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

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

King’s College London

Software Development Team
2018-05-09
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 with annotations for Compilation and Profiling Interpreter]
What do VM claims pertain to?

![Diagram showing iteration time vs. in-process iteration with compilation, profiling interpreter, and peak performance points.]
What do VM claims pertain to?
Users *always* perceive warmup
Users *always* perceive warmup

Maybe we should know how long it is?
The Warmup Experiment

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
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, 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, ...)

10/50 HTTP://SOFT-DEV.ORG/
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, OpenBSD₄₇₉₀, Proc. exec. #14 (warmup)
Fannkuch Redux, LuaJIT, OpenBSD, Proc. exec. #14 (warmup)
Warmup & flat (1)

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

Changepoint

Software Development Team

12 / 50

http://soft-dev.org/
Warmup & flat (1)

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

Changepoint segment

Changepoint
Warmup & flat (1)

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

In-process iteration: 1.79372, 1.80244, 1.81116, 1.81988, 1.82860, 1.83731, 1.84603

Time (secs)
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)

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

In-process iteration

Time (secs)

Spectral Norm, PyPy, Linux$_{E3−1240v5}$, Proc. exec. #5 (warmup)
Richards, HotSpot, Linux_{E3-1240v5}, Proc. exec. #8 (slowdown)

In-process iteration

Time (secs)

Richards, HotSpot, Linux_{E3-1240v5}, Proc. exec. #8 (slowdown)

In-process iteration

Time (secs)
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*
Fasta, V8, Linux_4790, Proc. exec. #26 (slowdown)
No steady state (1)

Binary Trees, V8, Linux\textsubscript{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: \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{Good}
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*

**Bad**
Warmup or no steady state?

Fannkuch Redux, HotSpot, Linux\textsuperscript{4790}, Proc. exec. #1 (warmup)

In-process iteration

Time (secs)

Fannkuch Redux, HotSpot, Linux\textsuperscript{4790}, Proc. exec. #1 (warmup)
Inconsistent Process-executions

(Same machine)
(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>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TruffleRuby</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Metrics

- **Steady iter (#)**: Number of iterations that the benchmark performs.
- **Steady iter (s)**: Average time taken for each iteration.
- **Steady perf (s)**: Performance of the benchmark in seconds.

### Table Data

<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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TruffleRuby</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Additional Metrics

- **Richards**: Additional performance data.
- **fast**: Performance data for the fast mode.

---

**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>C</td>
<td>(27\overline{c}, 2\overline{c}, 1\overline{c})</td>
<td>775.0 ((1.5,780.0))</td>
<td>425.16 ((0.246,426.809))</td>
</tr>
<tr>
<td>Graal</td>
<td>(2\overline{c})</td>
<td>14.0 ((2.0,94.6))</td>
<td>13.60 ((0.830,98.737))</td>
</tr>
<tr>
<td>HHVM</td>
<td>(29\overline{c}, 1\varnothing)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>(\varnothing)</td>
<td>7.0 ((7.0,7.5))</td>
<td>1.91 ((1.902,3.645))</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(\neg)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>(27\overline{c}, 3\overline{c})</td>
<td>1.0 ((1.0,45.2))</td>
<td>0.00 ((0.000,20.597))</td>
</tr>
<tr>
<td>TruffleRuby</td>
<td>(25\overline{c}, 5\varnothing)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>(\neg)</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\text{\textsubscript{4790}}</th>
<th>Linux\text{\textsubscript{1240v5}}</th>
<th>OpenBSD\text{\textsubscript{4790}} †</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>〈VM, benchmark〉 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></td>
<td></td>
<td></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>⊥</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} †</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(VM, benchmark) 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>(\leq)</td>
<td>20.0%</td>
<td>17.8%</td>
<td>20.0%</td>
</tr>
<tr>
<td>(\geq)</td>
<td>4.4%</td>
<td>4.4%</td>
<td>3.3%</td>
</tr>
<tr>
<td>(\approx)</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>(\not\approx)</td>
<td>51.1%</td>
<td>53.3%</td>
<td>50.0%</td>
</tr>
</tbody>
</table>

### 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>(\leq)</td>
<td>22.0%</td>
<td>23.3%</td>
<td>37.7%</td>
</tr>
<tr>
<td>(\geq)</td>
<td>48.3%</td>
<td>43.9%</td>
<td>35.2%</td>
</tr>
<tr>
<td>(\approx)</td>
<td>20.1%</td>
<td>22.1%</td>
<td>12.1%</td>
</tr>
<tr>
<td>(\not\approx)</td>
<td>9.6%</td>
<td>10.8%</td>
<td>15.0%</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

24 / 50
Are odd effects correlated with compilation and GC?

Fasta, PyPy, Linux\textsubscript{E3} – 1240v5, Proc. exec. #4 (no steady state)

- Time (secs)
- GC
- JIT

In-process iteration

0.756×10\textsuperscript{9} 1.501×10\textsuperscript{9} 2.246×10\textsuperscript{9}

0.01453 0.00725 0.00003

2.62972 2.55091 2.47210

2.39328 2.31447 2.23565

2.15684

\textcopyright 2023 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)

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

In-process iteration

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

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

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

-0.00010
0.02600
0.05210

In-process iteration
Time (secs)
GC (secs)
JIT (secs)

Software Development Team
http://soft-dev.org/
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
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/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.inlining:
+                if warmrunnerstate.can_inline_callable(greenboxes):
+                    portal_code = targetjitdriver_sd.mainjitcode
+                    return self.metainterpreter.perform_call(portal_code, allboxes,
+                        greenkey=greenboxes)
+                    inline = True
+                    if self.metainterpreter.is_main_jitcode(portal_code):
+                        for gk, _ in self.metainterpreter.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.metainterpreter.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
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=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
Process execution #1

In-process iteration

-0.2442
2.0974
4.4389
6.7805
9.1221
11.4636
13.8052

Time (secs)

In-process iteration

0 276 551 826 1102 1378 1653 1928 2204 2480 2755

Software Development Team

35 / 50

http://soft-dev.org/
Octane: analysing pdf.js

Process execution #1

Time (secs)

In-process iteration

0 267 535 802 1069 1336 1604 1871 2138 2406 2673

0 0.0205 0.0679 0.1152 0.1625 0.2098 0.2571 0.3045

0 0.0679 0.1152 0.1625 0.2098 0.2571 0.3045

0 0.0205 0.0679 0.1152 0.1625 0.2098 0.2571 0.3045

0 0.0205 0.0679 0.1152 0.1625 0.2098 0.2571 0.3045

0 0.0205 0.0679 0.1152 0.1625 0.2098 0.2571 0.3045

Software Development Team

http://soft-dev.org/
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.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 □□□□□

```javascript
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 Cannot enlarge memory arrays in asm.js (a memory leak? I don’t know)

Timings are made with a non-monotonic microsecond timer
Why aren’t more users more happy with our VMs?

3UMMARy

HyAREN

TMOREUSERSMOREHAPPyWITH

OUR6-S?

- yTHESIS: BENCHMARKING and BENCHMARKS ARE PERFORMANCE DESTINY. URSHAVEMISLEDUS.

39/50 HTTP://SOFT-DEV.ORG/
Why aren’t more users more happy with our VMs?

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

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

http://soft-dev.org/
How to benchmark a bit better

1. Run benchmarks for longer to uncover issues.
1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
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.
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
EPSRC: COOLER and Lecture.
Oracle.
Cloudflare.
Thanks for listening

Richards, HotSpot, LinuxE3-1240v5, Proc. exec. #8 (slowdown)

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

CAN WE EXISTING 6-S?

AT LEAST A BIT... BUT A LOT? IN CLEAR.

I CAN'T.

I HAVE AN IDEA...
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**

```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
```

---

**Meta-tracing JITs**
```
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

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

## User program (lang FL)

```python
45 / 50 HTTP://SOFT-DEV.ORG/
```
**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
```

**Initial trace**

```
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

Hot

Compile

Safepoint

Guard failure

46 / 50
http://soft-dev.org/
Meta-tracer states

Interpreter

Tracer

Machine code

Blackhole interpreter

Hot

Compile

Guard failure

Safepoint

46 / 50 HTTP://SOFT-DEV.ORG/
Meta-tracer states
Meta-tracer states

Interpreter → Tracer → Machine code

- Interpretation
- Tracing
- Compilation

States:
- Interpreter
- Tracer
- Machine code

Transitions:
- Hot
- Compile
- Safepoint
- Guard failure

- Blackhole interpreter

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

- Interpreter
- Tracer
- Machine code

Flow:
- Hot from Interpreter to Tracer
- Compile from Tracer to Machine code
- Safepoint from Machine code to Interpreter
- Guard failure from Machine code to Blackhole interpreter

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

Interpreter → Tracer → Machine code → Blackhole interpreter → Interpreter

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

Interpreter → Tracer → Machine code → Blackhole interpreter → Interpreter

- Hot
- Compile
- Guard failure
- Safepoint
Meta-tracer performance (now)

Interpreter → Tracer → Machine code

1x

Blackhole interpreter

Hot

Compile

Safepoint

Guard failure

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

Interpreter → Tracer → Machine code

Hot

Compile

1x

200x

0.1x

Blackhole interpreter

Guard failure

Safepoint
Meta-tracer performance (our aim)

Interpreter Tracer Machine code

1x 2x 0.1x

Hot Compile Guard failure Safepoint

Blackhole interpreter

48 / 50 HTTP://SOFT-DEV.ORG/
How long to run things for (0.8)
<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 (27l, 2r, 1ω)</td>
<td>0.75 (0.535, 5.873)</td>
<td>0.16188 ± 0.000738</td>
</tr>
<tr>
<td>fannkuch Redux</td>
<td>0 (26l, 4ω)</td>
<td>0.86 (0.704, 1.090)</td>
<td>0.13677 ± 0.000343</td>
</tr>
<tr>
<td>fasta</td>
<td>6.0 (5.0, 7.0)</td>
<td>0.95 (0.879, 9.745)</td>
<td>0.26465 ± 0.00761</td>
</tr>
<tr>
<td>nbody</td>
<td>2.0 (2.0, 35.3)</td>
<td>13.60 (2.0, 94.6)</td>
<td>1.05685 ± 0.00126</td>
</tr>
<tr>
<td>richards</td>
<td>14.0 (2.0, 94.6)</td>
<td>13.60 (0.890, 98.737)</td>
<td>1.05685 ± 0.00126</td>
</tr>
</tbody>
</table>