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
2018-05-17
JVMs bring "gcc -O2" to the masses

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

![Graph showing iteration time vs. in-process iteration.](image-url)
What do VM claims pertain to?
What do VM claims pertain to?

![Graph showing iteration time vs. in-process iteration with a peak labeled Profiling Interpreter.](http://soft-dev.org/)
What do VM claims pertain to?

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

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

- Compilation
- Profiling Interpreter
- Peak Performance
What do VM claims pertain to?
Users *always* perceive warmup
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.
Method 1: Which benchmarks?

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

- Linux\textsubscript{4790}, Debian 8, 24GiB RAM
- Linux\textsubscript{E3-1240\,v5}, 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
- 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.
Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #14 (warmup)
Warmup & flat (1)

Fannkuch Redux, LuaJIT, OpenBSD⁴⁷⁹⁰, Proc. exec. #14 (warmup)
Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #14 (warmup)
Warmup & flat (1)

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

Changepoint
Warmup & flat (1)

Changepoint segment

Fankuch Redux, LuaJIT, OpenBSD_{4790}, Proc. exec. #14 (warmup)
Warmup & flat (1)

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

Time (secs)

1.79372
1.80244
1.81116
1.81988
1.82860
1.83731
1.84603

In-process iteration

12/51 HTTP://SOFT-DEV.ORG/
Method 7: Classification

Classification algorithm (steps in order):

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

All segs are equivalent: flat

Final seg is in fastest set: warmup
Warmup & flat (2)

Fasta, V8, Linux\textsubscript{4790}, Proc. exec. #15 (warmup)

\begin{center}
\begin{tikzpicture}
\begin{axis}[
    xtick={0,100,200,300,400,500},
    xticklabels={1,201,401,601,801,1001},
    xlabel={In-process iteration},
    ylabel={Time (secs)},
    xmin=0, xmax=2000,
    ymin=1.12811, ymax=1.15432
]
\addplot[blue] table [y=Fasta, V8, Linux\textsubscript{4790}, Proc. exec. #15 (warmup)];
\end{axis}
\end{tikzpicture}
\end{center}
Spectral Norm, PyPy, Linux$_{E3-1240v5}$, Proc. exec. #5 (warmup)

In-process iteration

Time (secs)

1 201 401 601 801 1001 1201 1401 1601 1801 2000

0.46470
0.46618
0.46766
0.46915
0.47063
0.47211
0.47359

0.46503
0.46939
0.47374
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*
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
1.14087
1.14754
1.15420

Time (secs)
Fasta, V8, Linux4790, Proc. exec. #26 (slowdown)

1 26 50

1.16160
1.15818
1.15476
1.15134
1.14793
1.14451
1.14109

In-process iteration

1 201 401 601 801 1001 1201 1401 1601 1801 2000
No steady state (1)

Binary Trees, V8, Linux_{4790}, Proc. exec. #6 (no steady state)

In-process iteration

0.49089
0.49541
0.49993
0.50444
0.50896
0.51348
0.51800

Time (secs)

0.51800
0.51348
0.50896
0.50444
0.49993
0.49541
0.49089

1 201 401 601 801 1001 1201 1401 1601 1801 2000

In-process iteration

http://soft-dev.org/
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

19/51 HTTP://SOFT-DEV.ORG/
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 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)

In-process iteration

Time (secs)
Inconsistent Process-executions

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

Binary Trees, V8, Linux, 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 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>20</td>
<td>1.0</td>
<td>0.00</td>
<td>Graal</td>
<td>25</td>
<td>0.1338</td>
<td>1.0492</td>
</tr>
<tr>
<td>Graal</td>
<td>(31, 50, 1)</td>
<td>(1.0, 536.9)</td>
<td>...</td>
<td>V8</td>
<td>(19, 82)</td>
<td>(0.6, 0.3)</td>
<td>0.00310</td>
</tr>
<tr>
<td>HHVM</td>
<td>(25, 10, 2)</td>
<td>(1.0, 1120.3)</td>
<td>0.50</td>
<td>PyPy</td>
<td>(32, 67)</td>
<td>(20, 10)</td>
<td>0.19677</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(27, 42)</td>
<td>(15, 29, 1)</td>
<td>1.0</td>
<td>TruffleRuby</td>
<td>(281, 27)</td>
<td>(20, 10)</td>
<td>0.21648</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(25, 3, 1)</td>
<td>(15, 9, 84)</td>
<td>0.46</td>
<td>Richards</td>
<td>(28, 1, 1)</td>
<td>(18, 7, 5)</td>
<td>0.2665</td>
</tr>
<tr>
<td>V8</td>
<td>(15, 8, 1)</td>
<td>(1.0, 536.9)</td>
<td>0.00</td>
<td>PyPy</td>
<td>(28, 1, 1)</td>
<td>(18, 7, 5)</td>
<td>0.2665</td>
</tr>
</tbody>
</table>

**Note:**
- Steady iter (#) represents the number of iterations.
- Steady iter (s) represents the time taken in seconds.
- Steady perf (s) represents the performance in seconds.

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

<table>
<thead>
<tr>
<th></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>( \times ) (27( \downarrow ), 2( \leftarrow ), 1( \Gamma ))</td>
<td>775.0</td>
<td>425.16</td>
<td>0.54581</td>
</tr>
<tr>
<td>Graal</td>
<td>( \leftarrow )</td>
<td>14.0</td>
<td>13.60</td>
<td>1.05685</td>
</tr>
<tr>
<td>HHVM</td>
<td>( \times ) (29( \downarrow ), 1( \omega ))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>( \leftarrow )</td>
<td>7.0</td>
<td>1.91</td>
<td>0.31472</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>( = )</td>
<td>1.0</td>
<td>0.00</td>
<td>0.46480</td>
</tr>
<tr>
<td>PyPy</td>
<td>( \times ) (27( \leftarrow ), 3( \downarrow ))</td>
<td>1.0</td>
<td>0.00</td>
<td>0.46480</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>( \times ) (25( \Gamma ), 5( \omega ))</td>
<td>3.0</td>
<td>0.52</td>
<td>0.25362</td>
</tr>
<tr>
<td>V8</td>
<td>( \leftarrow )</td>
<td>3.0</td>
<td>0.52</td>
<td>0.25362</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>(VM, benchmark) pairs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-)</td>
<td>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\perp)</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>(\lhd)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>(\bowtie)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\not\equiv)</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
<tr>
<td>Process executions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-)</td>
<td>22.0%</td>
<td>23.3%</td>
<td>37.7%</td>
</tr>
<tr>
<td>(\perp)</td>
<td>48.3%</td>
<td>43.9%</td>
<td>35.2%</td>
</tr>
<tr>
<td>(\lhd)</td>
<td>20.1%</td>
<td>22.1%</td>
<td>12.1%</td>
</tr>
<tr>
<td>(\bowtie)</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} †</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8.9%</td>
<td>11.1%</td>
<td>13.3%</td>
</tr>
<tr>
<td>1</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>2</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>3</td>
<td>4.4%</td>
<td>4.4%</td>
<td>0.0%</td>
</tr>
<tr>
<td>4</td>
<td>11.1%</td>
<td>8.9%</td>
<td>13.3%</td>
</tr>
<tr>
<td>5</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

\[\text{⟨VM, benchmark⟩ pairs}\]

<table>
<thead>
<tr>
<th>Process executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
</tr>
<tr>
<td>⊥</td>
</tr>
<tr>
<td>⊤</td>
</tr>
<tr>
<td>¬</td>
</tr>
</tbody>
</table>
Classical warmup occurs for only:
Summary

Classical warmup occurs for only:

67.2%–70.3% of process executions
Summary

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_{E3 − 1240v5}, Proc. exec. #4 (no steady state)

- Time (secs)
  - 0.756 × 10^9
  - 1.501 × 10^9
  - 2.246 × 10^9

- JIT
  - -0.00003
  - 0.00725
  - 0.01453

- GC

- In-process iteration
  - 1 201 401 601 801 1001 1201 1401 1601 1801 2000

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

Richards, HotSpot, Linux\textsubscript{E3-1240v5}, Proc. exec. #3 (slowdown)

In-process iteration

Time (secs)

Richards, HotSpot, Linux\textsubscript{E3-1240v5}, Proc. exec. #3 (slowdown)

JIT (secs)

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

Fannkuch Redux, HotSpot, Linux E3-1240v5, Proc. exec. #4 (slowdown)

In-process iteration

Time (secs)

GC (secs)

JIT (secs)

http://soft-dev.org/
Benchmarks guide our optimisations
Benchmarks guide our optimisations

Are they complete guides?
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
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 @@
     if warmrunnerstate.inlining:
         if warmrunnerstate.can_inline_callable(greenboxes):
             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.metainterp.perform_call(portal_code, allboxes,
-                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
+                        i = 0
+                        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>
</tbody>
</table>

polymorphism   | 0.07| 0.07| 0.07| 0.07| 0.08| 0.09|

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

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)
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
$ 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=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

Process execution #1

In-process iteration

Time (secs)

In-process iteration

0 11 23 34 46 57 68 80 91 103 114
In-process iteration

Time (secs)

Process execution #1

In-process iteration

Time (secs)
Octane: debugging

```javascript
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.setTimeout(() => {
    pdfjs.js
  }, 0);
}
```
Fix memory leak in pdfjs.js. #42

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

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

1 pdfjs.js

```javascript
// -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)
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

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. Don’t benchmark 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.
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.
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.
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.
The big question

CAN WE

1. EXISTING 6-S?

AT LEAST A BIT... BUT A LOT?

INCLEAR.

IN CASE WE CAN’ T,

I HAVE AN IDEA...

41/51 HTTP://SOFT-DEV.ORG/
Can we fix existing VMs?
The big question

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...
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
    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 isistance(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
```
### Meta-tracing JITs

<table>
<thead>
<tr>
<th>FL Interpreter</th>
<th>User program (lang FL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>program_counter = 0; stack = []</td>
<td>if x &lt; 0:</td>
</tr>
<tr>
<td>vars = {...}</td>
<td>x = x + 1</td>
</tr>
<tr>
<td>while True:</td>
<td>else:</td>
</tr>
<tr>
<td>jit_merge_point(program_counter)</td>
<td>x = x + 2</td>
</tr>
<tr>
<td>instr = load_instruction(program_counter)</td>
<td>x = x + 3</td>
</tr>
<tr>
<td>if instr == INSTR_VAR_GET:</td>
<td></td>
</tr>
<tr>
<td>stack.push(</td>
<td></td>
</tr>
<tr>
<td>vars[read_var_name_from_instruction()])</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_VAR_SET:</td>
<td></td>
</tr>
<tr>
<td>vars[read_var_name_from_instruction()] = stack.pop()</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_INT:</td>
<td></td>
</tr>
<tr>
<td>stack.push(read_int_from_instruction())</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>elif instr == INSTR_LESS_THAN:</td>
<td></td>
</tr>
<tr>
<td>rhs = stack.pop()</td>
<td></td>
</tr>
<tr>
<td>lhs = stack.pop()</td>
<td></td>
</tr>
<tr>
<td>if isinstance(lhs, int) and isinstance(rhs, int):</td>
<td></td>
</tr>
<tr>
<td>if lhs &lt; rhs:</td>
<td></td>
</tr>
<tr>
<td>stack.push(True)</td>
<td></td>
</tr>
<tr>
<td>else:</td>
<td></td>
</tr>
<tr>
<td>stack.push(False)</td>
<td></td>
</tr>
<tr>
<td>else: ...</td>
<td></td>
</tr>
<tr>
<td>program_counter += 1</td>
<td></td>
</tr>
<tr>
<td>if x &lt; 0:</td>
<td></td>
</tr>
<tr>
<td>x = x + 1</td>
<td></td>
</tr>
<tr>
<td>else:</td>
<td></td>
</tr>
<tr>
<td>x = x + 2</td>
<td></td>
</tr>
<tr>
<td>x = x + 3</td>
<td></td>
</tr>
</tbody>
</table>
### 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
Tracer → Machine code
Machine code → Blackhole interpreter
Blackhole interpreter → Interpreter

Hot → Compile
Compile → Guard failure
Guard failure → Safepoint
Safepoint → Hot

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

Interpreter

Tracer

Machine code

Blackhole interpreter

Guard failure

Compile

Hot

Safepoint
Meta-tracer states

Interpreter → Tracer → Machine code

- Hot
- Compile
- Guard failure
- Safepoint
- Blackhole interpreter
Meta-tracer states

- Interpreter
- Tracer
- Machine code

States:
- Hot
- Compile

Transitions:
- Safepoint
- Guard failure
Meta-tracer states

Interpreter → Tracer → Machine code

Hot

Compile

Guard failure

Safepoint

Blackhole interpreter
Meta-tracer states

- Interpreter
- Tracer
- Machine code

- Hot
- Compile
- Guard failure
- Safepoint

- Blackhole interpreter
Meta-tracer states

- Interpreter
- Tracer
- Machine code
- Blackhole interpreter

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

Interpreter → Tracer → Machine code

Blackhole interpreter

Hot → Compile

Guard failure → Safepoint
Meta-tracer performance (now)

Interpreter → Tracer → Machine code

1x

Hot

Compile

Safe point

Guard failure

Blackhole interpreter
Meta-tracer performance (now)

- Interpreter
- Tracer
- Machine code
- Blackhole interpreter

1x -> Hot -> Tracer

0.1x -> Compile -> Machine code

Safepoint -> Guard failure
Meta-tracer performance (now)

- **Interpreter**
  - 1x
  - Hot
  - Safepoint

- **Tracer**
  - 200x

- **Blackhole interpreter**
  - Guard failure

- **Machine code**
  - 0.1x
  - Compile
Meta-tracer performance (our aim)

- Interpreter
- Tracer
- Machine code

- 1x
- 2x
- 0.1x

- Hot
- Compile
- Safepoint
- Guard failure

- Blackhole interpreter

http://soft-dev.org/
References

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
<table>
<thead>
<tr>
<th><strong>warmup_stats</strong></th>
<th>Use our statistical method on your VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="http://soft-dev.org/src/warmup_stats/">http://soft-dev.org/src/warmup_stats/</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Krun</strong></th>
<th>Run experiments in a controlled manner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><a href="http://soft-dev.org/src/krun/">http://soft-dev.org/src/krun/</a></td>
</tr>
</tbody>
</table>
• EPSRC: COOLER and Lecture.
• Oracle.
• Cloudflare.
Thanks for listening
How long to run things for (0.8)

In-process iterations

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

% similarity to n=2000

Steady iteration (# or s)
Steady performance (s)
Overall
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>× (27l, 2f, 1w)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fannkuch Redux</td>
<td>× (26l, 4w)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fasta</td>
<td>⊥</td>
<td>4.0 (3.0, 5.4)</td>
<td>0.75 (0.535, 5.873)</td>
</tr>
<tr>
<td>nbody</td>
<td>⊥</td>
<td>6.0 (5.0, 7.0)</td>
<td>0.86 (0.704, 1.090)</td>
</tr>
<tr>
<td>Grails</td>
<td>⊥</td>
<td></td>
<td></td>
</tr>
<tr>
<td>richards</td>
<td>⊥</td>
<td>2.0 (2.0, 3.5)</td>
<td>0.95 (0.879, 9.745)</td>
</tr>
<tr>
<td>spectralnorm</td>
<td>⊥</td>
<td>14.0 (2.0, 94.6)</td>
<td>13.60 (0.890, 98.737)</td>
</tr>
</tbody>
</table>

Software Development Team

http://soft-dev.org/