What Exactly do we Mean by \textit{JIT Warmup}?

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

\textit{SOFTWARE DEVELOPMENT TEAM}

April 20, 2016
JIT Warmup Background

The Back-Story

The Warmup Experiment v2.0

Results

Automated Analyses

Conclusion and Future Work
JIT Warmup Background
Informally:

*Time taken for a JITted VM to reach peak performance*
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

PROFILING INTERPRETER
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

COMPIRATION

PROFILING INTERPRETER
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

COMPILED

PROFILING
INTERPRETER

PEAK PERFORMANCE
JIT Warmup Background

CLASSICAL JIT WARMUP

ITERATION TIME

← WARMUP →

IN-PROCESS ITERATION
JIT Warmup Background

MORE REALISTIC JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION
JIT Warmup Background

MORE REALISTIC JIT WARMUP

ITERATION TIME

SOME NOISE

IN-PROCESS ITERATION

http://soft-dev.org/
MORE REALISTIC JIT WARMUP

ITERATION TIME

IN-PROCESS ITERATION

COMPILATION TIERS

SOME NOISE
JIT Warmup Background

MORE REALISTIC JIT WARMUP

Iteration Time

Compilation Tiers

Some Noise

GC Spikes

In-Process Iteration
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.
The Back-Story
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.
The Warmup Experiment v1.0

- 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 warm up under the classic model.
New goal:

Try to understand why we see “weird” results.
The Warmup Experiment v2.0
Microbenchmarks Revisited

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

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

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

- 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
- GCC-4.9.3 (not really a VM)

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 many more in-process iterations (2000).

Plot results and see if we see classic warmup now.
Richards, Graal, Linux1/i7-4790K, Process execution #3

Classical Warmup
Classical Warmup

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

In-process iteration

Time(s)

In-process iteration

http://soft-dev.org/
(Different machines)
In-process iteration

Fannkuch Redux, LuaJIT, OpenBSD/i7-4790, Process execution #10

Time(s)
Changing Phases

Fasta, LuaJIT, OpenBSD/i7-4790, Process execution #5

Time (s)

In-process iteration
Vastly Inconsistent Process-executions

(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

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

Spectral Norm, PyPy, Linux1/i7-4790K, Process execution #2
Automated Analyses: Outlier Detection

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

Recurring outliers

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

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

Unique outliers (0.05%)

Common outliers (0.40%)
Automated Analyses: Change-point Analysis

fannkuch_redux:Hotspot:default-java, run: 1
Automated Analyses: Change-point Analysis

binarytrees:PyPy:default–python, run: 1

Time

0 500 1000 1500 2000

0.46 0.47 0.48 0.49 0.50

data.set.ts(x)
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, \ldots\}.
- Memory consumption over time.
  - Correlation with iteration times?
- Look at hardware performance counters?
- What else?
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!
Krun Controls

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 \texttt{performance} governor is used.
- The kernel is “tickless” (\texttt{NO_HZ_FULL_ALL}).
- The \texttt{perf} sample rate is lowest possible (1Hz).
- ASLR is disabled.

(Note: Linux ignores ulimits)
OpenBSD controls. Krun checks:
- Malloc 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).