VM Warmup Blows Hot and Cold
OOPSLA 2017, Vancouver

Edd Barrett
Carl Friedrich Bolz-Tereick
Rebecca Killick (Lancaster)
Sarah Mount
Laurence Tratt

KING'S College LONDON
Software Development Team
2017-10-25

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

- Invoke VM
How to benchmark a JIT:

- Invoke VM
  - “process execution”
How to benchmark a JIT:

- Invoke VM
  - “process execution”

- Run Benchmark in a Loop
The Current State of the Art of Benchmarking

How to benchmark a JIT:

- Invoke VM
  - “process execution”

- Run Benchmark in a Loop
  - “in-process iterations”
The Current State of the Art of Benchmarking

![Graph: Iteration Time vs. In-Process Iteration]
The Current State of the Art of Benchmarking

JUST IN TIME (JIT) COMPILATION

Iteration Time

In-process Iteration
The Current State of the Art of Benchmarking

JUST IN TIME (JIT) COMPILATION

ITERATION TIME

IN-PROCESS ITERATION

Profiling Interpreter
The Current State of the Art of Benchmarking

JUST IN TIME (JIT) COMPILATION

Compilation

Profiling Interpreter

Iteration Time

In-Process Iteration
The Current State of the Art of Benchmarking

![Graph showing iteration time vs. in-process iteration, with phases labeled as Just In Time (JIT) Compilation, Compilation, Profiling Interpreter, and Peak Performance.]
The Current State of the Art of Benchmarking

MORE REALISTIC VM WARMUP

ITERATION TIME

IN-PROCESS ITERATION
The Current State of the Art of Benchmarking

MORE REALISTIC VM WARMUP

ITERATION TIME

IN-PROCESS ITERATION

SOME NOISE
The Current State of the Art of Benchmarking

MORE REALISTIC VM WARMUP

ITERATION TIME vs. IN-PROCESS ITERATION

COMPILATION TIERS

SOME NOISE
The Current State of the Art of Benchmarking

MORE REALISTIC VM WARMUP

ITERATION TIME

IN-PROCESS ITERATION

SOME NOISE

COMPILATION TIERS

GC SPIKES
The Current State of the Art of Benchmarking

**The Warmup Phase**

**Iteration Time**

**In-Process Iteration**

Software Development Team

http://soft-dev.org/
1. Users dislike poor warmup.
1. Users dislike poor warmup.

2. VM authors dislike poor warmup.
Warmup Matters

1. Users dislike poor warmup.

2. VM authors dislike poor warmup.

Warmup is important!
We should measure the warmup of modern language implementations
We should measure the warmup of modern language implementations

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

http://benchmarksgame.alioth.debian.org/
Method 2: How Long to Run?
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 off.
- Hyper-threading off.
Control as many confounding variables as possible
Control as many confounding variables as possible

- 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, ...)

KRUN
Method 6: Changepoint Analysis and Classification
Method 6: Changepoint Analysis and Classification

Outliers identified and ignored
Method 6: Changepoint Analysis and Classification
Method 6: Changepoint Analysis and Classification
Method 6: Changepoint Analysis and Classification

- All segs are equivalent:
  \[
  \text{FLAT} \ (-) 
  \]
Method 6: Changepoint Analysis and Classification

- All segs are equivalent: \texttt{FLAT (−)}
- Length(final equivalent segs) \(\geq 500\) iters:
  - and are the fastest segs: \texttt{WARMUP (⎕)}
Method 6: Changepoint Analysis and Classification

- All segs are equivalent: \textcolor{green}{\textsc{Flat} (−)}

- Length(final equivalent segs) >= 500 iters:
  - and are the fastest segs: \textcolor{red}{\textsc{Warmup} (┑)}
  - but not the fastest segs: \textcolor{purple}{\textsc{Slowdown} (┙)}
Method 6: Changepoint Analysis and Classification

- All segs are equivalent:
  \textsc{Flat} (−)

- Length(final equivalent segs) \geq 500 iters:
  - and are the fastest segs:
    \textsc{Warmup} (descending)
  - but not the fastest segs:
    \textsc{Slowdown} (descending)

- Otherwise:
  \textsc{No Steady State} (\textvisiblespace)
Results
Results: Flat

N-Body, PyPy, Linux_{E3-1240v5}, Proc. exec. #6 (flat)
Results: Warmup

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

In-process iteration

Time (secs)

1 201 401 601 801 1001 1201 1401 1601 1801 2000

1.18854
1.18276
1.17699
1.17122
1.16545
1.15968
1.15391
Results: Warmup

Spectral Norm, PyPy, Linux\textsuperscript{E3} = \textsuperscript{-1240v5}, Proc. exec. #13 (warmup)

Time (secs)
Results: Slowdown

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

In-process iteration

Time (secs)
Results: Slowdown

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

In-process iteration

Time (secs)

Fasta, V8, Linux, Proc. exec. #14 (slowdown)
Results: No Steady State

Binary Trees, V8, Linux, Proc. exec. #24 (no steady state)
Results: Inconsistent Process-executions

(Same machine)
## Quantitative Results

<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>32.0 (17, 691.8)</td>
<td>6.60 (3,729, 36.669)</td>
<td>0.18594 ± 0.008315</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>(271, 3r)</td>
<td></td>
<td></td>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td>(251, 5r)</td>
<td>7.0 (7.0, 25.5)</td>
<td>1.19 (1.823, 7.805)</td>
<td>0.18279 ± 0.008116</td>
<td>(161, 11r, 3m)</td>
<td>2.0 (2.0, 2.0)</td>
<td>17.95 (17.716, 18.127)</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>(999, 0, 1232.5)</td>
<td>1082.0 (2039, 204.218)</td>
<td>2.05150 ± 0.037238</td>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>(277, 3r)</td>
<td></td>
<td></td>
<td>V8</td>
<td>(15, 91, 6r)</td>
<td>1.5 (1, 674)</td>
<td>0.25 (0, 000, 010)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>(21, 61, 2r, 1w)</td>
<td></td>
<td></td>
<td>HHVM</td>
<td>(281, 1r, 1f)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td></td>
<td></td>
<td></td>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td>10.0 (10, 0.0, 018)</td>
<td>52.66 (52.660, 52.780)</td>
<td>1.35779 ± 0.031948</td>
<td>(26, 2r, 2r)</td>
<td>1021.0 (1024.5, 1027.0)</td>
<td>917.30 (901.7, 946.809)</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>(999, 1232, 3)</td>
<td>1016.5 (1039, 0, 04)</td>
<td>1.08333 ± 0.038300</td>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td>(151, 13, 2r)</td>
<td>2.0 (1, 0, 28)</td>
<td>1.57 (0, 000, 03)</td>
<td>1.55442 ± 0.029549</td>
<td>(291, 7, 2r)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>(191, 11)</td>
<td>2.0 (1, 0, 25)</td>
<td>0.31 (0, 000, 7, 525)</td>
<td>0.30401 ± 0.009545</td>
<td>(29, 1)</td>
<td>4.0 (1, 0, 16)</td>
<td>1.44 (1, 451, 7, 155)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>HHVM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graal</td>
<td>(291, 1)</td>
<td></td>
<td></td>
<td>HotSpot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HotSpot</td>
<td></td>
<td>261.0 (6, 395.0)</td>
<td>30.73 (946.78.001)</td>
<td>0.11744 ± 0.001725</td>
<td>(201, 1)</td>
<td>997.0 (201, 0.101)</td>
<td>546.38 (546.547, 717)</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>(181, 12)</td>
<td>15.0 (2, 6.31)</td>
<td>12.40 (9.61, 17.800)</td>
<td>0.89293 ± 0.006977</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LuaJIT</td>
<td></td>
<td></td>
<td></td>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PyPy</td>
<td></td>
<td></td>
<td></td>
<td>V8</td>
<td>(191, 10, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Class.</th>
<th>Steady iter (#)</th>
<th>Steady iter (s)</th>
<th>Steady perf (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>( \sim )</td>
<td>32.0</td>
<td>6.60</td>
</tr>
<tr>
<td>Graal</td>
<td>( \sim ) (27\ houses, 3\ floors)</td>
<td>(17.0, 193.8)</td>
<td>(3.729, 36.608)</td>
</tr>
<tr>
<td>HHVM</td>
<td>( \sim ) (24\ houses, 4\ floors, 2\ windows)</td>
<td>7.0 (7.0, 53.5)</td>
<td>1.19 (1.182, 9.703)</td>
</tr>
<tr>
<td>HotSpot</td>
<td>( \sim ) (25\ houses, 5\ floors)</td>
<td>1082.0 (999.0, 1232.5)</td>
<td>2219.59 (2039.304, 2516.021)</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>( \sim )</td>
<td>1.5 (1.0, 794.0)</td>
<td>0.25 (0.000, 391.026)</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>( \sim ) (23\ houses, 4\ floors, 2\ windows, 1\ window)</td>
<td>(27\ houses, 3\ windows)</td>
<td>(30\ houses, 4\ floors)</td>
</tr>
</tbody>
</table>

Software Development Team

http://soft-dev.org/
<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>(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>
<tr>
<td></td>
<td>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>_</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>
### Results: Summary

<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>(_)</td>
<td>28.3%</td>
<td>23.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>(\sum)</td>
<td>6.5%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\approx)</td>
<td>4.3%</td>
<td>6.5%</td>
<td>0.0%</td>
</tr>
<tr>
<td>(\approx)</td>
<td>6.5%</td>
<td>6.5%</td>
<td>13.3%</td>
</tr>
<tr>
<td>(\approx)</td>
<td>45.7%</td>
<td>43.5%</td>
<td>70.0%</td>
</tr>
<tr>
<td></td>
<td>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>(\approx)</td>
<td>48.3%</td>
<td>51.5%</td>
<td>52.1%</td>
</tr>
<tr>
<td>(\sum)</td>
<td>16.7%</td>
<td>17.9%</td>
<td>11.1%</td>
</tr>
<tr>
<td>(\approx)</td>
<td>8.7%</td>
<td>9.6%</td>
<td>2.8%</td>
</tr>
</tbody>
</table>

Best case no. good process executions: 86.1%
### Results: Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>$\text{Linux}_{4790}$</th>
<th>$\text{Linux}_{1240v5}$</th>
<th>$\text{OpenBSD}_{4790}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>8.7%</td>
<td>13.0%</td>
<td>6.7%</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>28.3%</td>
<td>23.9%</td>
<td>10.0%</td>
</tr>
<tr>
<td>$\downarrow$</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>$\propto$</td>
<td>45.7%</td>
<td>43.5%</td>
<td>70.0%</td>
</tr>
</tbody>
</table>

#### (VM, benchmark) pairs

#### Process executions

<table>
<thead>
<tr>
<th>Class</th>
<th>$\text{Linux}_{4790}$</th>
<th>$\text{Linux}_{1240v5}$</th>
<th>$\text{OpenBSD}_{4790}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-$</td>
<td>26.4%</td>
<td>20.9%</td>
<td>34.0%</td>
</tr>
<tr>
<td>$\uparrow$</td>
<td>48.3%</td>
<td>51.5%</td>
<td>52.1%</td>
</tr>
<tr>
<td>$\downarrow$</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>

**Best case good (VM, benchmark) pairings: 43.5%**
Hypothesis Invalid!

Hypothesis: Small, deterministic programs reach a steady state of peak performance.
Are the Effects due to JIT and GC?

Fasta, PyPy, Linux_E3-1240v5, Proc. exec. #5 (no steady state)

![Graph showing time (secs) and in-process iteration]

- GC
- JIT

In-process iteration

Time (secs)

Software Development Team

http://soft-dev.org/
Are the Effects due to JIT and GC?

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

In-process iteration

Time (secs)

GC (secs)

JIT (secs)

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

-0.00008
0.01950
0.03908

JIT (secs)

http://soft-dev.org/
However

In many cases, the JIT/GC can’t explain oddness
What Have We Learned?

- Benchmarks often don’t warmup as we expect.
- Repeating a benchmark often gives a different warmup characteristic.
- Have we been misled?
  - Ineffectual or bad optimisations?
What Can We Do?

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. We can’t always blame the compiler.
4. Always report warm-up time.
5. Engineer for predictable performance?

26/32 HTTP://SOFT-DEV.ORG/
1. Run benchmarks for longer to uncover issues.
1. Run benchmarks for longer to uncover issues.

2. Accept that peak performance may not occur.
What Can We Do?

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. We can’t always blame GC or JIT compilation.
What Can We Do?

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. We can’t always blame GC or JIT compilation.
4. Always report warmup time.
What Can We Do?

1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. We can’t always blame GC or JIT compilation.
4. Always report warmup time.
5. Engineer VMs for predictable performance?

(These slides correspond to version 6 of the paper)
Richards, Hotspot, Linux\textsubscript{E3-1240v5}, Proc. exec. #3 (slowdown)

In-process iteration

Time (secs)

JIT (secs)

GC (secs)

Richards, Hotspot, Linux\textsubscript{E3-1240v5}, Proc. exec. #3 (slowdown)
Backup Slides
Method 7: Summary Statistics

For each machine, we summarise each \( \langle \text{VM, Benchmark} \rangle \) pairing.
For each machine, we summarise each ⟨VM, Benchmark⟩ pairing.

**Consistent**

All 30 process executions were classified the same.

```plaintext
- [ ] [ ] [ ]
```
Method 7: Summary Statistics

For each machine, we summarise each \( \langle \text{VM, Benchmark} \rangle \) pairing.

### Consistent

All 30 process executions were classified the same.

![Consistent Classification](image)

### Inconsistent

A mix of classifications arose within the 30 process executions. E.g.:

- **Good inconsistent:** \( = (25\leftarrow, 5\rightarrow) \)
- **Bad inconsistent:** \( \not= (20\leftarrow, 3\rightarrow, 7\rightarrow\leftarrow) \)
Method 7: Summary Statistics

For each machine, we summarise each ⟨VM, Benchmark⟩ pairing.

<table>
<thead>
<tr>
<th>Consistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>All 30 process executions were classified the same.</td>
</tr>
<tr>
<td>⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗ ⸗</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inconsistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A mix of classifications arose within the 30 process executions. E.g.:</td>
</tr>
<tr>
<td>- Good inconsistent:  =$\sim$(25�wechat, 5�wechat)</td>
</tr>
<tr>
<td>- Bad inconsistent:  $\neq (20�wechat, 3\bar{\text{r}}�wechat, 7\bar{\text{r}}�wechat)$</td>
</tr>
</tbody>
</table>

If possible report: steady state performance, time until steady state, etc.