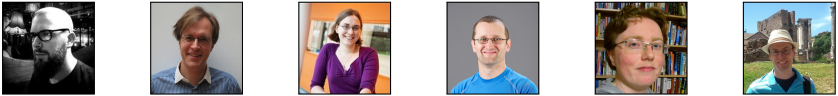
VM Warmup Blows Hot and Cold



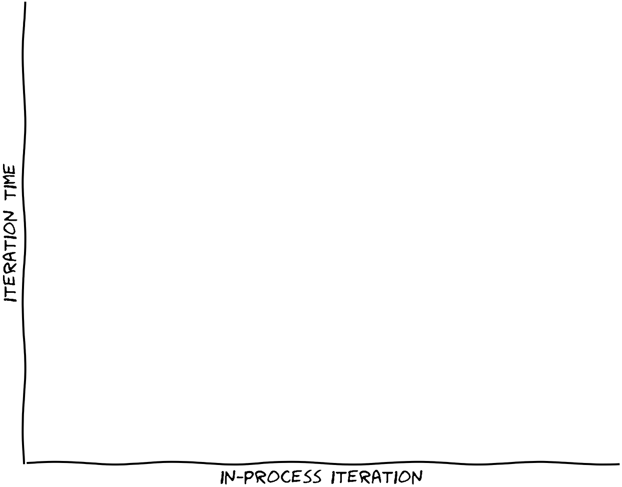
Edd Barrett, Carl Friedrich Bolz, Rebecca Killick (Lancaster),
Vincent Knight (Cardiff), Sarah Mount, Laurence Tratt

KING'S College LONDON

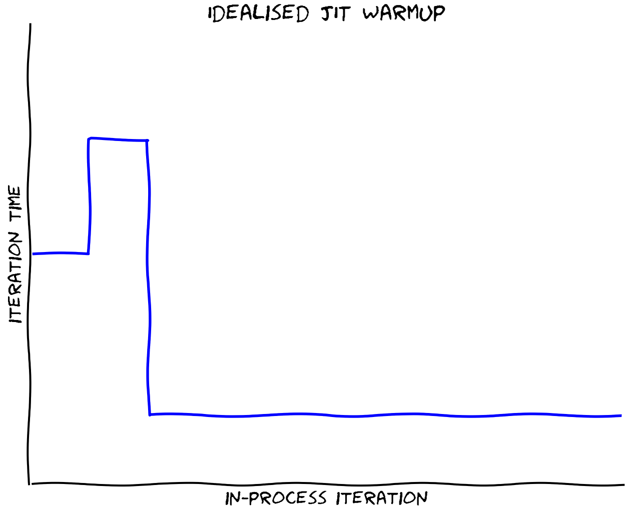
Software Development Team

2016-09-27

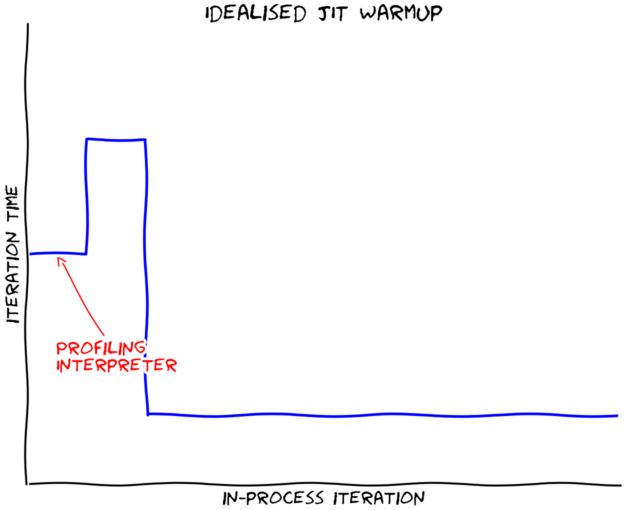
What’s ‘Warmup”?



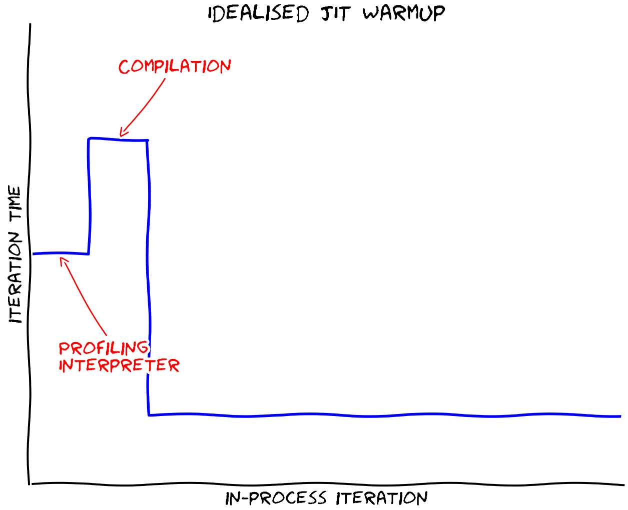
With a JIT



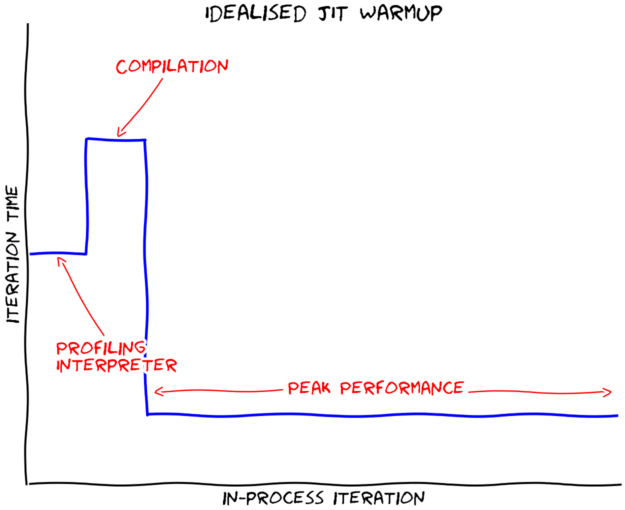
With a JIT


IDEALISED JIT WARMUP
ITERATION TIME
PROFILING INTERPRETER
IN-PROCESS ITERATION

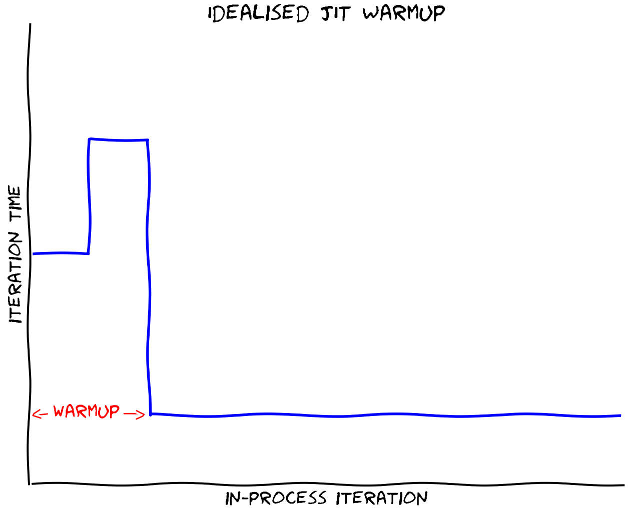
With a JIT


IDEALISED JIT WARMUP
COMPILATION
ITERATION TIME
PROFILING
INTERPRETER
IN-PROCESS ITERATION

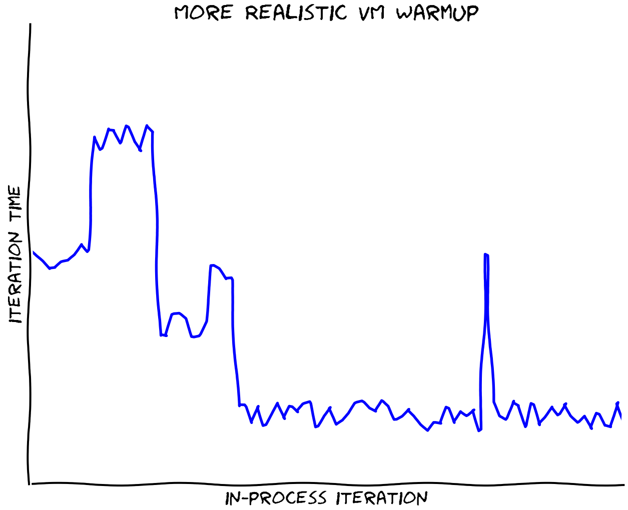
With a JIT

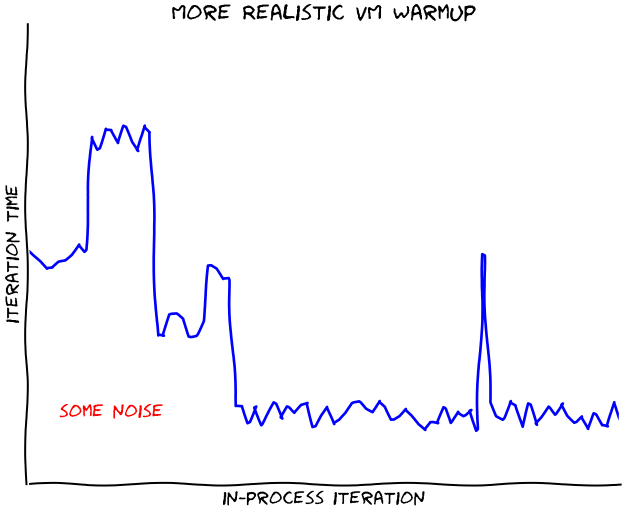

IDEALISED JIT WARMUP
COMPILATION
ITERATION TIME
PROFILING INTERPRETER
PEAK PERFORMANCE
IN-PROCESS ITERATION

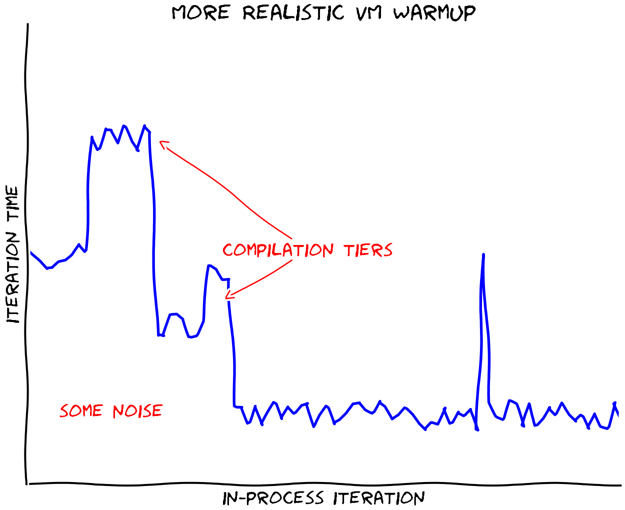
With a JIT

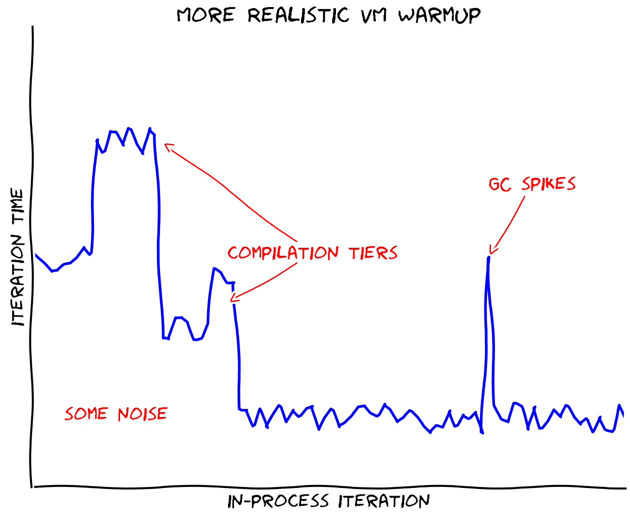


IDEALISED JIT WARMUP
ITERATION TIME
WARMUP
IN-PROCESS ITERATION

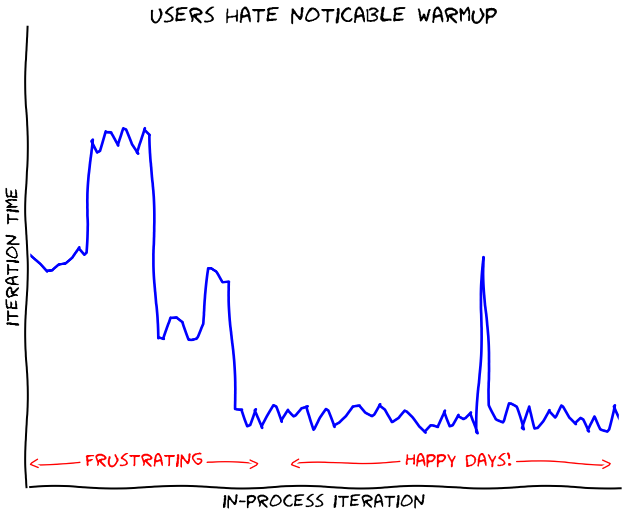

MORE REALISTIC VM WARMUP
ITERATION TIME
IN-PROCESS ITERATION

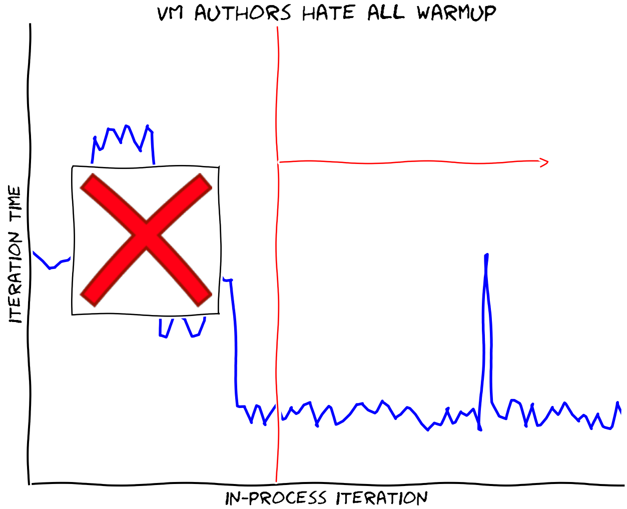

MORE REALISTIC VM WARMUP
ITERATION TIME
SOME NOISE
IN-PROCESS ITERATION


MORE REALISTIC VM WARMUP
ITERATION TIME
COMPILATION TIERS
SOME NOISE
IN-PROCESS ITERATION


MORE REALISTIC VM WARMUP
ITERATION TIME
IN-PROCESS ITERATION
COMPILATION TIERS
GC SPIKES
SOME NOISE

Why care?


USERS HATE NOTICABLE WARMUP
ITERATION TIME
FRUSTRATING
HAPPY DAYS!
IN-PROCESS ITERATION


VM AUTHORS HATE ALL WARMUP
ITERATION TIME
IN-PROCESS ITERATION

Warmup is bad for everyone.

The Warmup Experiment

Measure warmup of modern language implementations

The Warmup Experiment

Measure warmup of modern language implementations

Hypothesis: Small, deterministic programs exhibit classical warmup behaviour

Method 1: Which benchmarks?

The language benchmark games are perfect for us (unusually)

Method 1: Which benchmarks?

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*

10 *process executions*

Method 3: VMs

- Graal-0.13
- HHVM-3.12.0
- JRuby/Truffle (git #f82ac771)
- Hotspot-8u72b15
- LuaJit-2.0.4
- PyPy-4.0.1
- V8-4.9.385.21
- GCC-4.9.3

Note: same GCC (4.9.3) used for all compilation

Method 4: Machines

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

Method 4: Machines

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

- Turbo boost and hyper-threading disabled
- SSH blocked from non-local machines
- Daemons disabled (cron, smtpd)

Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible

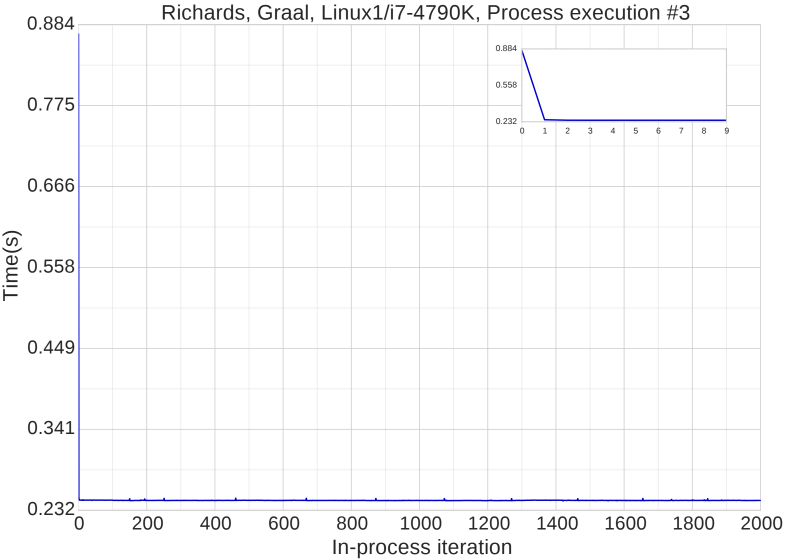
Method 5: Krun

Benchmark runner: tries to control as many confounding variables as possible e.g.:

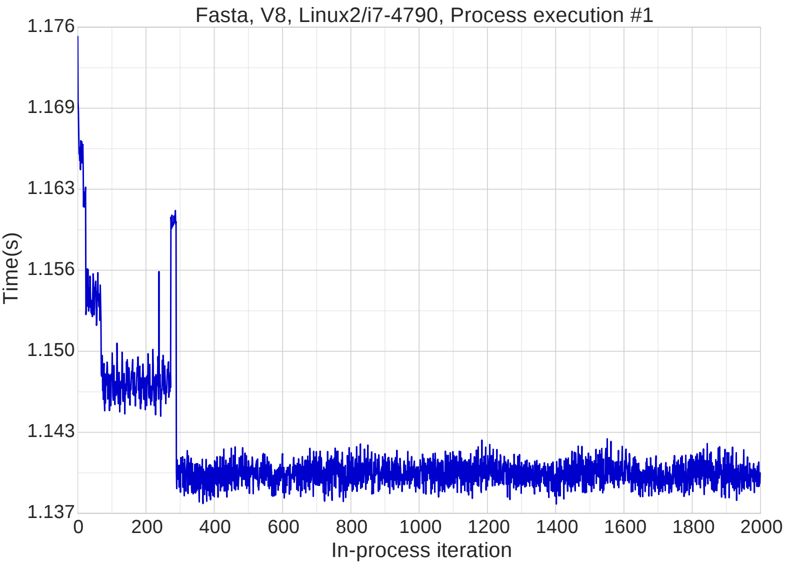
- Minimises I/O
- Sets fixed heap and stack ulimits
- Drops privileges to a ‘clean’ user account
- Automatically reboots the system prior to each proc. exec
- Checks `dmesg` for changes after each proc. exec
- Checks system at (roughly) same temperature for proc. execs
- Enforces kernel settings (tickless mode, CPU governors, ...)

Preliminary results

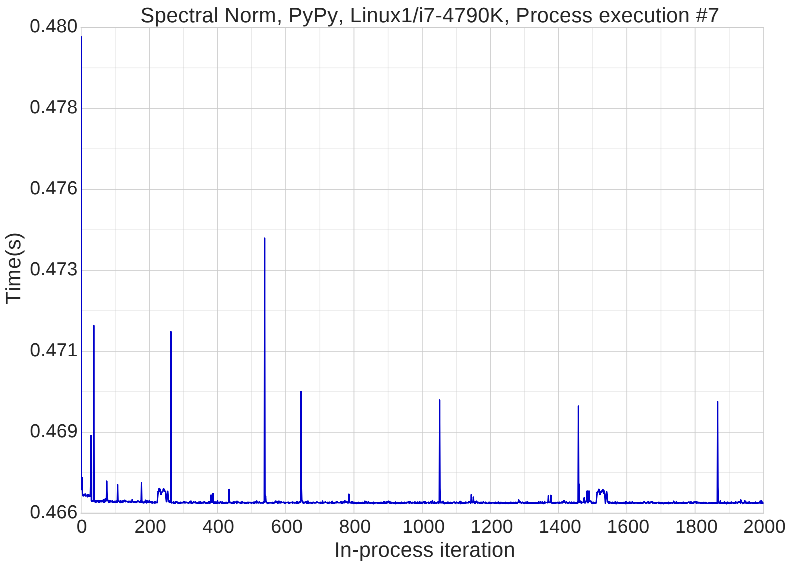
Classical Warmup



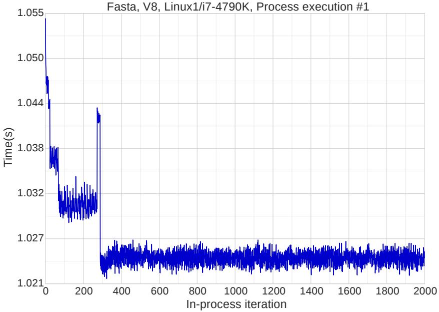
Classical Warmup

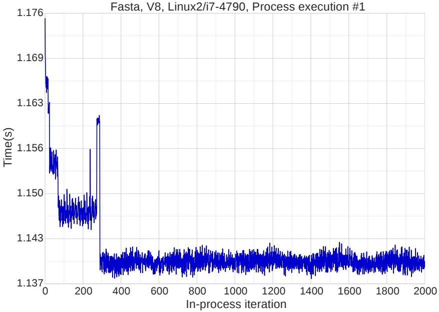


Classical Warmup



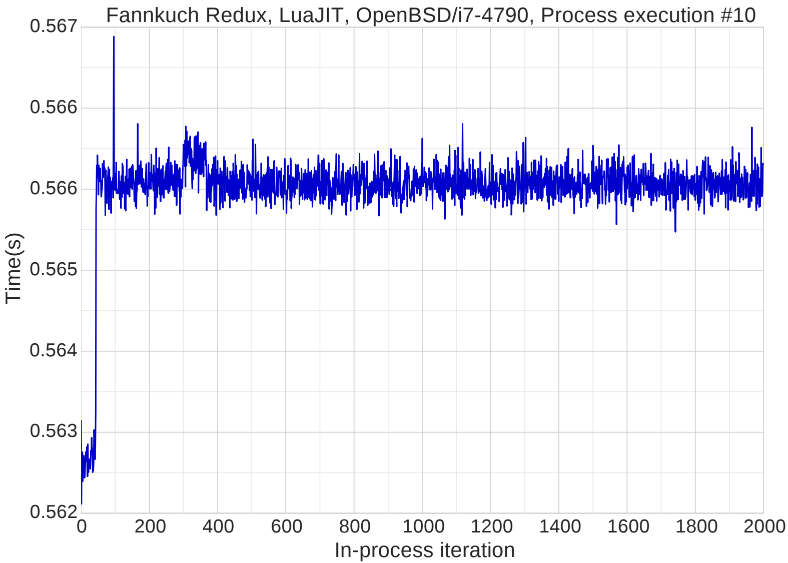
Classical Warmup



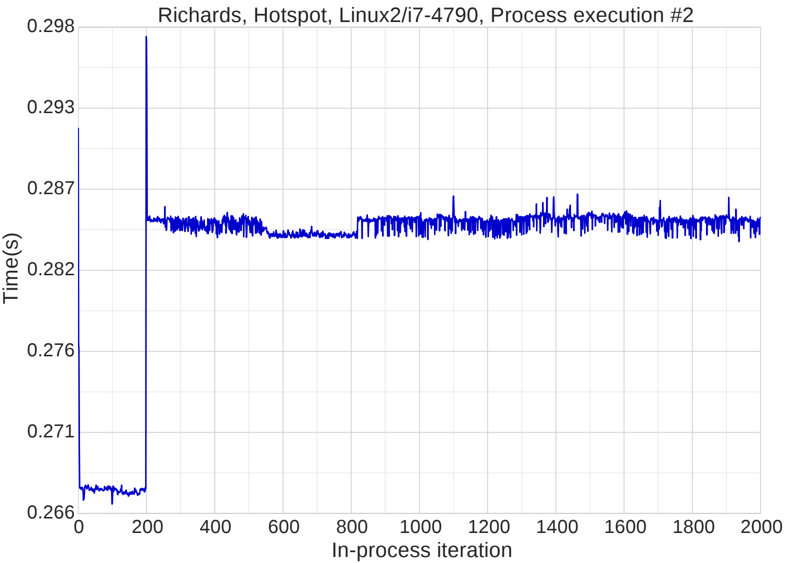


(Different machines)

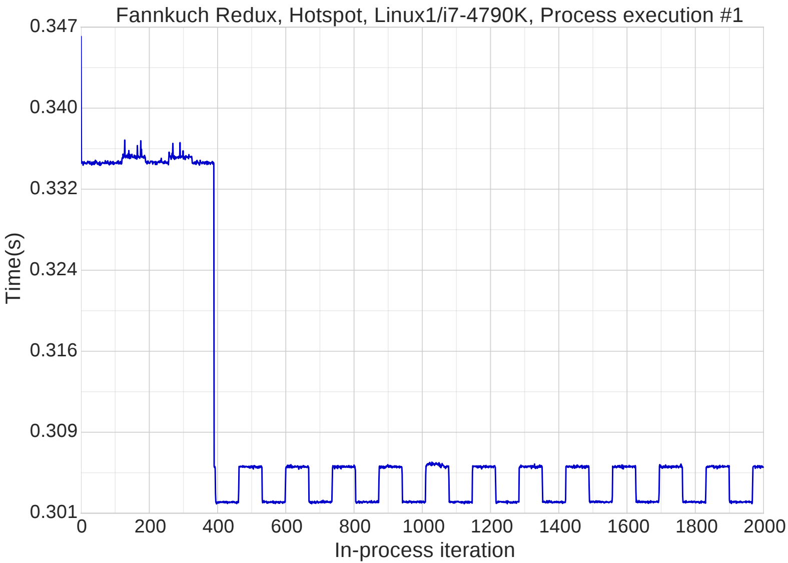
Slowdown


Fannkuch Redux, LuaJIT, OpenBSD/i7-4790, Process execution #10
Time(s)
0.567
0.566
0.566
0.565
0.564
0.563
0.562
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

Slowdown

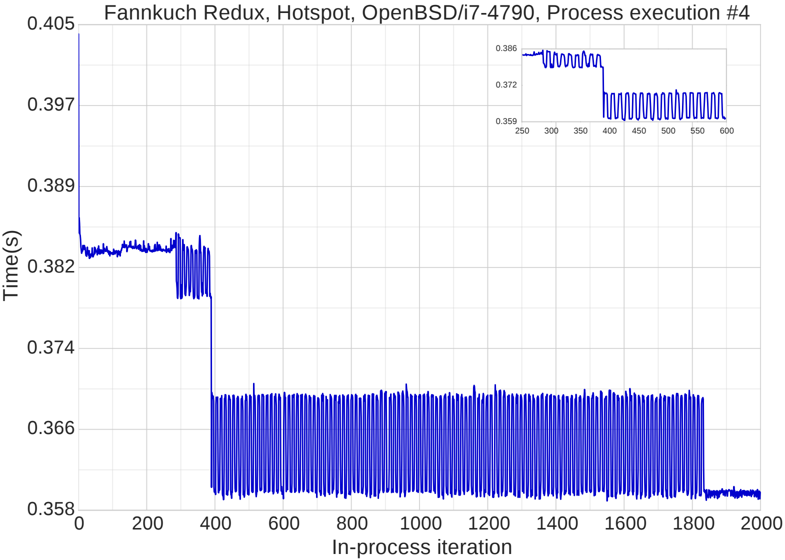

Richards, Hotspot, Linux2/i7-4790, Process execution #2
0.298
0.293
0.287
0.282
0.276
0.271
0.266
Time(s)
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

Cycles



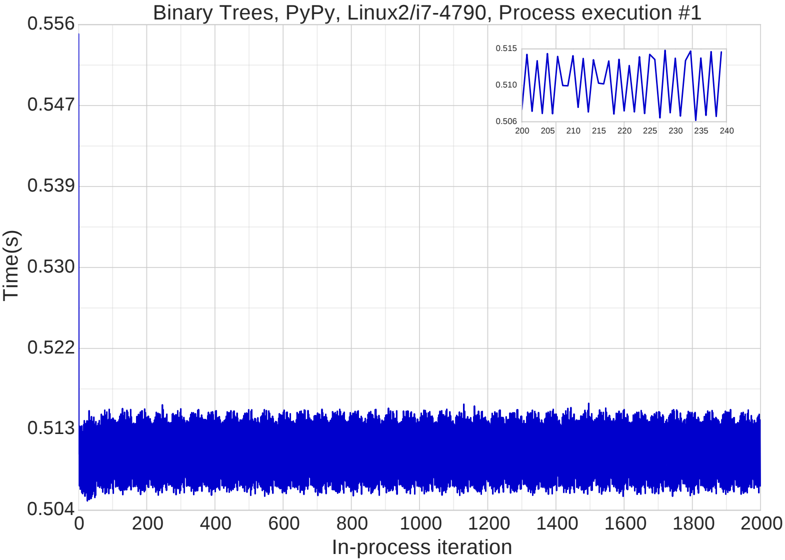
Fannkuch Redux, Hotspot, Linux1/i7-4790K, Process execution #1
Time(s)
0.347
0.340
0.332
0.324
0.316
0.309
0.301
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration

Cycles

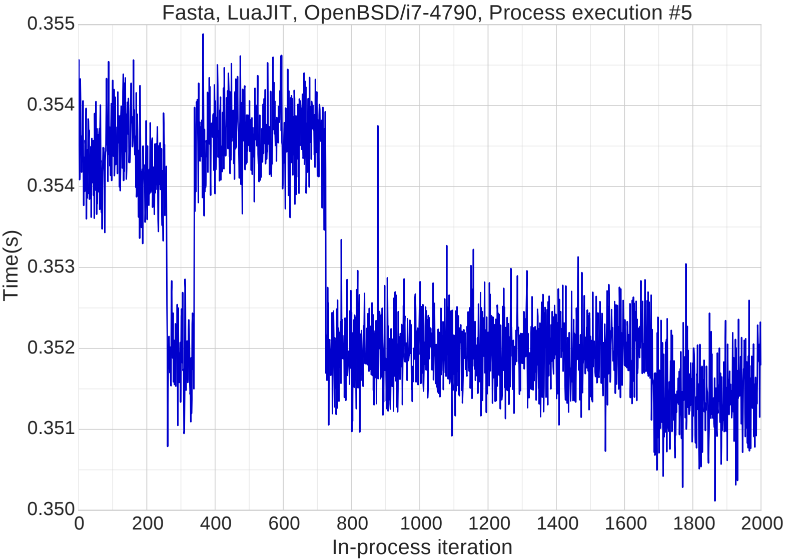


Fannkuch Redux, Hotspot, OpenBSD/i7-4790, Process execution #4
Time(s)
0.405
0.397
0.389
0.382
0.374
0.366
0.358
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration
0.386
0.372
0.359
250
300
350
400
450
500
550
600

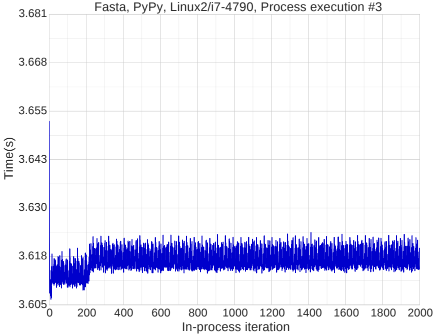
Cycles

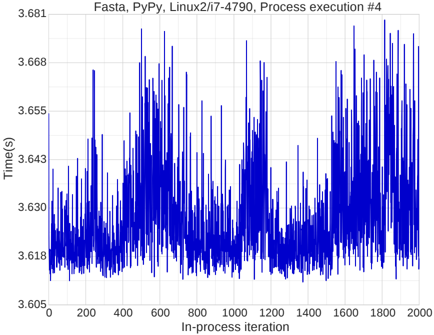

Binary Trees, PyPy, Linux2/i7-4790, Process execution #1
Time(s)
0.556
0.547
0.539
0.530
0.522
0.513
0.504
0
200
400
600
800
1000
1200
1400
1600
1800
2000
In-process iteration
0.515
0.510
0.506
200
205
210
215
220
225
230
235
240

Never-ending Phase Changes



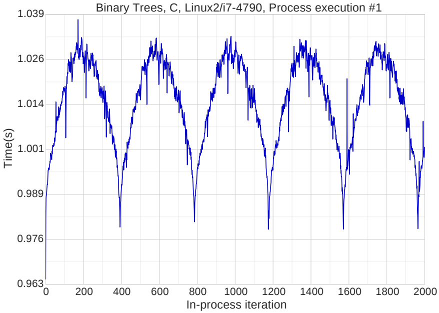
Inconsistent Process-executions

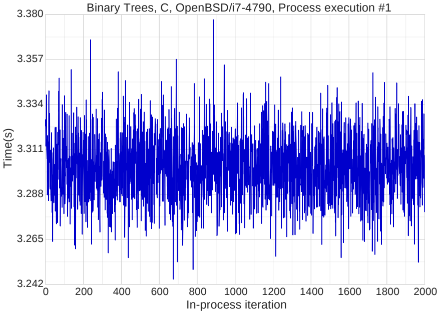




(Note: same machine)

Inconsistent Process-executions





(Note: different machines. Bouncing ball pattern Linux-specific)

Summary

Classical warmup occurs for only:

Summary

Classical warmup occurs for only:

50% of process executions

Summary

Classical warmup occurs for only:

50% of process executions

25% of (VM, benchmark) pairs

Summary

Classical warmup occurs for only:

50% of process executions

25% of (VM, benchmark) pairs

0% of benchmarks for all VMs, machines & proc execs.

Hypothesis Invalidated

~~*Hypothesis*: Small, deterministic programs exhibit classical warmup behaviour~~

Open Questions

Open Questions

How can we measure anything any more?

Open Questions

How can we measure anything any more?

For how long has this been going on?

Open Questions

How can we measure anything any more?

For how long has this been going on?

Is this really the fault of the VMs?

Ongoing/Future Work

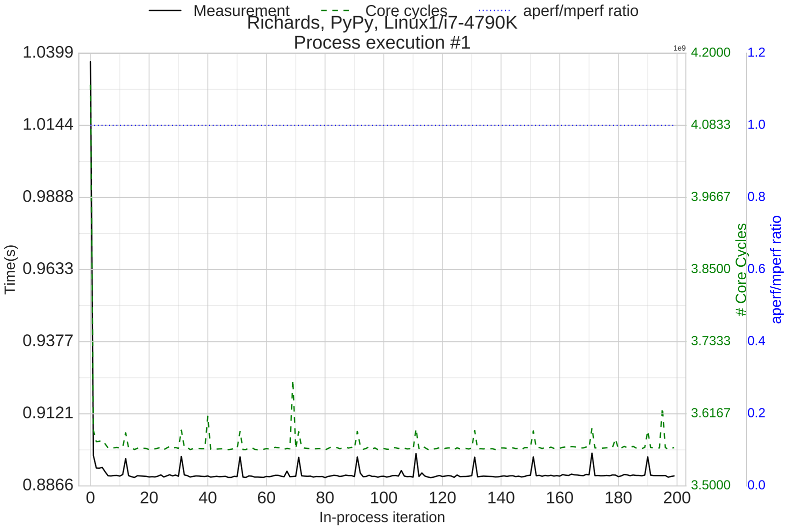
(a.k.a. “making sense of our results”)

Performance Counters

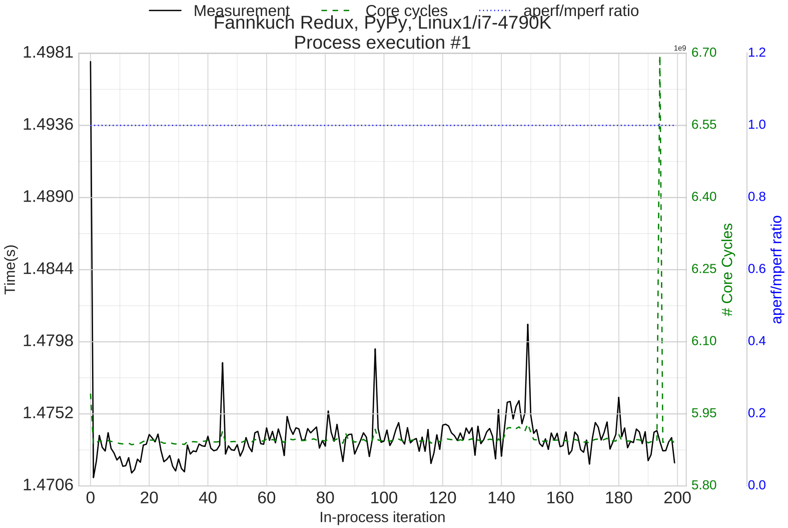
- `CPU_CLK_UNHALTED.CORE`
  - Counts the number of “core cycles” executed per-core.

- `IA32_APERF` / `IA32_MPERF` ratio.
  - `IA32_MPERF` increments at a fixed reference frequency.
  - `IA32_APERF` increments proportional to “actual performance”.
  - The ratio of two delta’s indicates if clock speed changed.

Plotting performance counters



Plotting performance counters

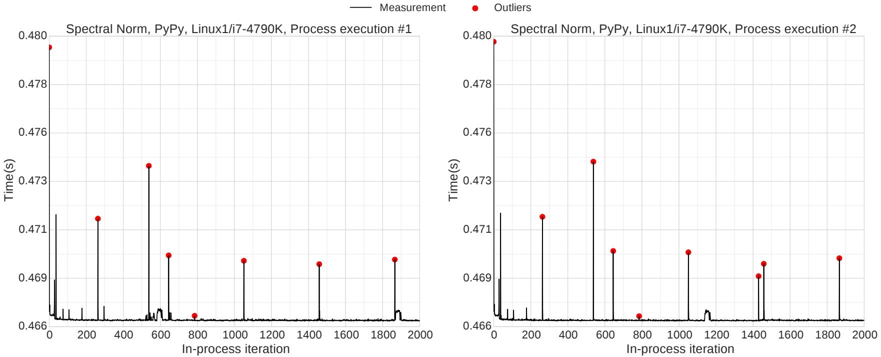


VM Instrumentation

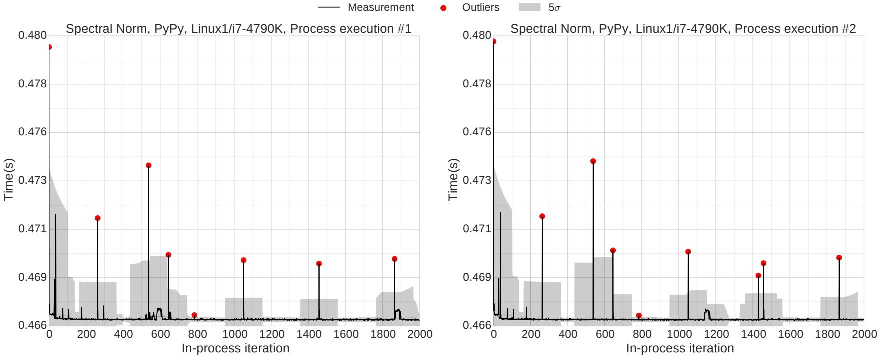
For PyPy and Hotspot, record:

- Time spent in GC.
- Time spent in Compilation.

Outlier Detection

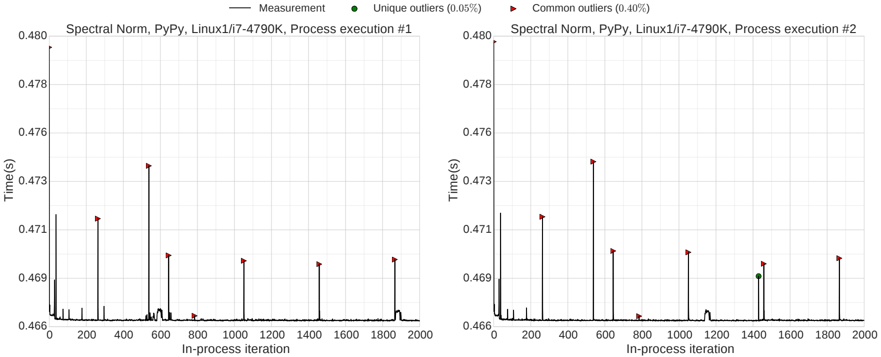


Outlier Detection



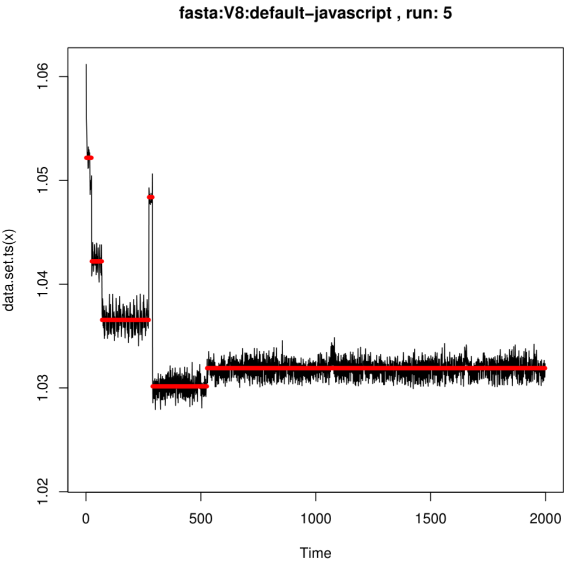
Outliers outside 5σ of rolling average

Outlier Detection

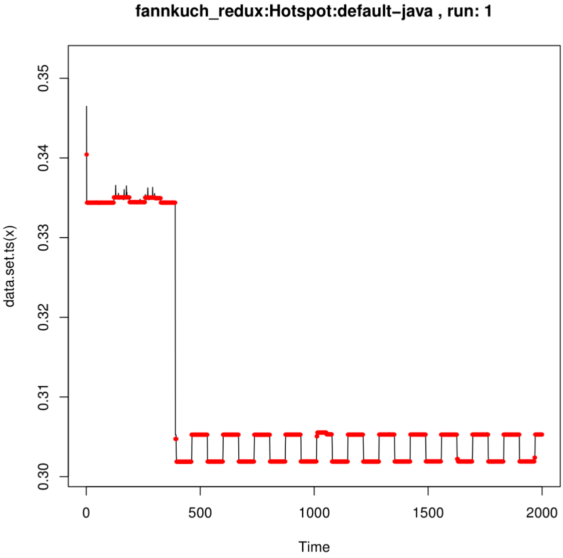


Recurring outliers

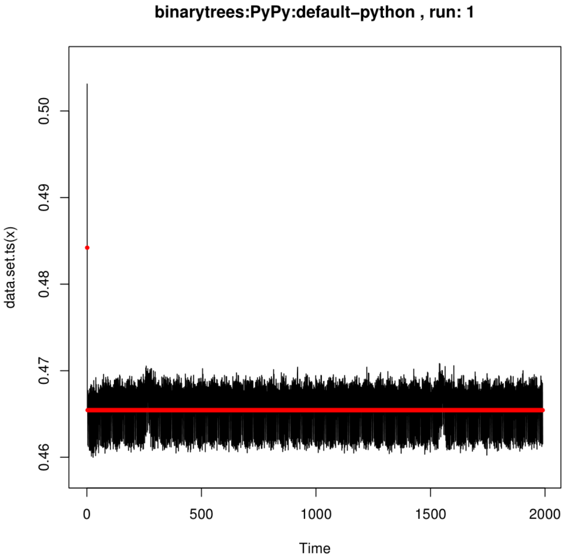
Change-point Analysis



Change-point Analysis



Change-point Analysis



Full (Preliminary) Results

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.

(Note: newer results available)

References

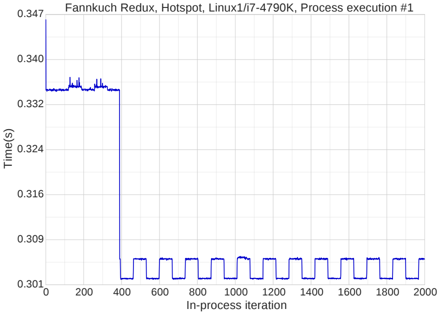
VM Warmup Blows Hot and Cold
E. Barrett, C. F. Bolz, R. Killick, V. Knight, S. Mount and L. Tratt.

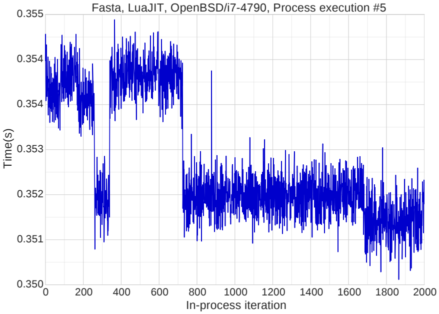
Rigorous Benchmarking in Reasonable Time
T. Kalibera and R. Jones

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


Fannkuch Redux, Hotspot, Linux1/i7-4790K, Process execution #1


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