</td>
<td>1.05685</td>
</tr>
<tr>
<td></td>
<td>(2.0, 0.946)</td>
<td>(0.839, 0.737)</td>
<td>±0.000126</td>
</tr>
</tbody>
</table>