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

How to benchmark a JIT:

How to benchmark a JIT:

- Invoke VM

How to benchmark a JIT:

- Invoke VM
  - *“process execution”*

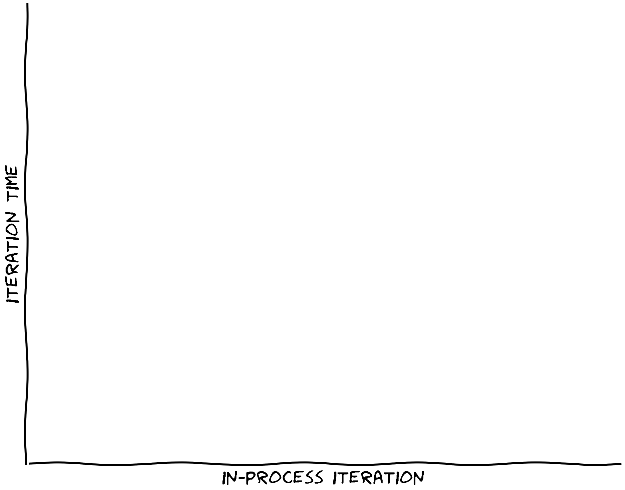
How to benchmark a JIT:

- Invoke VM
  - *"process execution"*
- Run Benchmark in a Loop

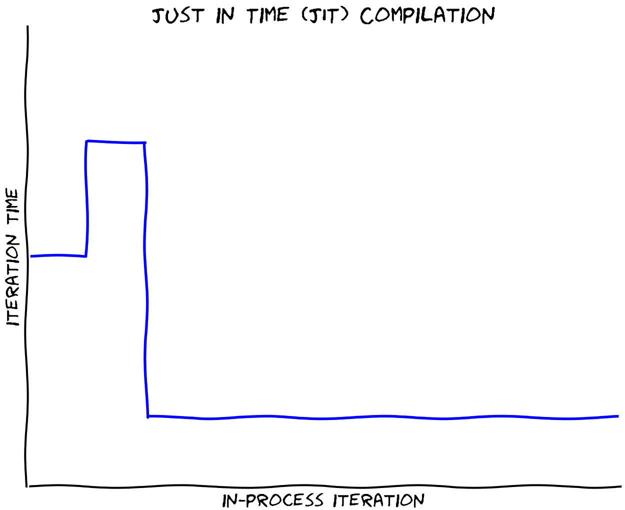
How to benchmark a JIT:

- Invoke VM
  - *“process execution”*
- Run Benchmark in a Loop
  - *“in-process iterations”*

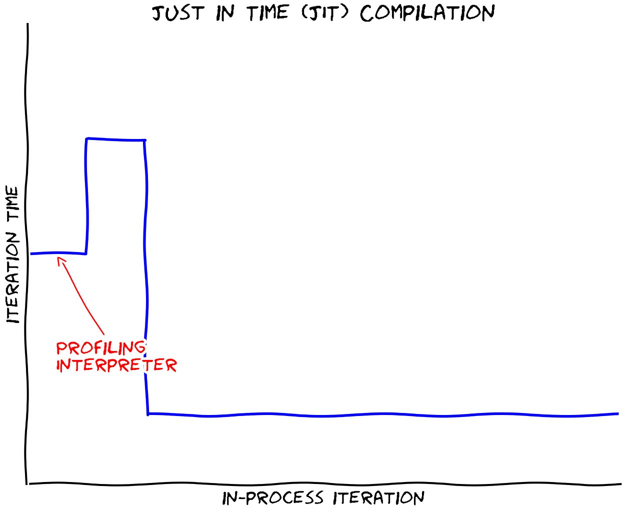
The Current State of the Art of Benchmarking



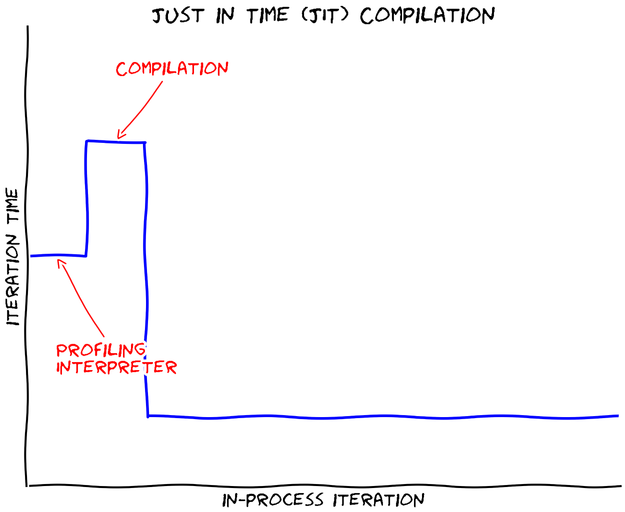
The Current State of the Art of Benchmarking



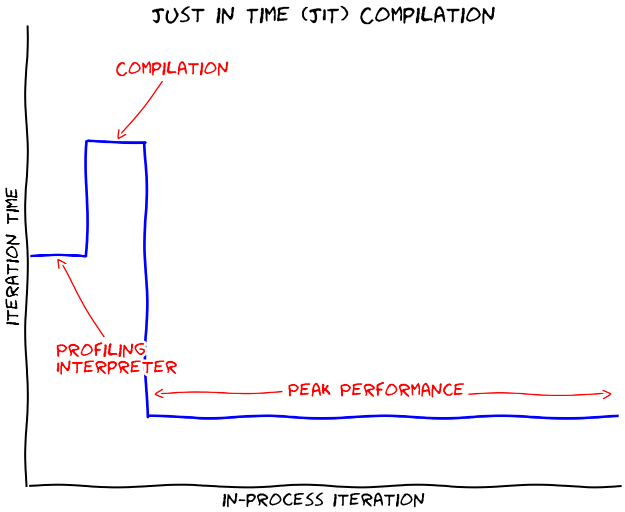
The Current State of the Art of Benchmarking



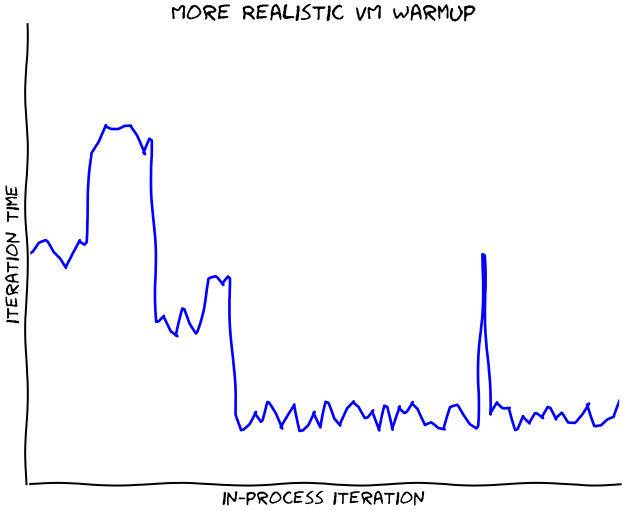
The Current State of the Art of Benchmarking



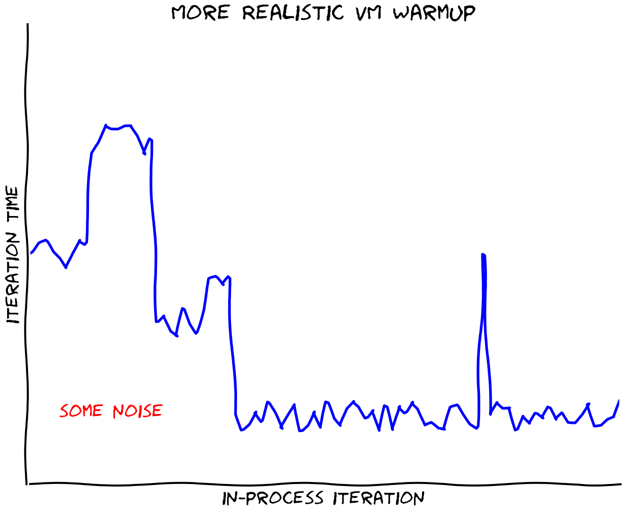
The Current State of the Art of Benchmarking

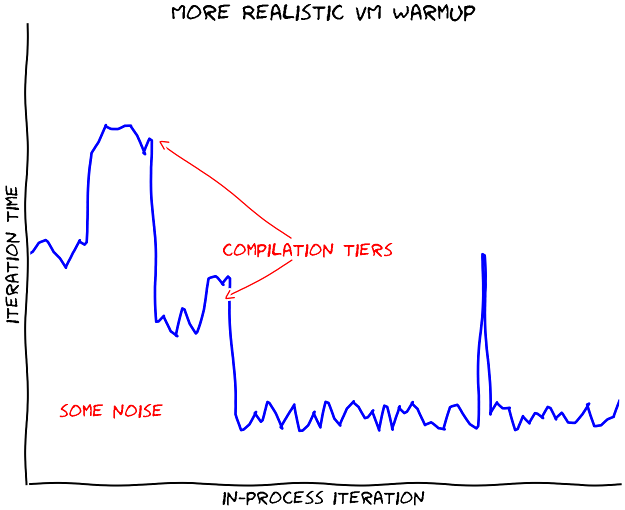


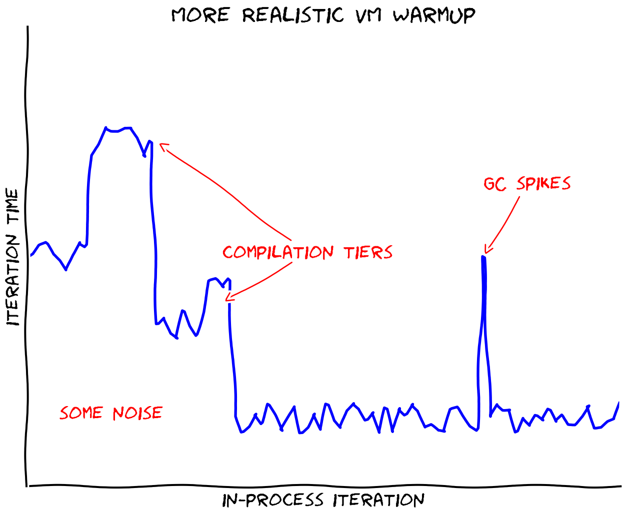
The Current State of the Art of Benchmarking



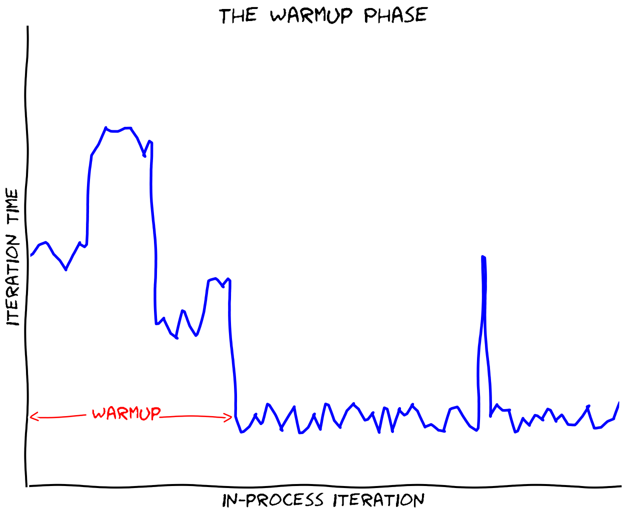
The Current State of the Art of Benchmarking




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


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

The Current State of the Art of Benchmarking



1. Users dislike poor warmup.

Warmup Matters

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!

The Warmup Experiment

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.

Method 1: Which Benchmarks?

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_{4790}, Debian 8, 24GiB RAM
- $\text{Linux}_{\text{E3-1240v5}}$, Debian 8, 32GiB RAM
- OpenBSD_{4790}, OpenBSD 6.0, 32GiB RAM

Method 4: Machines

- Linux_{4790}, Debian 8, 24GiB RAM
- $\text{Linux}_{\text{E3-1240v5}}$, Debian 8, 32GiB RAM
- OpenBSD_{4790}, OpenBSD 6.0, 32GiB RAM

- Turbo boost off.
- Hyper-threading off.

Method 5: Benchmark Harness

KRUN

Control as many confounding variables as possible

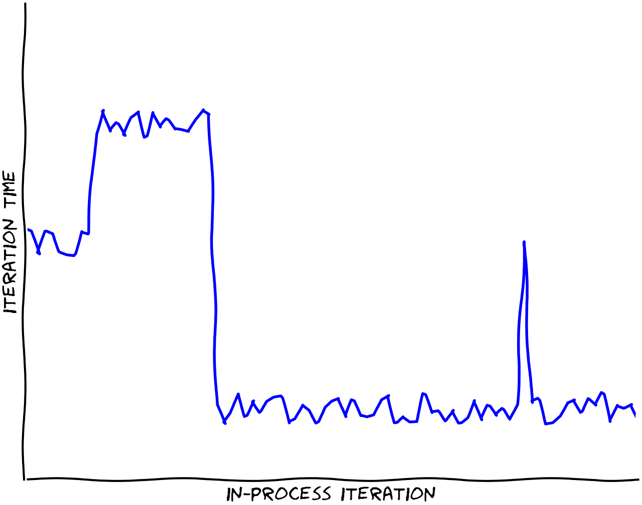
Method 5: Benchmark Harness

KRUN

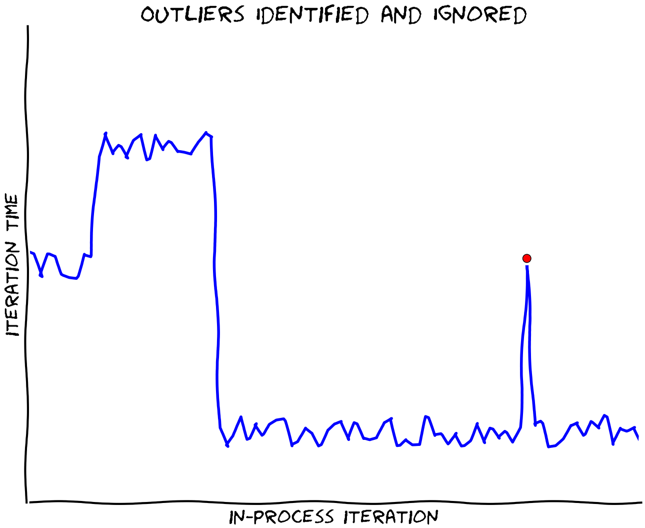
Control as many confounding variables as possible

- 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, ...)

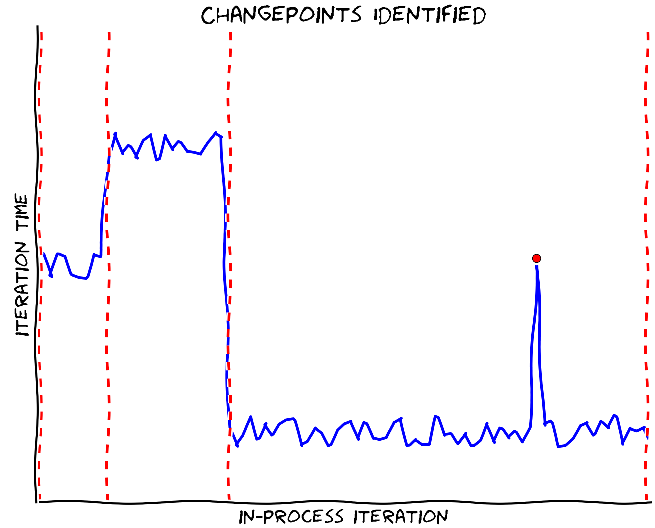
Method 6: Changepoint Analysis and Classification



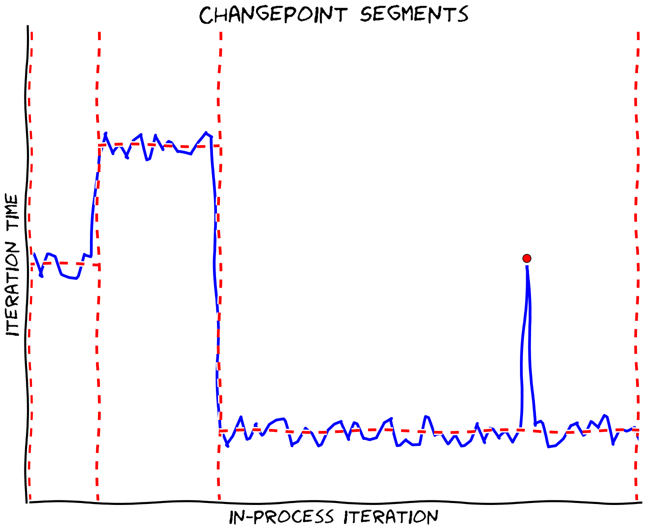
Method 6: Changepoint Analysis and Classification



Method 6: Changepoint Analysis and Classification



Method 6: Changepoint Analysis and Classification



- All segs are equivalent:
  FLAT ($-$)

Method 6: Changepoint Analysis and Classification

- All segs are equivalent:
  FLAT (—)
- Length(final equivalent segs) >= 500 iters:
  - and are the fastest segs:
    WARMUP (⌉⌊)

Method 6: Changepoint Analysis and Classification

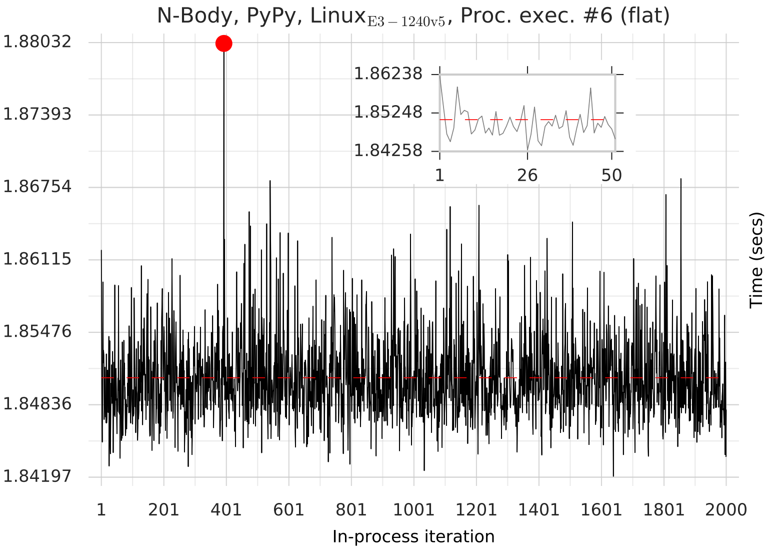
- All segs are equivalent:
  FLAT (—)
- Length(final equivalent segs) >= 500 iters:
  - and are the fastest segs:
    WARMUP (⎺⎽)
  - but not the fastest segs:
    SLOWDOWN (⎽⎺)

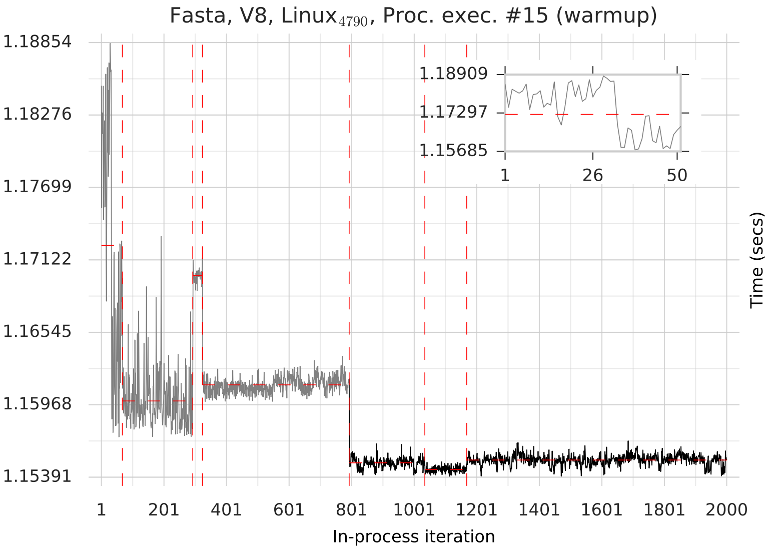
Method 6: Changepoint Analysis and Classification

- All segs are equivalent:
  FLAT (—)
- Length(final equivalent segs) >= 500 iters:
  - and are the fastest segs:
    WARMUP ()
  - but not the fastest segs:
    SLOWDOWN ()
- Otherwise:
  NO STEADY STATE ()

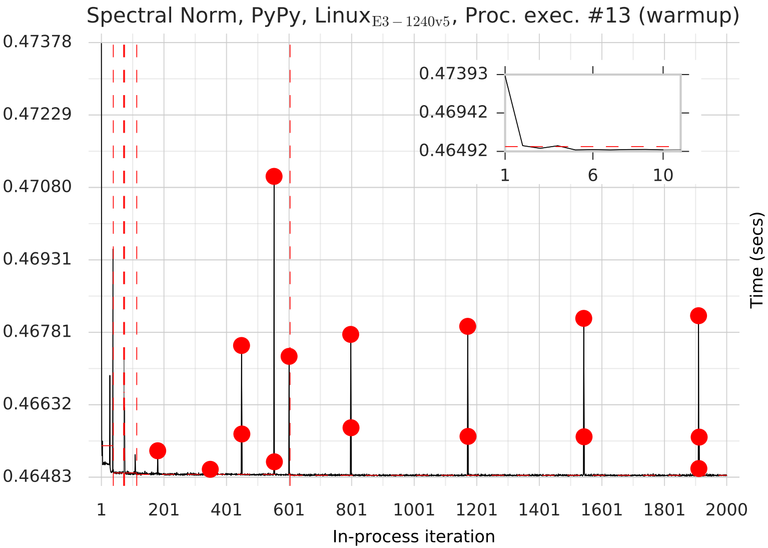
Results

Results: Flat

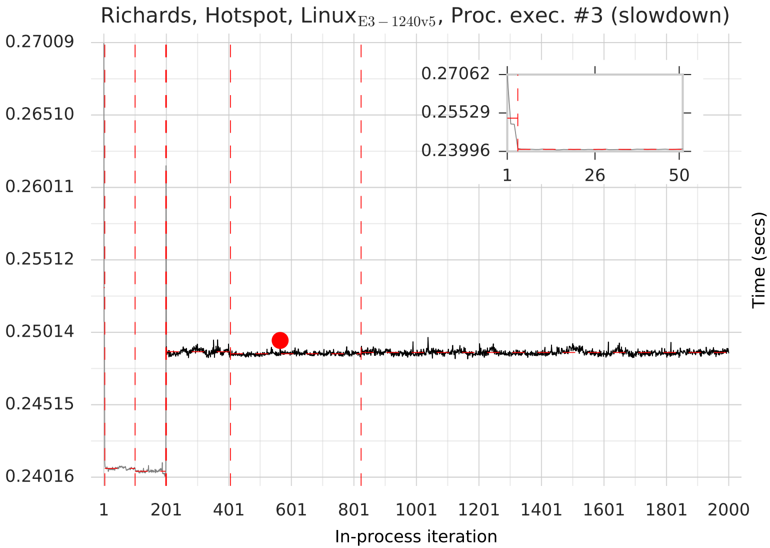



Fasta, V8, Linux$_{4790}$, Proc. exec. #15 (warmup)
1.18854
1.18276
1.17699
1.17122
1.16545
1.15968
1.15391
1.18909
1.17297
1.15685
1
26
50
Time (secs)
1
201
401
601
801
1001
1201
1401
1601
1801
2000
In-process iteration

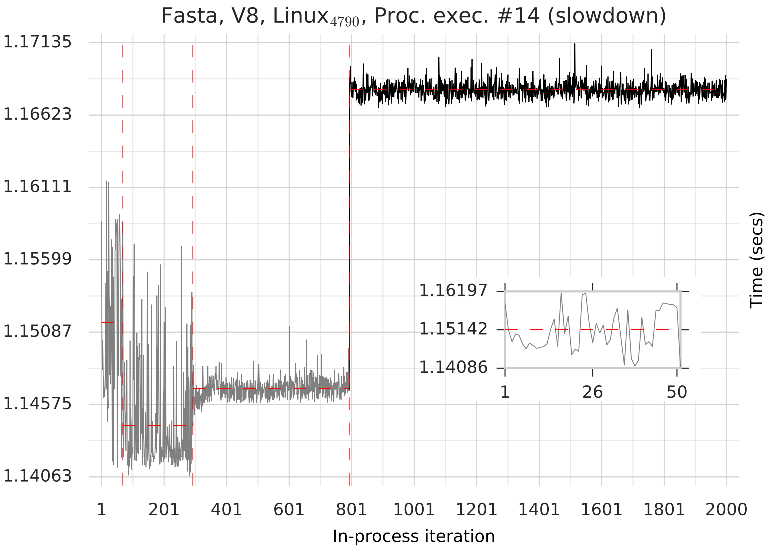
Results: Warmup



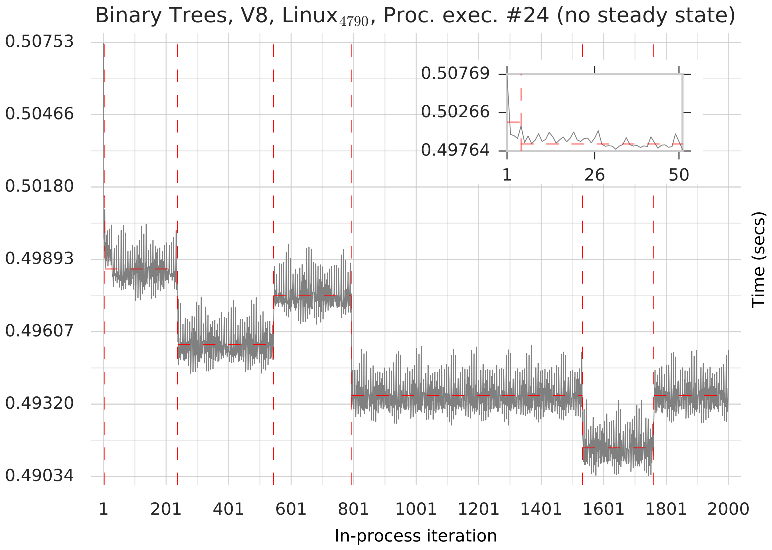
Results: Slowdown



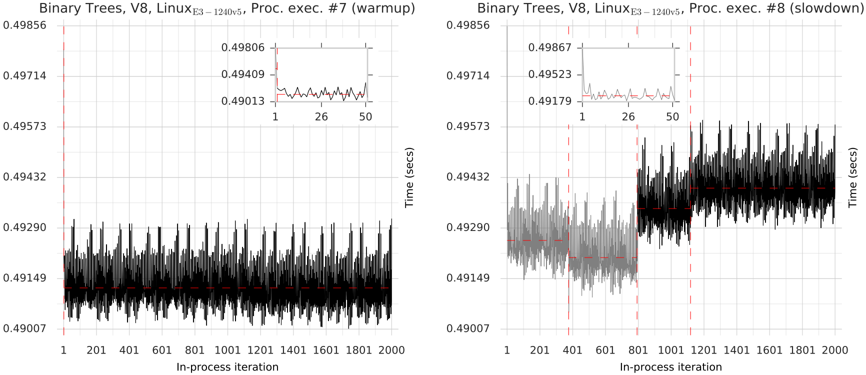
Results: Slowdown



Results: No Steady State



Results: Inconsistent Process-executions



(Same machine)

Quantitative Results

VM	Benchmark	Class.	Steady iter (#)	Steady iter (s)	Steady perf (s)	Benchmark	Class.	Steady iter (#)	Steady iter (s)	Steady perf (s)
C	*binary trees*	ʍ				*n-body*	—			0.40555 ±0.005110
Graal		≭ (27⅂, 3⅃)	32.0 (17.0, 193.8)	6.60 (3.729, 36.608)	0.18594 ±0.000315		⅂	8.0 (7.5, 8.5)	1.22 (1.126, 1.358)	0.13334 ±0.000045
HHVM		≭ (24⅂, 4⅃, 2ʍ)					≭ (16⅂, 11⅃, 3ʍ)			
HotSpot		≭ (25⅂, 5⅃)	7.0 (7.0, 53.5)	1.19 (1.182, 9.703)	0.18279 ±0.000116		⅂	2.0 (2.0, 2.0)	0.14 (0.141, 0.143)	0.13699 ±0.000032
JRuby+Truffle		⅃	1082.0 (999.0, 1232.5)	2219.59 (2039.304, 2516.021)	2.05150 ±0.017738		⅂	69.0 (69.0, 70.0)	17.95 (17.716, 18.127)	0.20644 ±0.001568
LuaJIT		≭ (23⅂, 4⅃, 2—, 1ʍ)					—			0.25399 ±0.004471
PyPy		≭ (27⅃, 3ʍ)					—			1.85835 ±0.012893
V8		≭ (15—, 9⅂, 6⅃)	1.5 (1.0, 794.0)	0.25 (0.000, 391.026)	0.49237 ±0.003198		= (25—, 5⅂)	1.0 (1.0, 361.6)	0.00 (0.000, 87.367)	0.24138 ±0.000389
C	*fannkuch redux*	≭ (21—, 6⅂, 2⅃, 1ʍ)				*Richards*	≭ (19—, 5⅃, 4ʍ, 2⅂)			
Graal		≭ (28⅂, 1ʍ, 1⅃)					≭ (28⅂, 1ʍ, 1⅃)			
HHVM		⅂	10.0 (10.0, 10.0)	52.66 (52.660, 52.708)	1.35779 ±0.011948					
HotSpot		⅂	390.0 (2.0, 390.0)	153.70 (0.407, 155.254)	0.36202 ±0.022767		≭ (26⅃, 2ʍ, 2⅂)			
JRuby+Truffle		⅃	1016.5 (999.0, 1023.1)	1039.04 (1014.290, 1059.967)	1.08833 ±0.038580		⅃	1021.0 (1014.9, 1027.0)	917.30 (901.708, 946.683)	0.89509 ±0.027590
LuaJIT		—			0.56285 ±0.000097		≭ (21ʍ, 7⅃, 2⅂)			
PyPy		≭ (15⅂, 13—, 2⅃)	2.0 (1.0, 28.9)	1.57 (0.000, 43.483)	1.55442 ±0.020549		⅂	2.0 (2.0, 5.0)	1.12 (1.114, 4.054)	0.96809 ±0.010950
V8		= (19⅂, 11—)	2.0 (1.0, 25.5)	0.31 (0.000, 7.525)	0.30401 ±0.000154		= (29⅂, 1—)	4.0 (4.0, 16.0)	1.44 (1.434, 7.135)	0.47421 ±0.001218
C	*fasta*	—			0.07048 ±0.000210	*spectralnorm*	≭ (28⅂, 1—, 1⅃)	997.0 (2.0, 1001.0)	546.38 (0.546, 547.717)	0.54547 ±0.002562
Graal		≭ (29⅂, 1ʍ)					≭ (29⅃, 1—)	15.0 (2.0, 21.0)	12.40 (0.812, 17.804)	0.89293 ±0.000087
HHVM		≭ (27⅂, 2ʍ, 1⅃)					⅂	35.0 (34.0, 41.0)	139.18 (136.909, 147.626)	1.40690 ±0.011133
HotSpot		≭ (18⅂, 12⅃)	261.0 (6.0, 595.0)	30.73 (0.614, 70.021)	0.11744 ±0.001723		⅂	7.0 (7.0, 8.6)	1.90 (1.901, 2.425)	0.31470 ±0.000029
JRuby+Truffle							⅂	1011.0 (1007.5, 1014.0)	893.38 (888.985, 896.562)	0.83633 ±0.014403
LuaJIT		ʍ					—			0.22435 ±0.000020
PyPy		ʍ					⅂	75.0 (75.0, 75.0)	34.43 (34.429, 34.437)	0.46489 ±0.000046
V8		≭ (19⅂, 10⅃, 1ʍ)					⅂	3.0 (3.0, 3.0)	0.55 (0.554, 0.554)	0.24963 ±0.000039

Results

		Class.	Steady iter (#)	Steady iter (s)	Steady perf (s)
C	*binary trees*	ʍ			
Graal		≍ (27˥, 3ʃ)	32.0 (17.0, 193.8)	6.60 (3.729, 36.608)	0.18594 ±0.000316
HHVM		≍ (24˥, 4ʃ, 2ʍ)			
HotSpot		≍ (25˥, 5ʃ)	7.0 (7.0, 53.5)	1.19 (1.182, 9.703)	0.18279 ±0.000116
JRuby+Truffle		ʃ	1082.0 (999.0, 1232.5)	2219.59 (2039.304, 2516.021)	2.05150 ±0.017737
LuaJIT		≍ (23˥, 4ʃ, 2–, 1ʍ)			
PyPy		≍ (27ʃ, 3ʍ)			
V8		≍ (15–, 9˥, 6ʃ)	1.5 (1.0, 794.0)	0.25 (0.000, 391.026)	0.49237 ±0.003198

Results: Summary

Class.	Linux_{4790}	Linux_{1240v5}	OpenBSD_{4790}
	⟨VM, benchmark⟩ pairs		
—	8.7%	13.0%	6.7%
ʅ	28.3%	23.9%	10.0%
ʃ	6.5%	6.5%	0.0%
∿	4.3%	6.5%	0.0%
=	6.5%	6.5%	13.3%
⨯	45.7%	43.5%	70.0%
	Process executions		
—	26.4%	20.9%	34.0%
ʅ	48.3%	51.5%	52.1%
ʃ	16.7%	17.9%	11.1%
∿	8.7%	9.6%	2.8%

Results: Summary

Class.	Linux_{4790}	Linux_{1240v5}	OpenBSD_{4790}
	⟨VM, benchmark⟩ pairs		
—	8.7%	13.0%	6.7%
⌉	28.3%	23.9%	10.0%
⌈	6.5%	6.5%	0.0%
〰	4.3%	6.5%	0.0%
=	6.5%	6.5%	13.3%
⋇	45.7%	43.5%	70.0%
	Process executions		
—	26.4%	20.9%	34.0%
⌉	48.3%	51.5%	52.1%
⌈	16.7%	17.9%	11.1%
〰	8.7%	9.6%	2.8%

Best case no. good process executions: 86.1%

Results: Summary

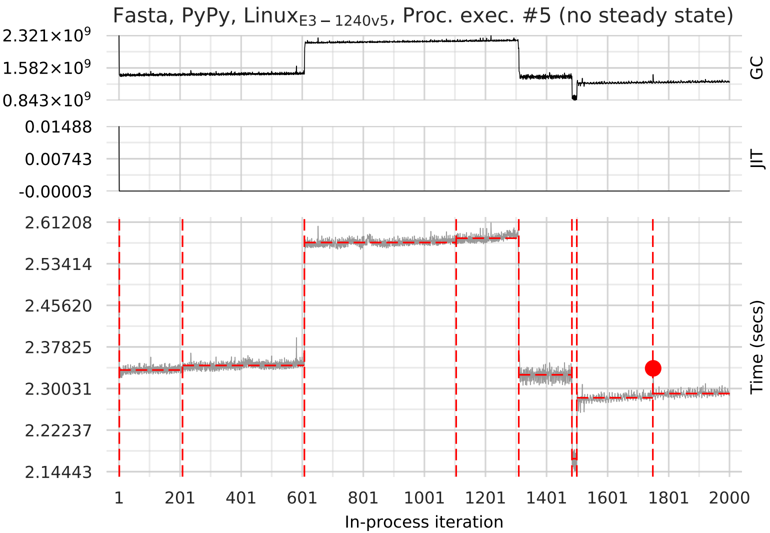
Class.	Linux_{4790}	Linux_{1240v5}	OpenBSD_{4790}
	⟨VM, benchmark⟩ pairs		
—	8.7%	13.0%	6.7%
⎿	28.3%	23.9%	10.0%
⎾	6.5%	6.5%	0.0%
⋀⋀	4.3%	6.5%	0.0%
=	6.5%	6.5%	13.3%
×	45.7%	43.5%	70.0%
	Process executions		
—	26.4%	20.9%	34.0%
⎿	48.3%	51.5%	52.1%
⎾	16.7%	17.9%	11.1%
⋀⋀	8.7%	9.6%	2.8%

Best case good ⟨VM, benchmark⟩ pairings: 43.5%

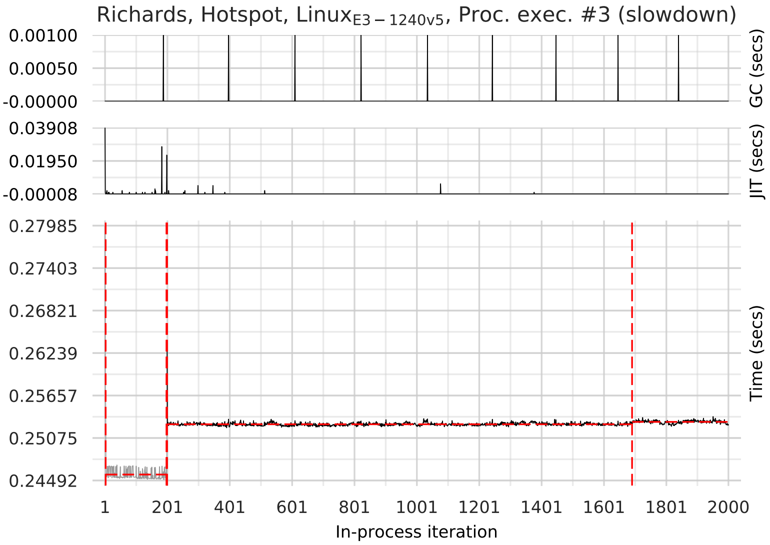
Hypothesis Invalid!

~~Hypothesis: Small, deterministic programs reach a steady state of peak performance.~~

Are the Effects due to JIT and GC?



Are the Effects due to JIT and GC?



Are the Effects due to JIT and GC?

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?

What Can We Do?

1. Run benchmarks for longer to uncover issues.

What Can We Do?

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?

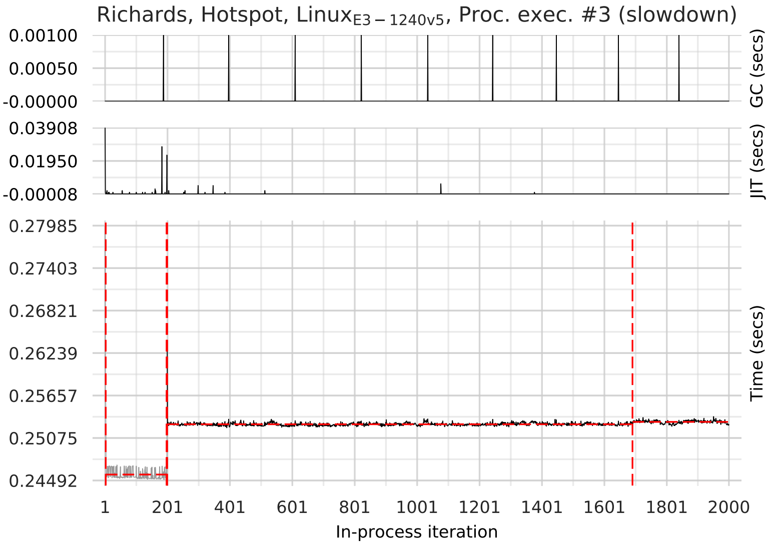
1. Run benchmarks for longer to uncover issues.
2. Accept that peak performance may not occur.
3. We can’t always blame GC or JIT compilation.
4. Always report warmup time.
5. Engineer VMs for predictable performance?

Next Version of the Paper

`https://arxiv.org/abs/1602.00602`

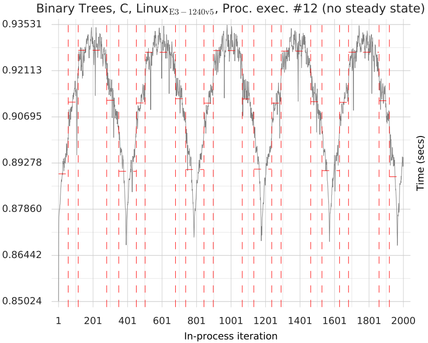
(These slides correspond to version 6 of the paper)

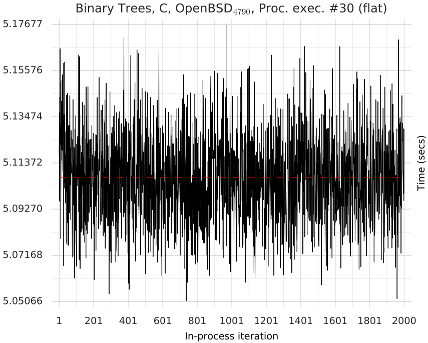
Thanks for Listening



Backup Slides

The Bouncing Ball Pattern





Method 7: Summary Statistics

For each machine, we summarise each $\langle$VM, Benchmark$\rangle$ pairing.

Method 7: Summary Statistics

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

Consistent

All 30 process executions were classified the same.

For each machine, we summarise each $\langle$VM, Benchmark$\rangle$ pairing.

Consistent

All 30 process executions were classified the same.

— ⎤⎣ ⎦⎡ ʌʌ

Inconsistent

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

- Good inconsistent: $=$ (25—, 5⎤⎣)
- Bad inconsistent: $\not\asymp$ (20—, 3⎦⎡, 7ʌʌ)

Method 7: Summary Statistics

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

Consistent

All 30 process executions were classified the same.

— ⎿ ⎾ ʌʌ

Inconsistent

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

- Good inconsistent: $=$ (25—, 5⎿)
- Bad inconsistent: $\asymp\!\!\!\!\times$ (20—, 3⎾, 7ʌʌ)

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