VM Warmup Blows Hot and Cold

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

Software Development Team
January 25, 2017

KING'S College LONDON

http://soft-dev.org/
What’s VM Warmup?
What's VM Warmup?
What’s VM Warmup?

IDEALISED JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

http://soft-dev.org/
What’s VM Warmup?

IDEALISED JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

PROFILING INTERPRETER
What’s VM Warmup?

IDEALISED JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

Compilation

Profiling Interpreter
What’s VM Warmup?

IDEALISED JIT WARMUP

ITERATION TIME

COMPIRATION

PROFILING INTERPRETER

IN-PROCESS ITERATION

PEAK PERFORMANCE
What’s VM Warmup?

IDEALISED JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

← WARMUP →
What’s VM Warmup?

MORE REALISTIC VM WARMUP

ITERATION TIME

IN-PROCESS ITERATION
What’s VM Warmup?

More realistic VM warmup

Iteration time

In-process iteration

Some noise
What’s VM Warmup?

More realistic VM warmup

Iteration time

Compilation tiers

Some noise

In-process iteration
What’s VM Warmup?

More realistic VM warmup

- Iteration time
- In-process iteration

- Compilation tiers
- GC spikes
- Some noise
Why care?
What’s VM Warmup?

Users hate noticeable warmup.

Iteration time vs. in-process iteration.

Frustrating → Happy Days!
What's VM Warmup?

VM AUTHORS HATE ALL WARMUP

ITERATION TIME

IN-PROCESS ITERATION
Warmup is bad for everyone.
Our Experiment
Goal:

Measure warmup of modern (JITted) language VMs
Goal:

Measure warmup of modern (JITted) language VMs

Hypothesis:

Small, deterministic programs exhibit classical warmup behaviour
## Which VMs?

<table>
<thead>
<tr>
<th>VM</th>
<th>Version</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graal</td>
<td>0.18</td>
<td>Java</td>
</tr>
<tr>
<td>HHVM</td>
<td>3.15.3</td>
<td>PHP</td>
</tr>
<tr>
<td>JRuby/Truffle</td>
<td>graalvm-0.18</td>
<td>Ruby</td>
</tr>
<tr>
<td>Hotspot</td>
<td>8u112b15</td>
<td>Java</td>
</tr>
<tr>
<td>LuaJit</td>
<td>2.0.4</td>
<td>Lua</td>
</tr>
<tr>
<td>PyPy</td>
<td>5.6.0</td>
<td>Python</td>
</tr>
<tr>
<td>V8</td>
<td>5.4.500.43</td>
<td>Javascript</td>
</tr>
<tr>
<td>GCC</td>
<td>4.9.3</td>
<td>C/C++</td>
</tr>
</tbody>
</table>

Note: same GCC (4.9.3) used for all compilation
The language benchmark games

https://benchmarksgame.alioth.debian.org/

(small benchmarks, multi-language, common optimisation target)
Which benchmarks?

The language benchmark games

https://benchmarksgame.alioth.debian.org/

(small benchmarks, multi-language, common optimisation target)

- We removed any CFG non-determinism
- We added checksums to all benchmarks
How long to run for?

2000 *in-process iterations* per process execution.
2000 *in-process iterations* per process execution.

10 *process executions*
Which Machines?

- Linux-Debian8/i4790K, 24GiB RAM
- Linux-Debian8/i4790, 32GiB RAM
- OpenBSD-5.8/i4790, 32GiB RAM
Which Machines?

- Linux-Debian8/i4790K, 24GiB RAM
- Linux-Debian8/i4790, 32GiB RAM
- OpenBSD-5.8/i4790, 32GiB RAM

- Turbo boost and hyper-threading disabled
- Daemons disabled (cron, smtpd)
- Tickless kernel (Linux only)
- Disable Intel P-state driver (Linux only)
- Linux machine software identical.
How to run the Benchmarks?

Benchmark runner: KRUN

https://github.com/softdevteam/krun

Tries to control confounding variables
How to run the Benchmarks?

- Network device taken down during benchmarking
- Drops privileges to a fresh user account for each proc. exec
- Automatically reboots the system prior to each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Minimises I/O
- Sets fixed heap and stack ulimits
- Checks `dmesg` for changes after each proc. exec
- Enforces kernel settings (tickless mode, CPU governors, ...)
- ...
Preliminary Results
Classical Warmup

Fasta, Graal, Linux, Proc. exec. #1

Time (secs)
In-process iteration

Software Development Team
http://soft-dev.org/
Classical Warmup

Binary Trees, HHVM, Linux\textsubscript{4790K}, Proc. exec. #7

Binary Trees, HHVM, Linux\textsubscript{4790}, Proc. exec. #1

(different machines)
In-process iteration

Richards, Hotspot, Linux4790K, Proc. exec. #3
Slowdown

Fasta, PyPy, Linux⁴₇₉₀, Proc. exec. #3

Time (secs)

In-process iteration

2.5003
2.5962
2.6921
2.7879
2.8838
2.9797
3.0755

20/40 HTTP://SOFT-DEV.ORG/
No Steady State

Fasta, PyPy, OpenBSD, Proc. exec. #10

In-process iteration

Time (secs)
No Steady State

Fannkuch Redux, Hotspot, Linux4790K, Proc. exec. #1

In-process iteration

Time (secs)
Inconsistent Process-executions

(same machine)
Inconsistent Process-executions

Binary Trees, C, OpenBSD, Proc. exec. #4

Binary Trees, C, Linux, Proc. exec. #5

(different machines)
Summarising with Changepoint Analysis
Summarising with Changepoint Analysis

![Diagram showing in-process iteration and time (secs)]
Summarising with Changepoint Analysis

In-process iteration

0.30174
0.30963
0.31753
0.32543
0.33333
0.34123
0.34913

Time (secs)

0.696
1.392

CC #0
Fannkuch Redux, Hotspot, Linux

0.000
0.696
1.392

CC #1

0.000
0.696
1.392

CC #2

0.000
0.696
1.392

CC #3

0.000
0.696
1.392
For each \((\text{Machine, VM, Benchmark})\) combo, the process executions can be either:

- All flat
- All warmup
- All slowdown
- All no steady state
- Inconsistent (\(>1\) of the above)
## Summarising with Changepoint Analysis

<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>(\infty)</td>
<td>12 \pm 5</td>
<td>2.78 \pm 1.047</td>
</tr>
<tr>
<td>Graal</td>
<td>(\subseteq)</td>
<td>73 \pm 11</td>
<td>151.71 \pm 24.233</td>
</tr>
<tr>
<td>HHVM</td>
<td>(\subseteq)</td>
<td>6 \pm 7</td>
<td>1.12 \pm 1.296</td>
</tr>
<tr>
<td>HotSpot</td>
<td>(\subseteq)</td>
<td>7 \pm 7</td>
<td>0.17329 \pm 0.17220</td>
</tr>
<tr>
<td>JRuby+Truffle</td>
<td>binary trees</td>
<td>(7,\ll, 3,J)</td>
<td>0.37885 \pm 0.000073</td>
</tr>
<tr>
<td>LuaJIT</td>
<td>(5,\ll, 4,J, 1,l)</td>
<td>(\infty)</td>
<td>0.50777 \pm 0.000048</td>
</tr>
<tr>
<td>PyPy</td>
<td>(6,\ll, 4,J)</td>
<td>(\infty)</td>
<td>0.27360 \pm 0.000030</td>
</tr>
<tr>
<td>V8</td>
<td>(9,\ll, 1,J)</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
</tbody>
</table>
How many benchmarks were consistently – (flat) or \(\uparrow\) (warmup)?

<table>
<thead>
<tr>
<th>Machine</th>
<th>% (\langle\text{Bench, VM}\rangle) pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux(_{4790K})</td>
<td>50.0</td>
</tr>
<tr>
<td>Linux(_{4790})</td>
<td>56.5</td>
</tr>
<tr>
<td>OpenBSD(_{4790})</td>
<td>43.3</td>
</tr>
</tbody>
</table>
Hypothesis: Small, deterministic programs exhibit classical warmup behaviour
How can we measure anything any more?

Standard methods assume:

- Convergence upon steady state
- Normally distributed samples
Are our experiments repeatable?

Many inconsistent process executions
Understanding Why
For PyPy and HotSpot we measured (per in-process iteration):

- Time spent in the GC
- Time spent in the JIT

Can this help to explain why?
GC Instrumentation

Fasta PyPy Linux\textsubscript{4790K} Proc exec #1

Time (secs)

In-process iteration

$\text{In-process iteration}$
JIT Instrumentation

Richards, Hotspot, Linux4790K, Proc. exec. #1

In-process iteration

Time (secs)

JIT (secs)
But...

In many cases we can’t explain

(Future work to find out?)
Conclusions
Conclusions

- Only \( \approx 50\% \) of \( \langle \text{Machine, VM, Benchmark} \rangle \) combos consistently warm up OK.
  - \((-\), \(\subseteq\)).

- We can’t assume that benchmarks will warmup or stabilise
  - \(\left(\right\rangle, \simeq\)).

- We can’t assume that benchmarking is repeatable
  - Even on the same machine, same installation, ...
  - \(\left(\not\approx\right)\).
Downloads

Code

https://github.com/softdevteam/krun/

https://github.com/softdevteam/warmup_experiment/

Data

https://archive.org/download/softdev_warmup_experiment_artefacts/v0.7/
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 for listening

**Users hate noticeable warmup**

**VM authors hate all warmup**

Richards, Hotspot, Linux, Proc. exec. #3

0 200 400 600 800 1000 1200 1400 1600 1800 2000

In-process iteration

0.22302
0.22673
0.23045
0.23417
0.23788
0.24160
0.24531

Time (secs)