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
2018-01-22
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 versus in-process iteration, with labels for Compilation and Profiling Interpreter.](http://soft-dev.org/)
What do VM claims pertain to?

![Graph showing iteration time vs in-process iteration with key points labeled: Compilation, Profiling Interpreter, Peak Performance.](http://soft-dev.org/)
What do VM claims pertain to?

![Graph showing iteration time and in-process iteration with a warmup phase.](http://soft-dev.org/)
Warmup

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.
Method 1: Which benchmarks?

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
- 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)
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, ...)

10/48 HTTP://SOFT-DEV.ORG/
Warmup & flat (1)

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

Time (secs)

In-process iteration
In-process iteration
0.56523
0.56703
0.56882
0.57061
0.57241
0.57420
0.57600

Time (secs)
Fannkuch Redux, LuaJIT, OpenBSD
Proc. exec. #12 (warmup)
Warmup & flat (1)

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

In-process iteration

Time (secs)

In-process iteration

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

1 210 401 601 801 1001 1201 1401 1601 1801 2000

1 26 50

0.56806
0.57209
0.57613

11/48 HTTP://SOFT-DEV.ORG/
Warmup & flat (1)

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

Changepoint
Warmup & flat (1)

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

Changepoint

Changepoint segment
Warmup & flat (1)

N-Body, PyPy, Linux $E_3^{E3-1240v5}$, Proc. exec. #6 (flat)
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)
Warmup & flat (2)

Spectral Norm, PyPy, Linux $E_3^{\frac{1240}{v5}}$, Proc. exec. #13 (warmup)
Richards, Hotspot, Linux_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

Time (secs)
Fasta, V8, Linux, Proc. exec. #14 (slowdown)

1 26 50
1.14086
1.15142
1.16197
No steady state (1)

Binary Trees, V8, Linux, 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: *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, Proc. exec. #1 (warmup)
Inconsistent Process-executions

Binary Trees, V8, Linux\textsubscript{E3–1240v5}, Proc. exec. #7 (warmup)

Binary Trees, V8, Linux\textsubscript{E3–1240v5}, Proc. exec. #8 (slowdown)

(Same machine)
Inconsistent Process-executions

(Binary Trees, C, Linux, Proc. exec. #12 (no steady state))

(Binary Trees, C, OpenBSD, Proc. exec. #30 (flat))

(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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Binary trees**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Fannkuch redux**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Richards**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**spectrnorm**

<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>× (27l, 3f)</td>
<td>32.0 (17.0, 193.8)</td>
<td>6.60 (3.729, 36.608)</td>
<td>0.18594 ±0.000316</td>
</tr>
<tr>
<td>HHVM</td>
<td>× (24l, 4f, 2w)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>× (25l, 5f)</td>
<td>7.0 (7.0, 53.5)</td>
<td>1.19 (1.182, 9.703)</td>
<td>0.18279 ±0.000116</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td></td>
<td>1082.0 (999.0, 1232.5)</td>
<td>2219.59 (2039.304, 2516.021)</td>
<td>2.05150 ±0.017737</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>× (23l, 4f, 2–, 1w)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>× (27f, 3w)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>× (15–, 9l, 6f)</td>
<td>1.5 (1.0, 794.0)</td>
<td>0.25 (0.000, 391.026)</td>
<td>0.49237 ±0.003198</td>
</tr>
</tbody>
</table>
## Overall benchmark stats

<table>
<thead>
<tr>
<th>Class</th>
<th>Linux(_{4790})</th>
<th>Linux(_{1240v5})</th>
<th>OpenBSD(_{4790}^{†})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\langle VM, benchmark \rangle 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>$\backslash$</td>
<td>28.3%</td>
<td>23.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>$\Join$</td>
<td>6.5%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>$\ast$</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>
<tr>
<td></td>
<td>\textbf{Process executions}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$-$</td>
<td>26.4%</td>
<td>20.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>$\backslash$</td>
<td>48.3%</td>
<td>51.5%</td>
<td>52.1%</td>
</tr>
<tr>
<td>$\Join$</td>
<td>16.7%</td>
<td>17.9%</td>
<td>11.1%</td>
</tr>
<tr>
<td>$\ast$</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>

## 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></td>
<td></td>
<td></td>
</tr>
<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:
Summary

Classical warmup occurs for only:

72.4%–74.7% of process executions
Classical warmup occurs for only:

72.4%–74.7% of process executions

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

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

- GC
- JIT

In-process iteration

Time (secs)

Software Development Team

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

Richards, Hotspot, Linux\textsubscript{E3 - 1240v5}, Proc. exec. #3 (slowdown)
Are odd effects correlated with compilation and GC?

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

\begin{center}
\begin{tabular}{c}
\hline
In-process iteration & Time (secs) & JIT (secs) & GC (secs) \\
\hline
1 & 0.39292 & & 0.00000 \\
201 & 0.39554 & & 0.00000 \\
401 & 0.39817 & & 0.00000 \\
601 & 0.40079 & & 0.00000 \\
801 & 0.40342 & & 0.00000 \\
1001 & 0.40604 & & 0.00000 \\
1201 & 0.40867 & & 0.00000 \\
1401 & & & \\
1601 & & & \\
1801 & & & \\
2000 & & & \\
\hline
\end{tabular}
\end{center}

\texttt{24/48 HTTP://SOFT-DEV.ORG/}
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
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):
             # We’ve found a potentially inlinable function; now we need to
-            if warmrunnerstate.can_inline_callable(greenboxes):
+            # 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.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
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></th>
<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></td>
<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></td>
<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></td>
<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></td>
<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></td>
<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></td>
<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
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
In-process iteration

0.0126
5.9004
11.7882
17.6760
23.5638
29.4516
35.3394

Process execution #1

Time (secs)

35.3394
29.4516
23.5638
17.6760
11.7882
5.9004
0.0126

In-process iteration

0 11 23 34 46 57 68 80 91 103 114

34 / 48

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

iltratt wants to merge 2 commits into chromium:master fromiltratt:master

Changes from all commits 1 file +1 −0

1 pdfjs.js

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

+ function runPdfJS() {
    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)
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?
Why aren’t more users more happy with our VMs?

My thesis: benchmarking \textit{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. Unbenchmark 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/
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 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.
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
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...
FL Interpreter

```python
def 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 isa_instance(lhs, int)
            and isa_instance(rhs, int):
                stack.push(lhs + rhs)
        else: ...
        program_counter += 1
```

**Meta-tracing JITs**
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**

### 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
```

```python
if x < 0:
    x = x + 1
else:
    x = x + 2
x = x + 3
```
**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_VAR_SET)
v7 = dict_get(v2, "x")
list_append(v1, v7)
v8 = add(v5, 1)
v9 = load_instruction(v8)
guard_eq(v9, INSTR_VAR_GET)
v10 = dict_get(v2, "x")
list_append(v1, v10)
v11 = add(v7, 1)
v12 = load_instruction(v11)
guard_eq(v12, INSTR_VAR_SET)
v13 = dict_get(v2, "x")
guard_eq(v13, INSTR_VAR_GET)
```

...
Meta-tracer states

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

Interpreter \rightarrow \text{Hot} \rightarrow \text{Tracer} \rightarrow \text{Compile} \rightarrow \text{Machine code} \rightarrow \text{Guard failure} \rightarrow \text{Safepoint} \rightarrow \text{Blackhole interpreter} \rightarrow \text{Interpreter}
Meta-tracer states

Interpreter → Tracer → Machine code

Hot

Compile

Guard failure

Safepoint

Blackhole interpreter
Meta-tracer states

Interpreter → Tracer → Machine code

Hot

Compile

Safepoint

Blackhole interpreter

Guard failure
Meta-tracer states

Interpreter Tracer Machine code

Hot Compile

Guard failure

Safepoint

Blackhole interpreter
Meta-tracker states

Interpreter

Tracer

Machine code

Hot

Compile

Guard failure

Safepoint

Blackhole interpreter

42 / 48

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

Interpreter → Tracer → Machine code

Hot → Compile

Safepoint → Guard failure

Blackhole interpreter
Meta-tracer performance (now)

Interpreter → Tracer → Machine code → Blackhole interpreter → Interpreter

- Hot
- Compile
- Guard failure
- Safepoint

1x
Meta-tracer performance (now)

Interpreter 1x
Tracer
Blackhole interpreter

Hot
Compile

Safepoint
Guard failure

Machine code 0.1x
Meta-tracer performance (now)

Interpreter → Tracer → Machine code

- 1x
- 200x
- 0.1x
Meta-tracer performance (our aim)

Interpreter

Tracer

Blackhole interpreter

Machine code

Hot

Compile

Guard failure

Safepoint

1x

2x

0.1x

Interpreter Tracer Machine
code
Blackhole
interpreter
Hot Compile
Guard failure Safepoint
1x 0.1x2x

44 / 48 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
Thanks

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

Process executions

% similarity to n=30

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

http://soft-dev.org/