What Exactly do we Mean by *JIT Warmup*?

Edd Barrett, Carl Friedrich Bolz, Rebecca Killick (Lancaster), Vincent Knight (Cardiff), Sarah Mount, Laurence Tratt

Software Development Team
April 4, 2016
1. JIT Warmup Background
2. The Back-Story
3. The Warmup Experiment v2.0
4. Results
5. Automated Analyses
6. Conclusion and Future Work
JIT Warmup Background
Informally:

“The time taken to reach peak performance”
JIT Warmup Background

Classical JIT Warmup

Iteration Time vs. In-Process Iteration
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

PROFILING INTERPRETER

5 / 40

http://soft-dev.org/
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

COMPIRATION

PROFILING
INTERPRETER
JIT Warmup Background

CLASSICAL JIT WARMUP

- Compilation
- Profiling Interpreter
- Peak Performance

Iteration Time vs. In-Process Iteration
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

← WARMUP →

IN-PROCESS ITERATION
JIT Warmup Background

MORE REALISTIC JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION
MORE REALISTIC JIT WARMUP

IN-PROCESS ITERATION

ITERATION TIME

SOME NOISE
MORE REALISTIC JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

SOME NOISE

COMPIILATION TIERS
MORE REALISTIC JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

SOME NOISE

GC SPIKES

COMPILATION TIERS
Why is warmup important?

- Warmup contributes to overall performance.
- Long warmup is bad for user-facing and short-lived programs.
- VM authors report peak performance.
We have a hunch that warmup is longer than people expect.

We have some preliminary ideas to improve warmup.
Goal:

Measure how long modern JITs take to warm up.
Microbenchmarks.

Reasonable number of repetitions.
- 10 process executions.
- 50 in-process iterations.

Run on various VMs.

Plot and report warmup time.
Weird Results

Many benchmarks don’t warmup under the classic model.
New goal:

Try to understand why we see “weird” results.
Microbenchmarks Revisited

- CFG determinism.
  - Each run takes same path through CFG.

- Checksums.
  - Harder for VMs to optimise away whole benchmark.
  - Ensures different languages do the same work.

Code for microbenchmarks:

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

Benchmark runner that aims to control sources of variation.

WRT: memory limits, I/O, system state, …

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

- GCC-4.9.3
- Graal-0.13
- HHVM-3.12.0
- JRuby/Truffle (recent git version)
- Hotspot-8u72b15
- LuaJit-2.0.4
- PyPy-4.0.1
- V8-4.9.385.21

Same GCC across the board, minor VM patching.
Machines

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

- “Turbo boost” disabled.
- SSH blocked from non-local machines.
- Daemons disabled (e.g. cron, smtpd).
Run for Longer

- Run many more in-process iterations (2000).

- Plot results and see if we see classic warmup now.
Results
Classical Warmup

In-process iteration

Time(s)

Fasta, V8, Linux2/i7-4790, Process execution #1

Software Development Team

http://soft-dev.org/
Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #7

In-process iteration

Time(s) vs. In-process iteration

0 200 400 600 800 1000 1200 1400 1600 1800 2000

0.466
0.469
0.471
0.473
0.476
0.478
0.480
(Different machines)
Changing Phases

Fasta, C, Linux2/i7-4790, Process execution #8

In-process iteration

Time(s)
Vastly Inconsistent Process-executions

Fasta, PyPy, Linux2/i7-4790, Process execution #3

Fasta, PyPy, Linux2/i7-4790, Process execution #4

(same machine)
Vastly Inconsistent Process-executions

(Different machines. Bouncing ball pattern Linux-specific)
https://archive.org/download/softdev_warmup_experiment_artefacts/v0.2/

all_graphs.pdf  All plots in one huge PDF.
warmup_results*.json.bz2  Raw results.
Automated Analyses
Automated Analyses: Outlier Detection

outliers outside $5\sigma$ of rolling average
Automated Analyses: Outlier Detection

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #1

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #2

Recurring outliers

Unique outliers (0.05%)
Common outliers (0.40%)

Measurement
Automated Analyses: Change-point Analysis

Fasta:V8:default--javascript, run: 5

Time

Software Development Team

http://soft-dev.org/
Automated Analyses: Change-point Analysis

fannkuch_redux:Hotspot:default-java , run: 1

Time

data.set ls(x)
Automated Analyses: Change-point Analysis

binarytrees:PyPy:default–python , run: 1
Conclusion and Future Work
We can’t rely on the classical warmup model.
Future Work

- Extend automated analyses.

- More \{benchmarks, VMs, arches, OSs\}.

- Try to assign meaning to artefacts in plots.
  - E.g. is that spike at $x = 78$ actually \{GC, compilation, …\}.

- Memory consumption over time.
  - Correlation with iteration times?

- Look at hardware performance counters?

- What else?
References

JIT 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
References

NO_HZ: Reducing Scheduling-Clock Ticks
Linux Kernel Documentation

Intel P-state driver
Linux Kernel Documentation

malloc.conf(5)
OpenBSD Manual Pages
That’s a wrap. Thanks!
Platform independent controls:

- Minimises I/O.
- Consistently limits heap and stack ulimits.
- Drops privileges to a fresh *krun* UNIX account.
- Automatically reboots the system prior to each proc. exec.
- Checks dmesg for changes after each proc. exec.
- Checks system is at same temperature for each proc. exec.
Linux controls. Krun checks:

- Intel P-state support is disabled in the kernel.
- The performance governor is used.
- The kernel is “tickless” (NO_HZ_FULL_ALL).
- The perf sample rate is lowest possible (1Hz).
- ASLR is disabled.

(Note: Linux ignores ulimits)
OpenBSD controls. Krun checks:
- Alloc flags are the least noise inducing.
- `apmd` is running in performance mode.

On OpenBSD:
- We can’t disable ASLR
- We can’t disable ticks.
- We can’t disable P-states in software.
- There is no kernel profiler (good for us).