Automated Performance-Tracking of LLVM-Generated Code

Kristof Beyls

FOSDEM, January 2016

Why bother?

- Most of us care about Top-Of-Trunk always being in a releasable state.
  For all/majority of platforms supported.
  Or in other words – ToT always at least as good as the last release.
- Lots of different quality aspects – correctness, **speed of generated code**, size of generated code, compilation speed, …
- This talk is on how to get to a well-working continuous integration setup to monitor the speed of generated code:
  - Signalling issues quickly and reliably
  - With low false positive and low false negative rate
  - In a way that is actionable
  - Requiring as little as possible human effort
  - Enabling a culture of acting on deltas

Overview

- **Analysis of noise observed on a big.LITTLE Cortex®-A57/Cortex®-A53 system.**
- Improvements made to test-suite and LNT based on those insights.
- Other improvements made in the last year?
- Further ideas.
- Conclusions

I want to set up a low-noise perf tracking bot. What do I do?

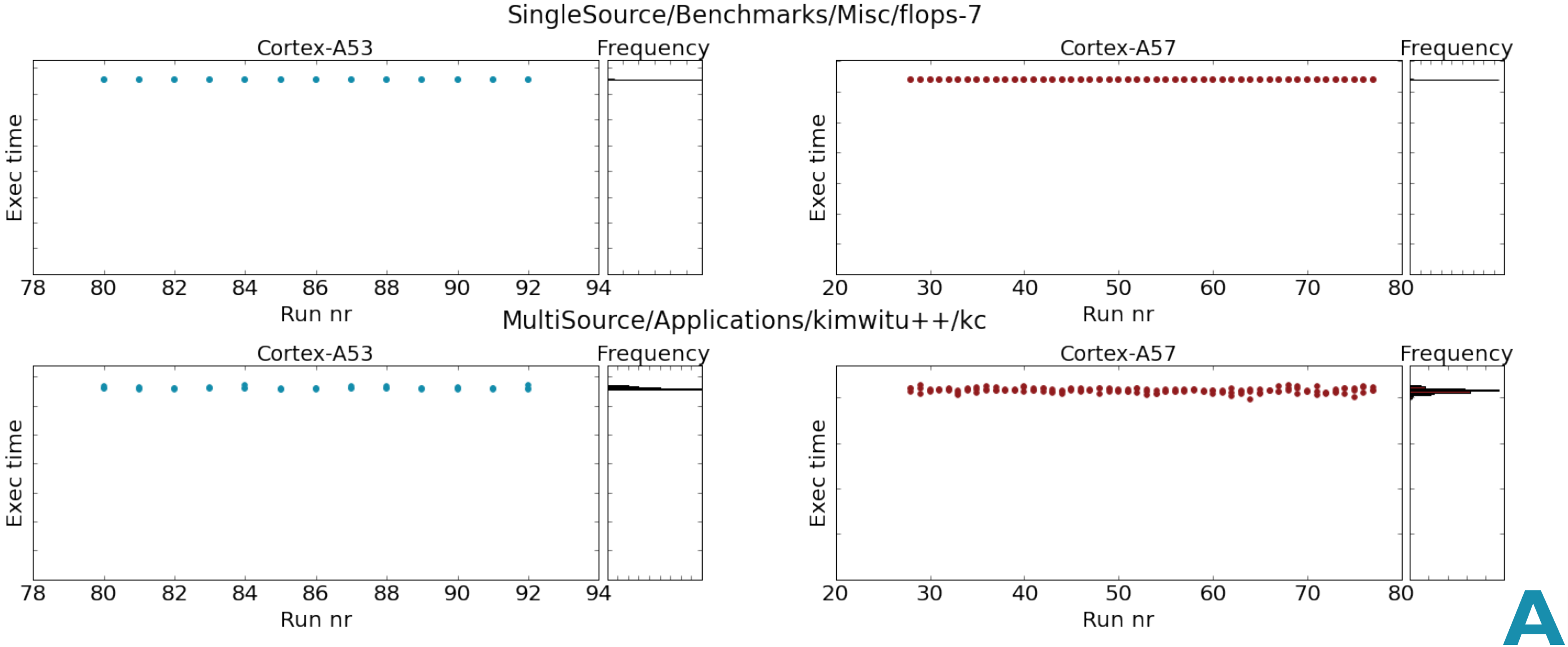
- Juno ARM development board
- 4x Cortex-A53 (in-order)
  2x Cortex-A57 (out-of-order)
  Can run both AArch64 and AArch32.

- We ***don't like*** noisy results
- We ***don't like*** late results
- We ***don't like*** false positives/negatives.
- We ***like*** actionable information.

- Not everyone has access to this platform – how can I make results more meaningful for everyone?

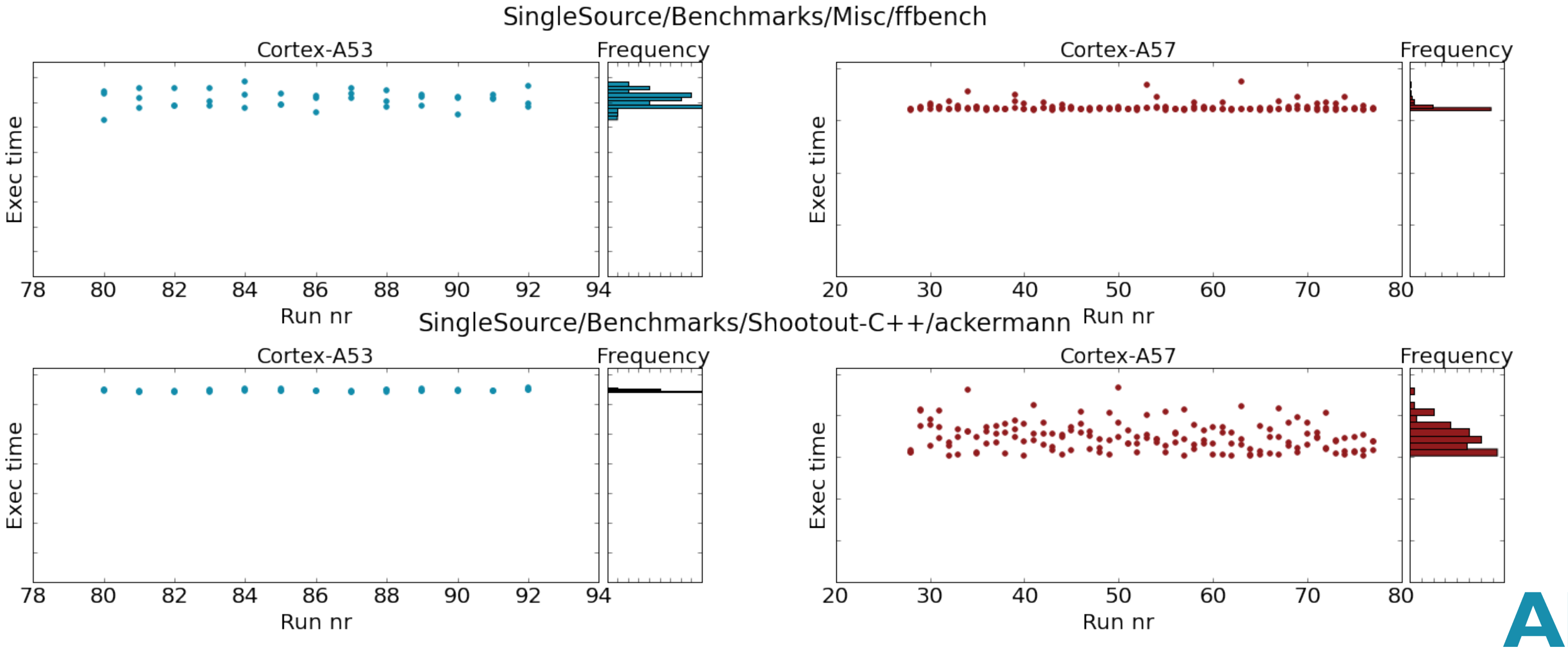
Q1: How much relative noise is there when running the same binary multiple times?

- Take the programs in the test-suite & run them a lot of times on both cores.
- Most are relatively low-noise:

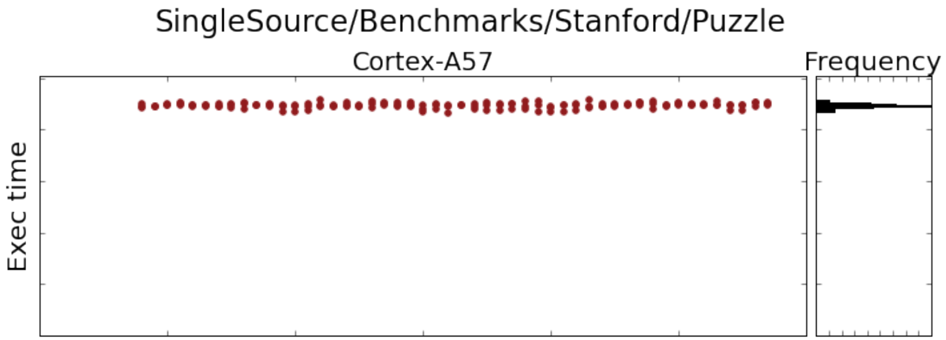


Q2. Is the noise typically consistent between cores?

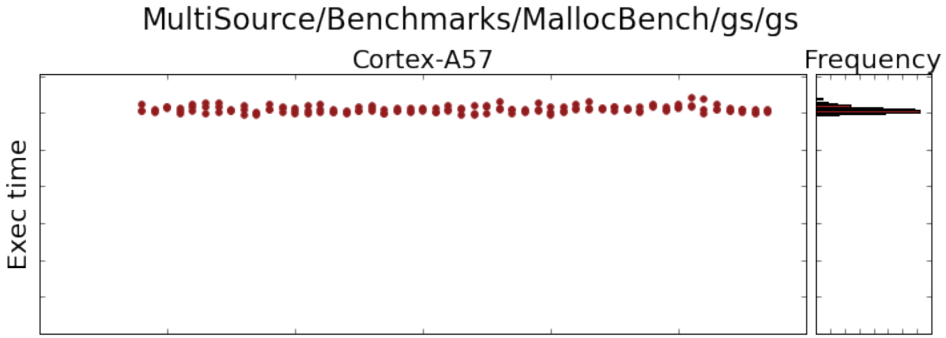
- For low-noise ones: Yes. D'uh!
- For high-noise ones: No.



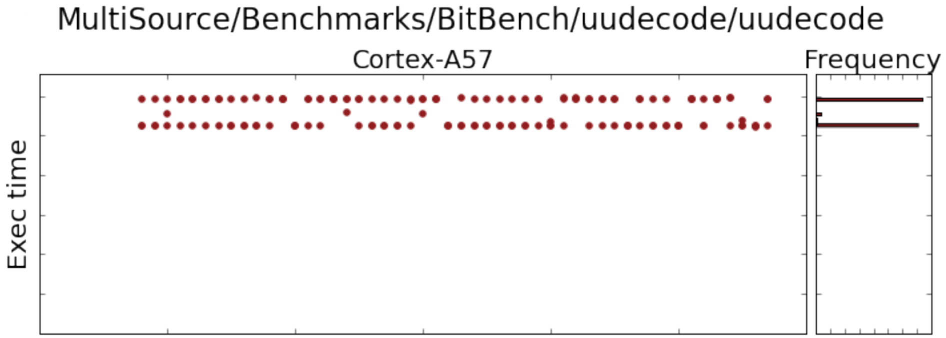
Q3. Is noise typically distributed in the same way?



Normal

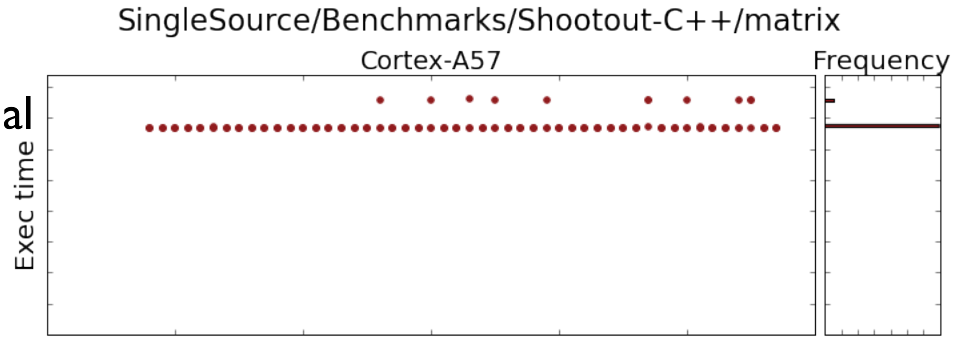


Skewed Normal

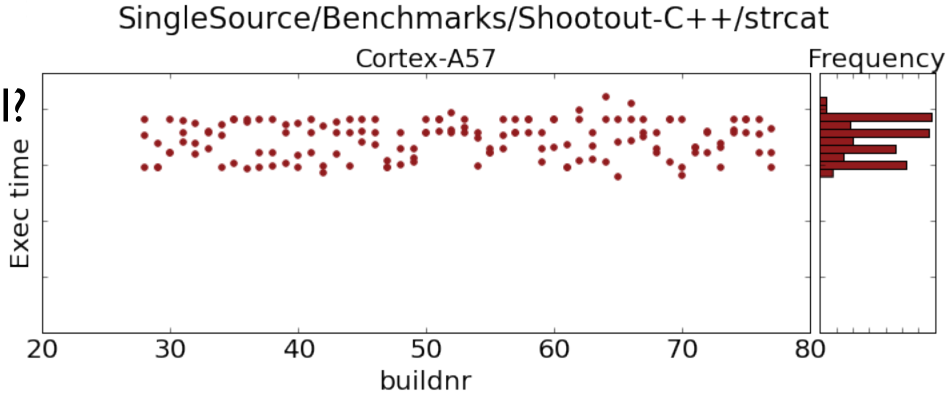


Bimodal

Skewed bimodal

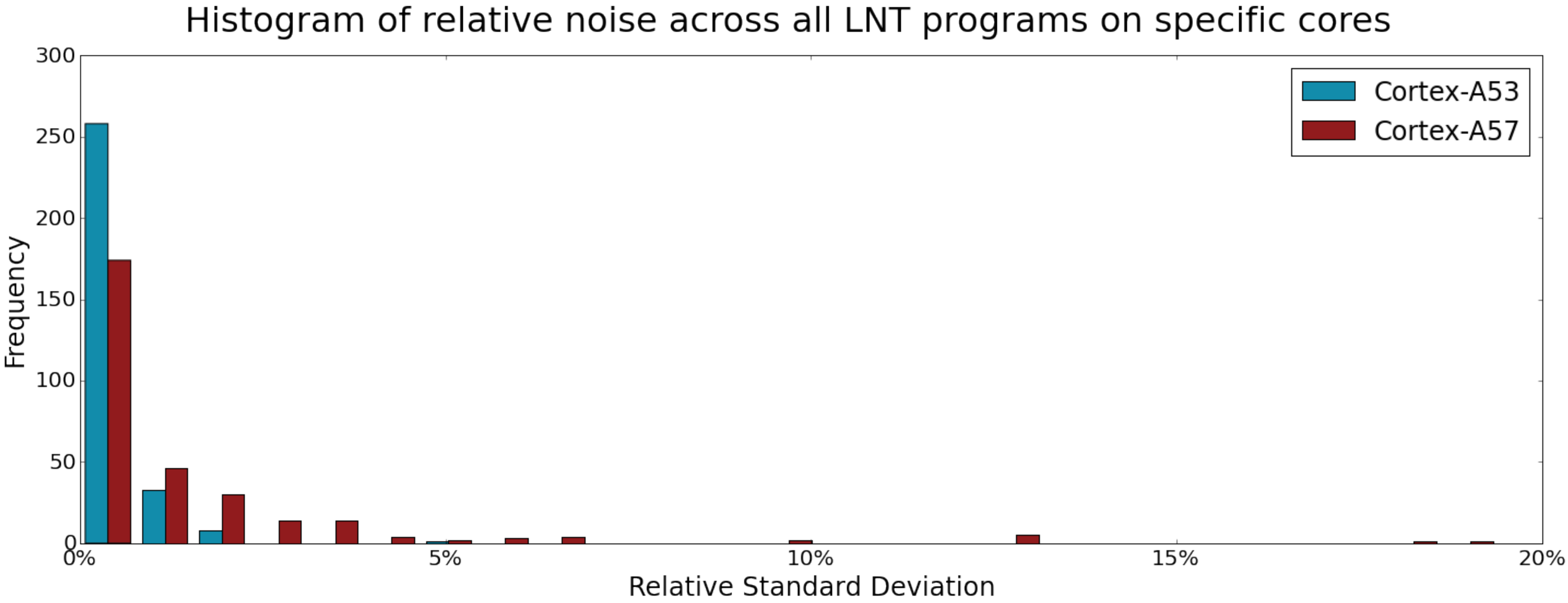


Quad-modal?



- No!

Q4. Is there a difference between both cores?



- Yes!

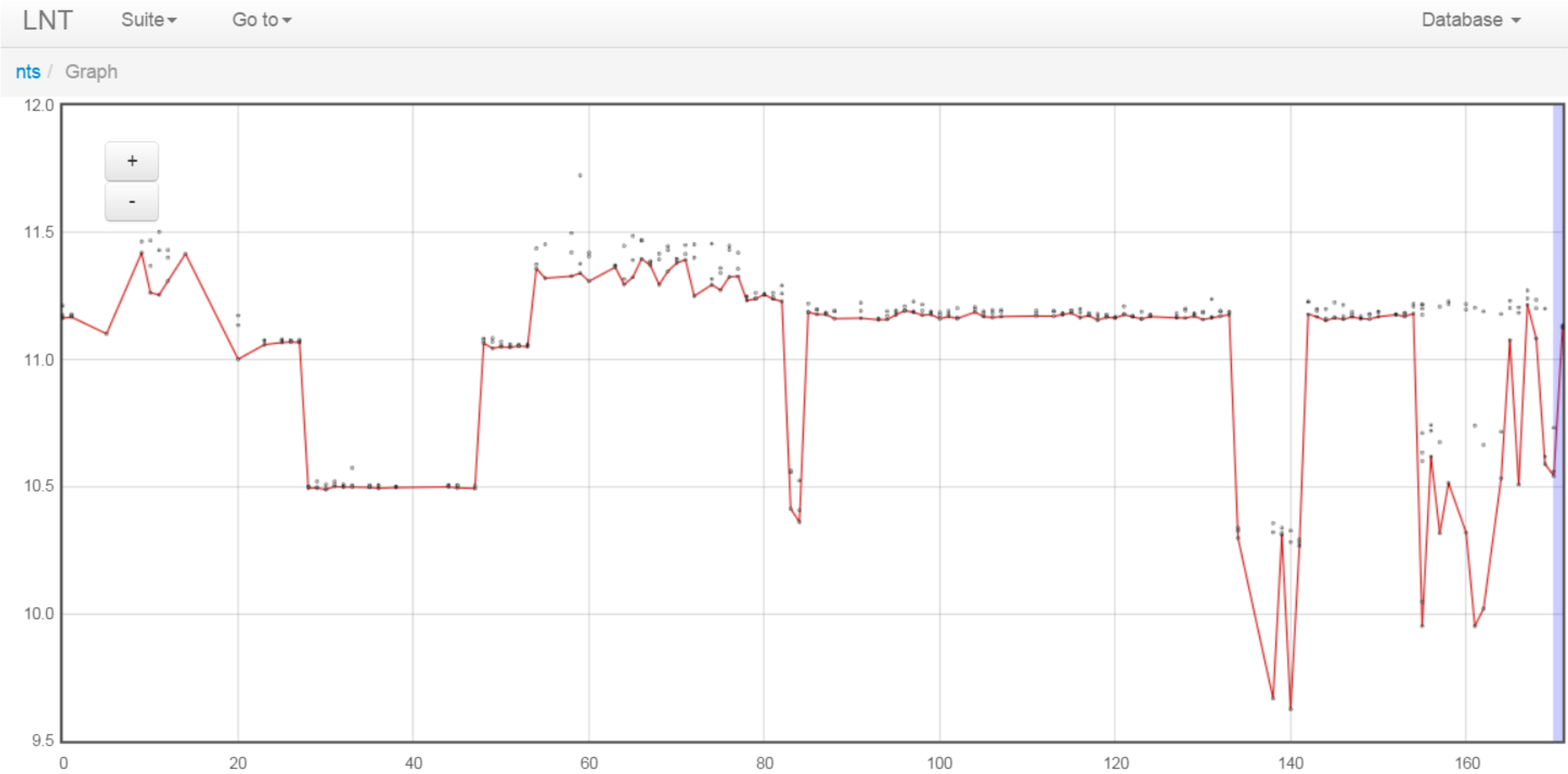
Summary of insights on the nature of noise observed

- Most programs have noise less than 1% relative standard deviation (RSD).
- 10% or more of the programs have more than 1% RSD noise.
- The noise is inherent to the nature of programs running on contemporary cores
  - Many runs of the same program shows some programs on some cores are noisy, others are not. I.e. the noise comes from a combination of address space layout randomization (ASLR) and micro-architectural effects.
  - There isn't always a single number accurately describing the performance of a program.
- Noise distribution isn't necessarily consistent across (program, core).
  We shouldn't make assumptions on distribution of noise when analyzing performance numbers.

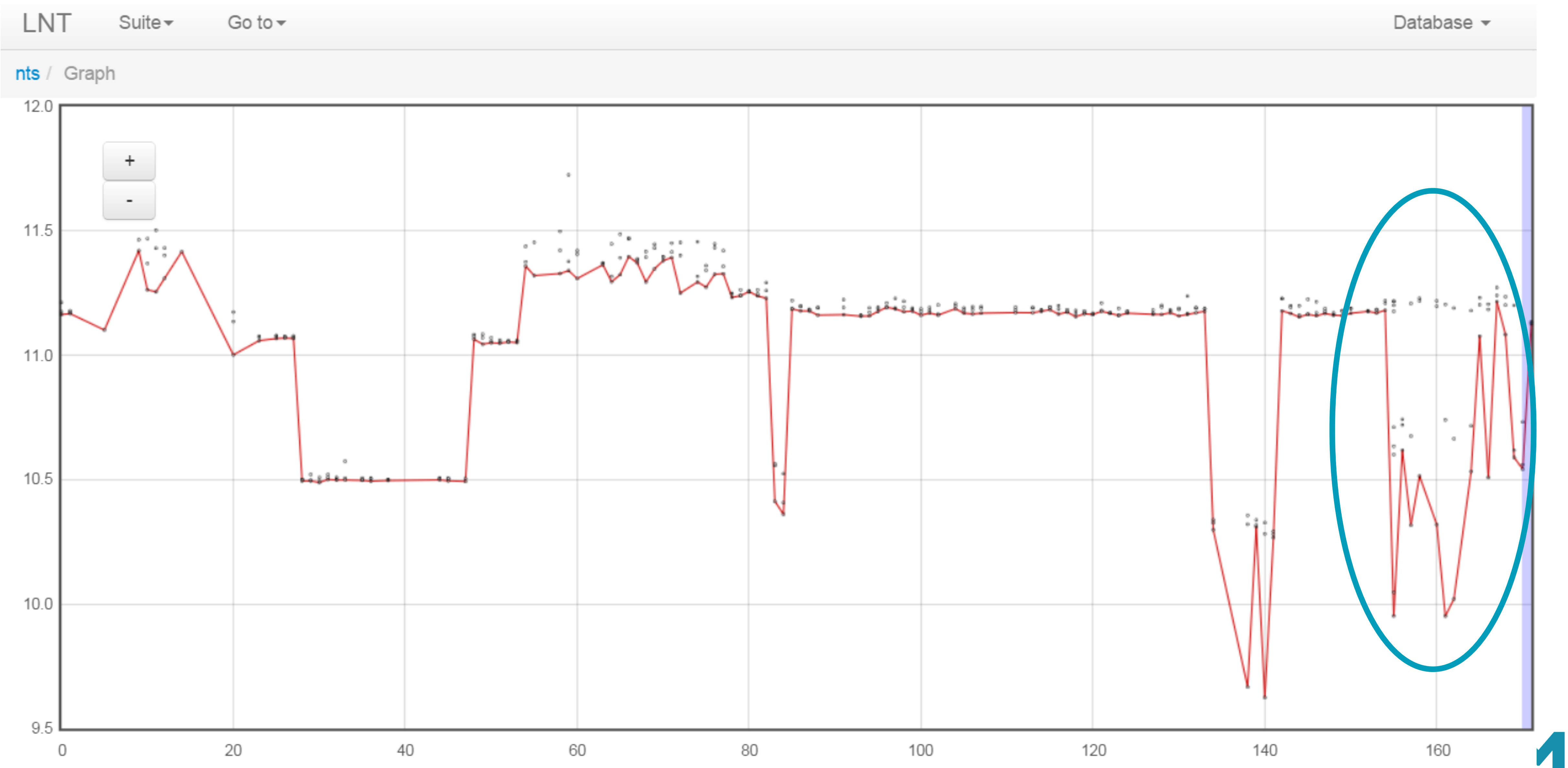
Overview

- Analysis of noise observed on a big.LITTLE Cortex®-A57/Cortex®-A53 system.
- **Improvements made to test-suite and LNT based on those insights.**
- Other improvements made in the last year?
- Further ideas.
- Conclusions

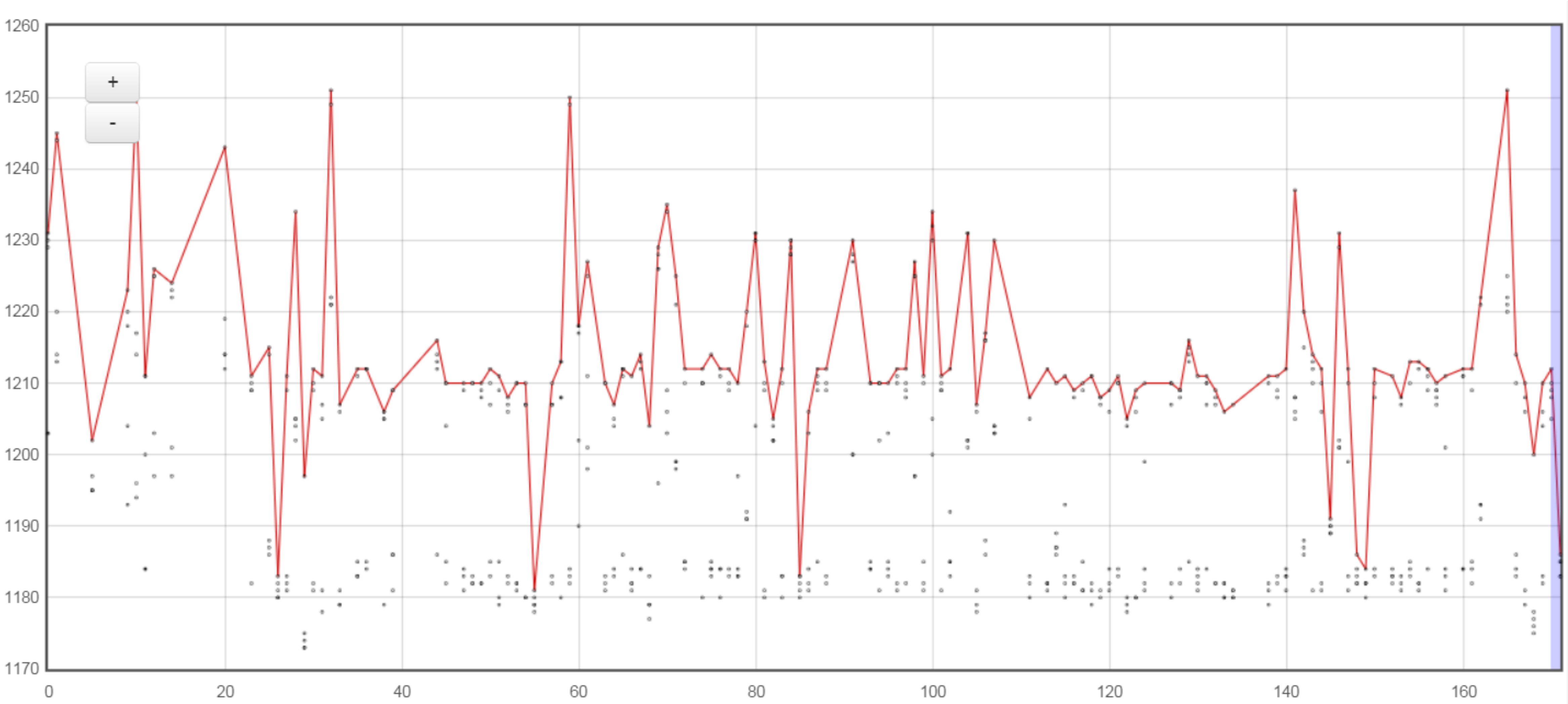
i1. Show multiple sample points by default.



i1. Is “min”/”max” the right aggregation function?

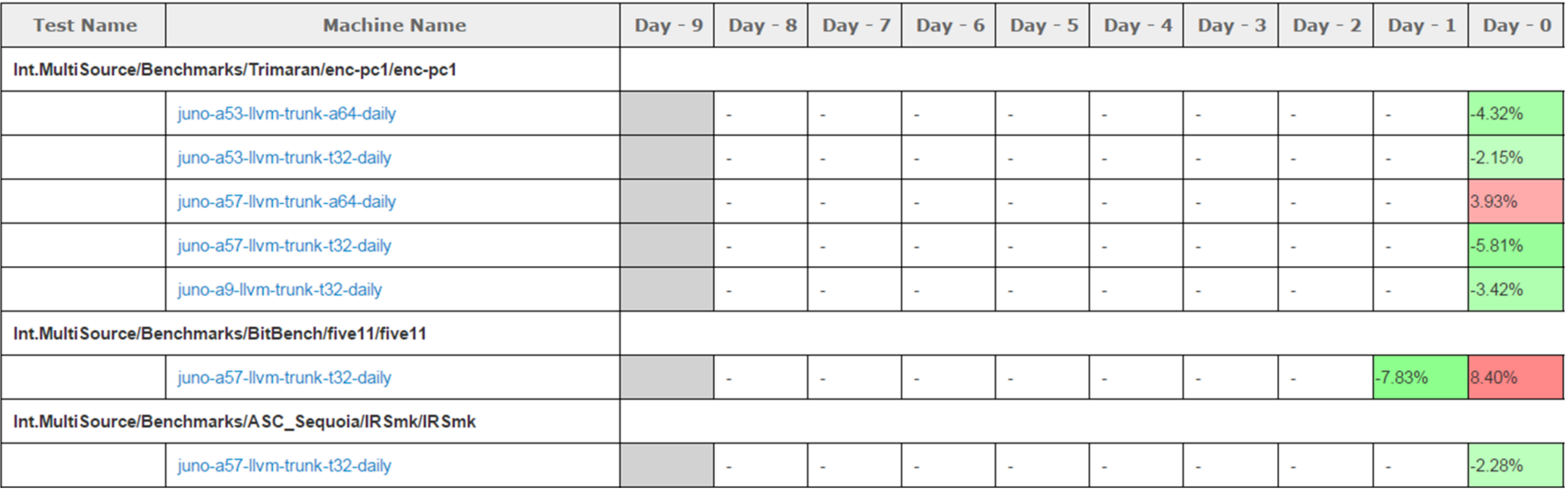


i1. Is “min”/”max” the right aggregation function?



i2. Sparklines on daily report page

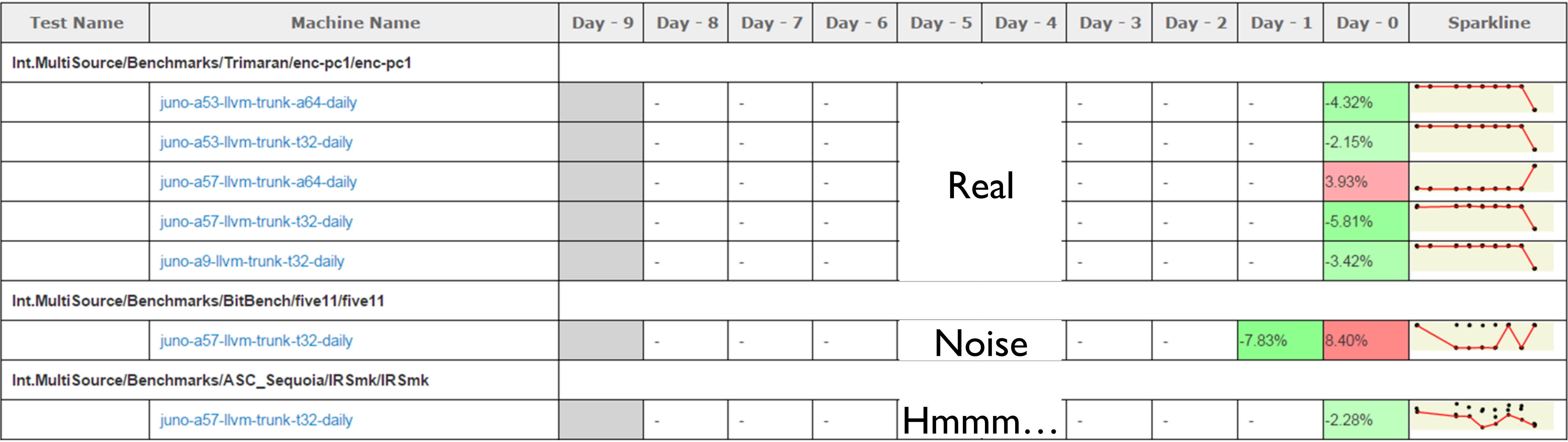
- Which performance deltas are real, which ones are noise?



Test Name	Machine Name	Day - 9	Day - 8	Day - 7	Day - 6	Day - 5	Day - 4	Day - 3	Day - 2	Day - 1	Day - 0
Int.MultiSource/Benchmarks/Trimaran/enc-pc1/enc-pc1											
	juno-a53-llvm-trunk-a64-daily		-	-	-	-	-	-	-	-	-4.32%
	juno-a53-llvm-trunk-t32-daily		-	-	-	-	-	-	-	-	-2.15%
	juno-a57-llvm-trunk-a64-daily		-	-	-	-	-	-	-	-	3.93%
	juno-a57-llvm-trunk-t32-daily		-	-	-	-	-	-	-	-	-5.81%
	juno-a9-llvm-trunk-t32-daily		-	-	-	-	-	-	-	-	-3.42%
Int.MultiSource/Benchmarks/BitBench/five11/five11											
	juno-a57-llvm-trunk-t32-daily		-	-	-	-	-	-	-	-7.83%	8.40%
Int.MultiSource/Benchmarks/ASC_Sequoia/IRSmk/IRSmk											
	juno-a57-llvm-trunk-t32-daily		-	-	-	-	-	-	-	-	-2.28%

i2. Sparklines on daily report page

- Which performance deltas are real, which ones are noise?



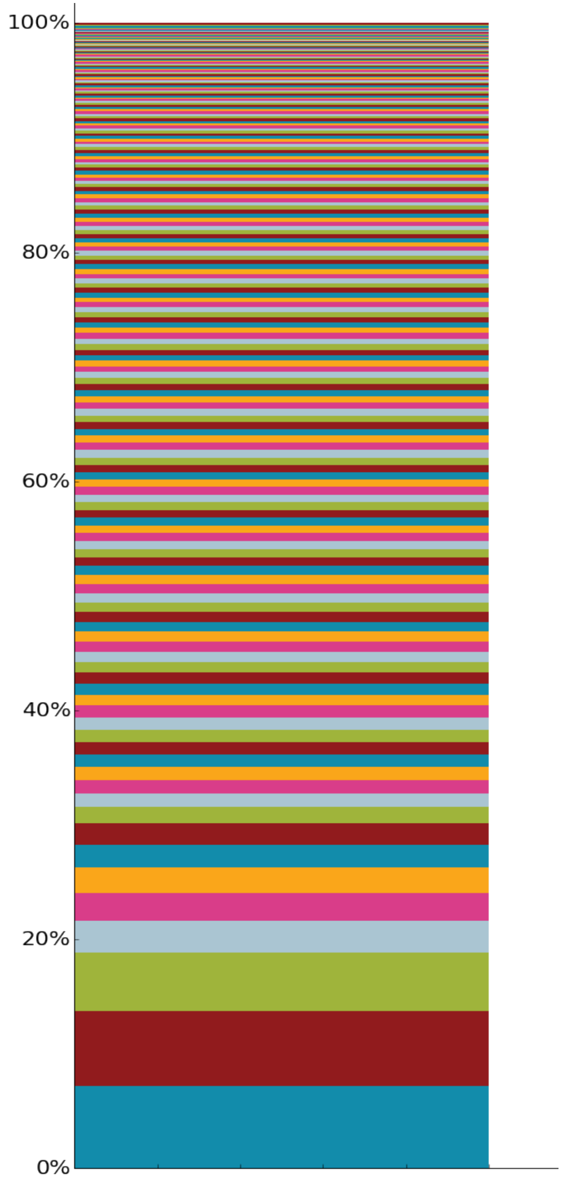
Test Name	Machine Name	Day - 9	Day - 8	Day - 7	Day - 6	Day - 5	Day - 4	Day - 3	Day - 2	Day - 1	Day - 0	Sparkline
Int.MultiSource/Benchmarks/Trimaran/enc-pc1/enc-pc1												
	juno-a53-llvm-trunk-a64-daily		-	-	-	Real		-	-	-	-4.32%	
	juno-a53-llvm-trunk-t32-daily		-	-	-			-	-	-	-2.15%	
	juno-a57-llvm-trunk-a64-daily		-	-	-			-	-	-	3.93%	
	juno-a57-llvm-trunk-t32-daily		-	-	-			-	-	-	-5.81%	
	juno-a9-llvm-trunk-t32-daily		-	-	-			-	-	-	-3.42%	
Int.MultiSource/Benchmarks/BitBench/five11/five11												
	juno-a57-llvm-trunk-t32-daily		-	-	-	Noise		-	-	-7.83%	8.40%	
Int.MultiSource/Benchmarks/ASC_Sequoia/IRSmk/IRSmk												
	juno-a57-llvm-trunk-t32-daily		-	-	-	Hmmm…		-	-	-	-2.28%	

i3. Remove very short-running programs (< 10ms) in benchmark mode?

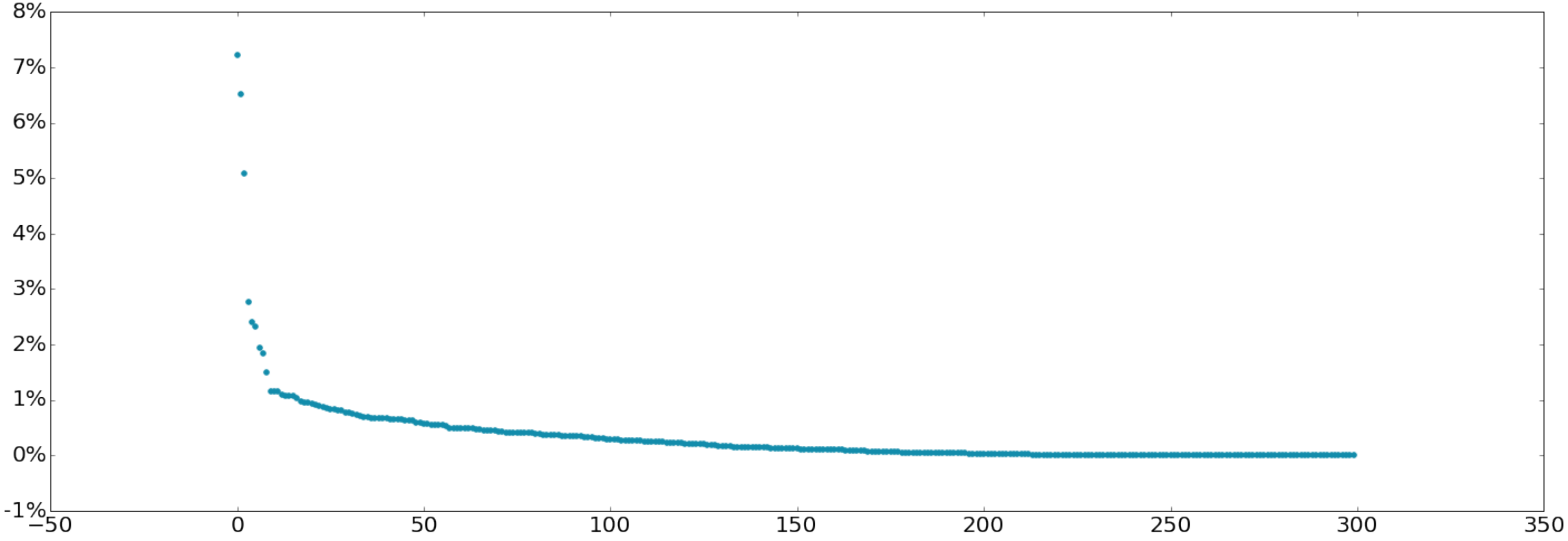
- Out of the 300 programs in the test-suite; 20-ish run for less than 10ms. Do they do enough work for the hardware to have a chance to produce low-noise data?

6 programs not having loops at all REMOVED	10 programs which do very little work REMOVED	3 programs where code seems optimized away completely KEPT
SingleSource/UnitTests/Vector/constpool SingleSource/UnitTests/Vector/simple SingleSource/UnitTests/Vector/AArch64/aarch64_neon_intrinsics SingleSource/UnitTests/2005-07-15-Bitfield-ABI SingleSource/UnitTests/2006-01-23-UnionInit SingleSource/UnitTests/2007-04-10-BitfieldTest	MultiSource/Benchmarks/Prolangs-C/lloader MultiSource/Benchmarks/McCat/15-trie MultiSource/Benchmarks/Prolangs-C/cdecl MultiSource/Benchmarks/MiBench/office-stringsearch MultiSource/Benchmarks/MiBench/telecom-adpcm SingleSource/Benchmarks/Stanford/IntMM SingleSource/Regression/C/matrixTranspose SingleSource/Regression/C/sumarray2d SingleSource/Regression/C/test_indvars: SingleSource/UnitTests/SignlessTypes:	SingleSource/Benchmarks/Misc/lowercase SingleSource/Benchmarks/Shootout/objinst SingleSource/Benchmarks/Shootout-C++/objinst

i4. Can the test-suite produce useful benchmark results faster?



- 27 out of 300 programs cover 50% of total run-time.
- Many of those are in Polybench sub-suite. They spent all their time printf-ing a large matrix.
  Renato fixed that. Results in 5% faster test-suite, less noise.



i5. Compile time is expected to be noisy when using all cores on a heterogenous big.LITTLE board

- In a fully parallel build,
  some compile jobs will land on the big&fast core,
  some compile jobs will land on the little&slower core.
- –exclude-stat-from-submission.
  To avoid submitting compile time numbers on our big.LITTLE board.
  Also should be used for other systems where one kind of metric just is unstable.

i6. Making it easier to develop LNT

- Make it easier to create regression tests for new functionality:
  - Transformed database regression tests to create DB from SQL statements rather than binary dump. Which in itself makes adding regression tests for new DB-based functionality straightforward.
  - Made checking of webui output in regression test possible.
  - Made running regression tests possible against both sqlite and postgres.
- Created an initial developer's guide

- The combination of the above raises LNT development practices to roughly the same level as other LLVM sub-projects.
  There are still many missing tests for existing functionality; but it shouldn't be too hard to add them bit by bit now.

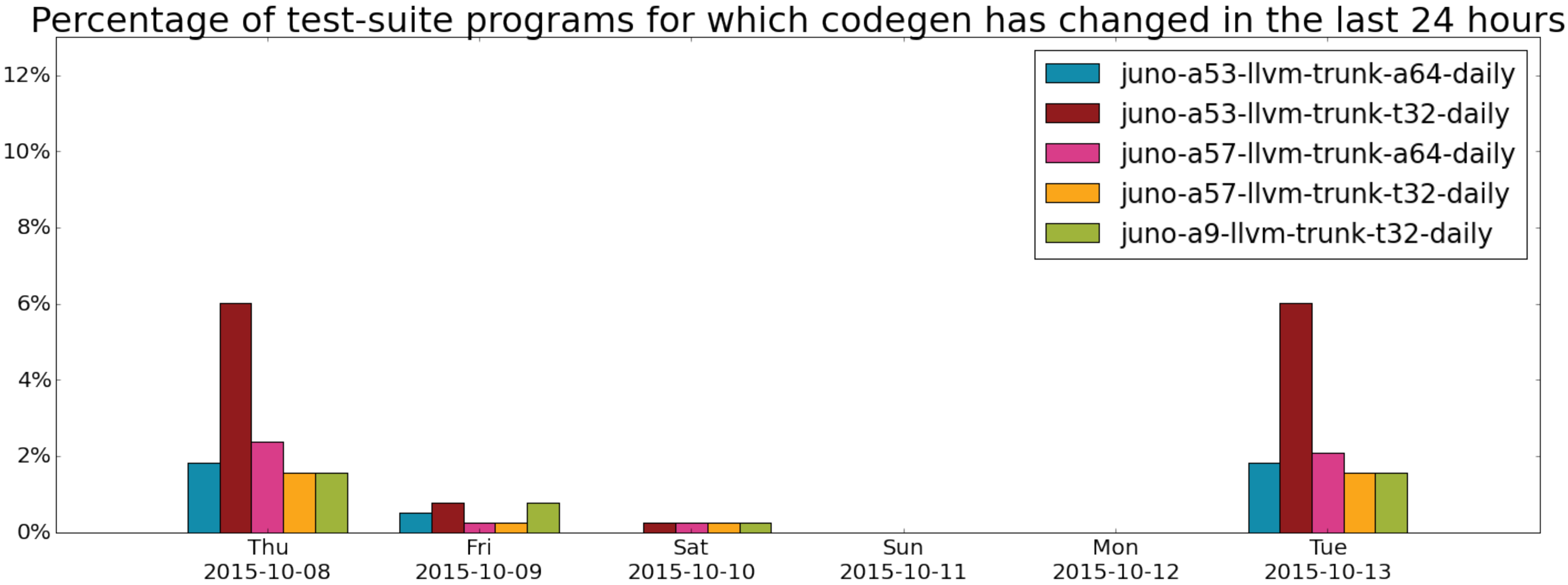
i7. Summary of improvements made based on analysis

- LNT
  - Show all sample points by default.
    Indicating min/max may not be the best aggregation function.
  - Sparklines – with all sample points – on daily report page.
  - –exclude-stat-from-submission.
    Allowing to not submit metrics that are known to be noisy.
- test-suite
  - Remove very short-running programs from benchmark mode
  - Renato fixed most polybench benchmarks spending all their time in printf.
- llvm-juno-lnt-perf__LNT-AArch64-A53-O3__clang_DEV__aarch64:39
  - make use Cortex-A53 rather than Cortex-A57.
  - Keep ASLR enabled.

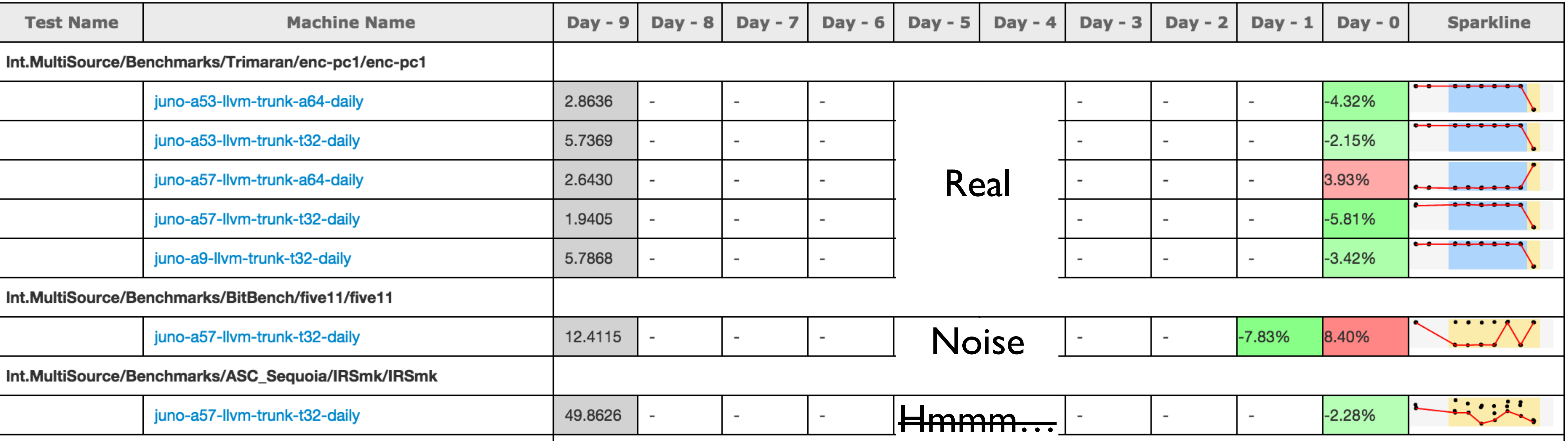
Overview

- **Analysis of noise observed on a big.LITTLE Cortex®-A57/Cortex®-A53 system.**
- Improvements made to test-suite and LNT based on those insights.
- **Other improvements made in the last year?**
- Further ideas.
- Conclusions

01. Recording hash of generated binary



01. Recording hash of generated binary



Test Name	Machine Name	Day - 9	Day - 8	Day - 7	Day - 6	Day - 5	Day - 4	Day - 3	Day - 2	Day - 1	Day - 0	Sparkline
Int.MultiSource/Benchmarks/Trimaran/enc-pc1/enc-pc1												
	juno-a53-llvm-trunk-a64-daily	2.8636	-	-	-			-	-	-	-4.32%	
	juno-a53-llvm-trunk-t32-daily	5.7369	-	-	-			-	-	-	-2.15%	
	juno-a57-llvm-trunk-a64-daily	2.6430	-	-	-			-	-	-	3.93%	
	juno-a57-llvm-trunk-t32-daily	1.9405	-	-	-			-	-	-	-5.81%	
	juno-a9-llvm-trunk-t32-daily	5.7868	-	-	-			-	-	-	-3.42%	
Int.MultiSource/Benchmarks/BitBench/five11/five11												
	juno-a57-llvm-trunk-t32-daily	12.4115	-	-	-			-	-	-7.83%	8.40%	
Int.MultiSource/Benchmarks/ASC_Sequoia/IRSmk/IRSmk												
	juno-a57-llvm-trunk-t32-daily	49.8626	-	-	-			-	-	-	-2.28%	

Real

Noise

~~Hmmm…~~

Noise!

O2. A few other major improvements

- Improving signal-to-noise and actionability (by Chris Matthews):
  - Better analysis algorithm to detect regressions – working, probably can be improved further.
  - Performance change tracking ui & db – in development. Goal is to make the data LNT produces more actionable.
  - Llvm-bisect tool – stores clang binaries built by bots in a cache. Scripts can fetch these builds to more quickly bisect issues.
- New metrics
  - score, mem_bytes. bigger-is-better
- Stability fixes to the server llvm.org/perf
  - REST and Ajax interface; offline computation in the webui; general bug fixes.
- Various ui polishings

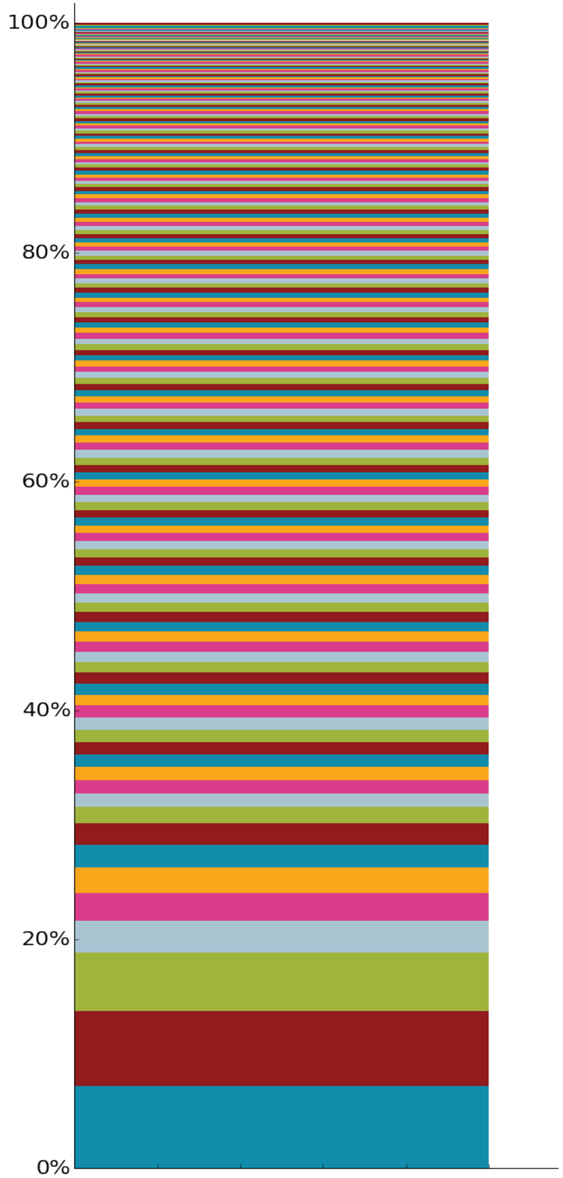
Overview

- **Analysis of noise observed on a big.LITTLE Cortex®-A57/Cortex®-A53 system.**
- Improvements made to test-suite and LNT based on those insights.
- Other improvements made in the last year?
- **Further ideas.**
- Conclusions

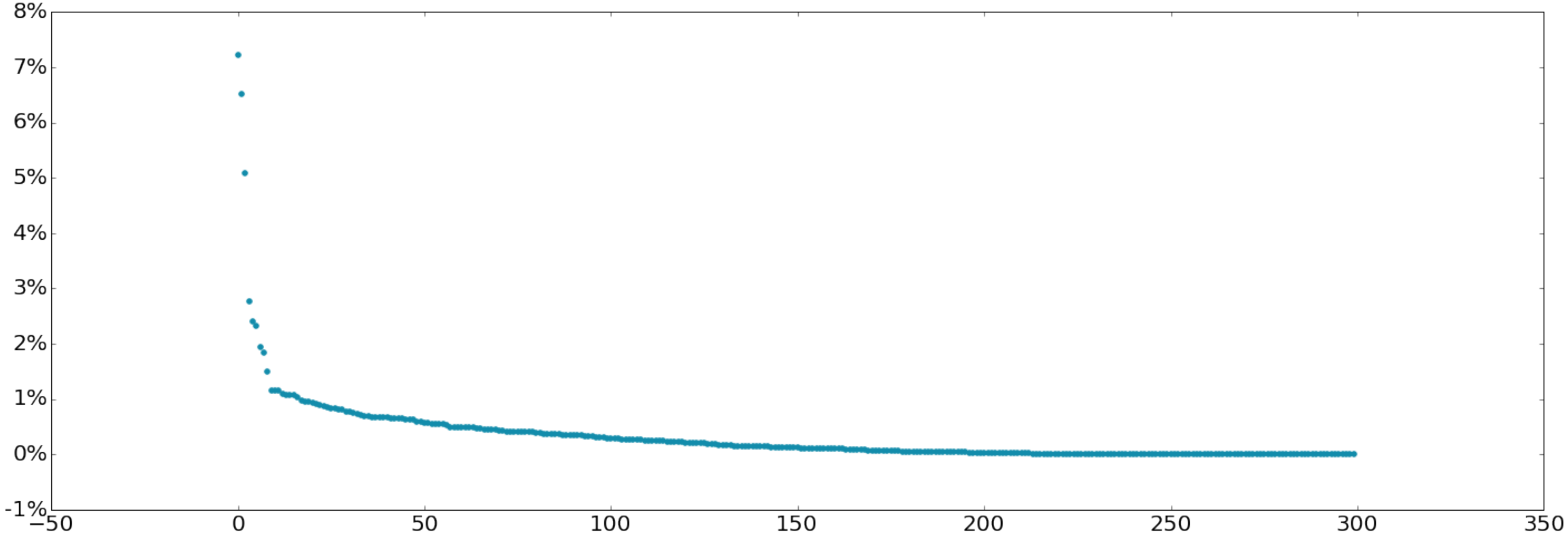
F1. What is the goal of running the test-suite as a benchmark?

- Results can be publicly shared – for many commercial benchmarks, T&C don't allow that.
- Commercial benchmarks sometimes run for a long time; we want quick feedback.
  - Should the test-suite in benchmarking mode be a set of micro-benchmark-ish-things?
    See Chandler's cppcon2015 presentation
- Is the test-suite representative enough of the "real world"?
  - Not sure how to measure this well…

F1. Can the test-suite produce useful benchmark results faster?



- Total runtime on Cortex-A53: 5769.33s
- If we'd adapt the programs to run more quickly:
  at most 100 ms = 26.94s (speedup: 214x)
  at most 1 s = 232.02s (speedup: 24x)



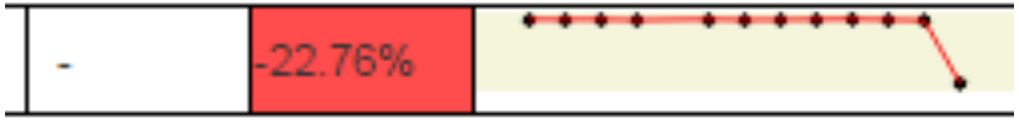
F1. Public/community performance tracking vs in-house tracking

- No-one(?) has access to all the platforms LLVM supports.
  - Does the test-suite provide good enough data on performance on a platform you don't have access to, but for which public performance tracking bots give you feedback?
  - For correctness testing, we have quite a few different public bots on different platforms.
  - For performance tracking we only have few so far.

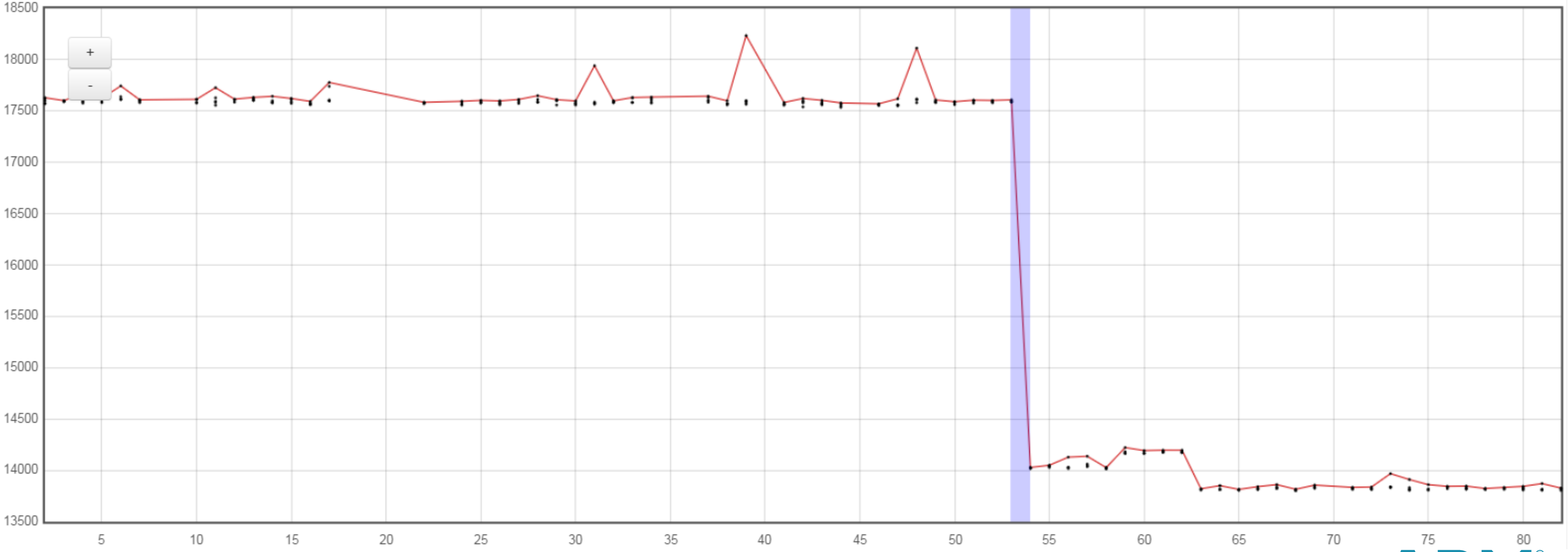
- Is the test-suite representative enough?
  - For what kind of programs/areas/segments?

- Continuous deployment of ToT LNT/test-suite?
  - Some public buildbots use ToT LNT.
  - But the server at llvm.org/perf isn't auto-updated.

F2. Less effort to go from perf delta to understanding what caused it



-22.76%



F2. OK – 20% regression. What caused it?

- Which commit?
  - Could we integrate some kind of bisecting service on perf-tracking builders?
  - Can it be built on top of the bisecting script and cache available now?
  - Building on top of rerun functionality in LNT; if needed using cross-built binaries for slow perf tracking bots?

```
r248018 | conghou | 2015-09-18 19:19:40 +0100 (Fri, 18 Sep 2015) | 7 lines
Scaling up values in ARMBaseInstrInfo::isProfitableToIfCvt() before they are
  scaled by a probability to avoid precision issue.
```

- Exactly what kind of code change caused the delta?
  - Could we store performance traces on the side, and get LNT to do some kind of analysis to highlight the “hot” differences?
  - Without needing access to the hardware where the performance change was seen?

F2. Show annotated assembly diffs – e.g linux perf output

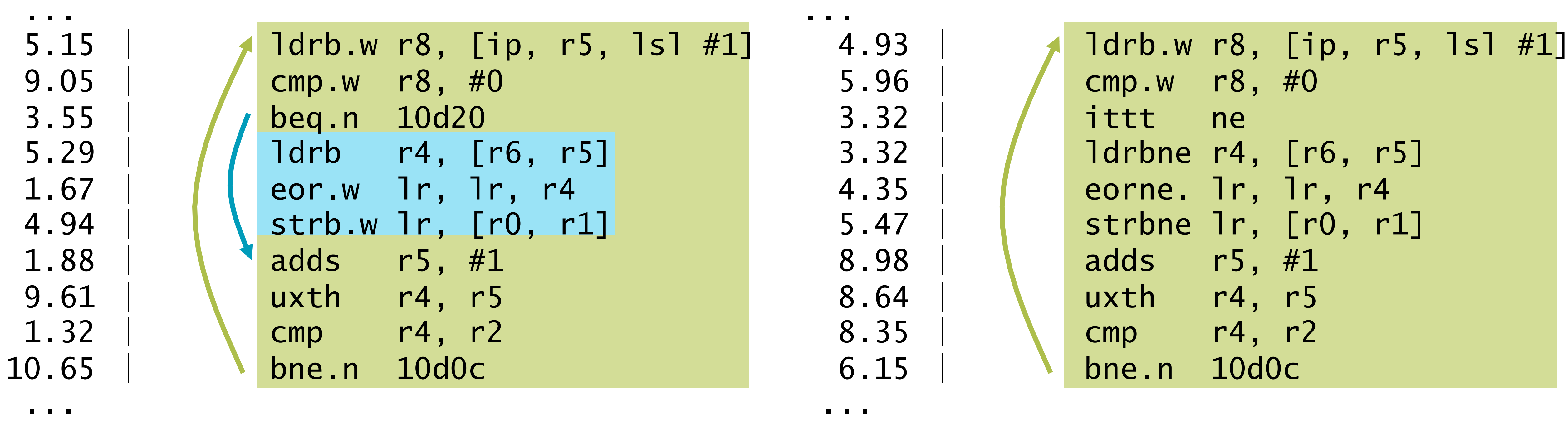
b53 (r247972)

```
  ...
  5.15 |         ldrb.w r8, [ip, r5, lsl #1]
  9.05 |         cmp.w  r8, #0
  3.55 |         beq.n  10d20
  5.29 |         ldrb   r4, [r6, r5]
  1.67 |         eor.w  lr, lr, r4
  4.94 |         strb.w lr, [r0, r1]
  1.88 |         adds   r5, #1
  9.61 |         uxth   r4, r5
  1.32 |         cmp    r4, r2
 10.65 |         bne.n  10d0c
  ...
```

b54 (r248094)

```
...
 4.93 |         ldrb.w r8, [ip, r5, lsl #1]
 5.96 |         cmp.w  r8, #0
 3.32 |         ittt   ne
 3.32 |         ldrbne r4, [r6, r5]
 4.35 |         eorne. lr, lr, r4
 5.47 |         strbne lr, [r0, r1]
 8.98 |         adds   r5, #1
 8.64 |         uxth   r4, r5
 8.35 |         cmp    r4, r2
 6.15 |         bne.n  10d0c
 ...
```

F2. Show annotated assembly diffs – e.g linux perf output



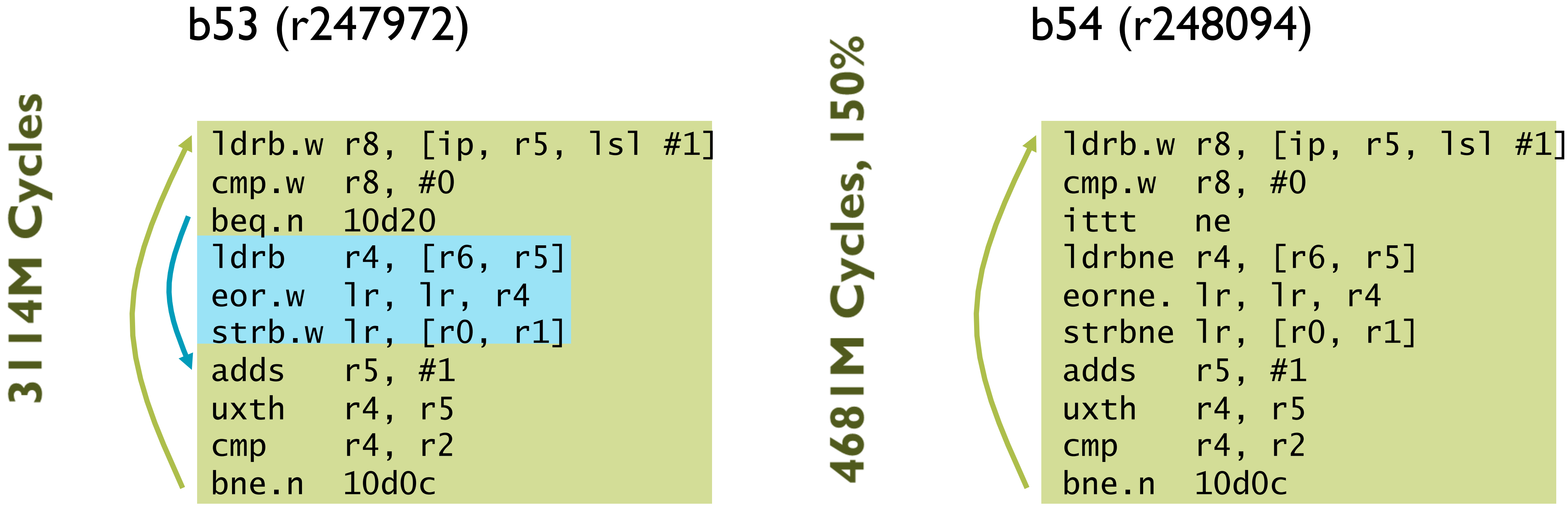
b53 (r247972)

```
  ...
  5.15 |      ldrb.w r8, [ip, r5, lsl #1]
  9.05 |      cmp.w  r8, #0
  3.55 |      beq.n  10d20
  5.29 |      ldrb   r4, [r6, r5]
  1.67 |      eor.w  lr, lr, r4
  4.94 |      strb.w lr, [r0, r1]
  1.88 |      adds   r5, #1
  9.61 |      uxth   r4, r5
  1.32 |      cmp    r4, r2
 10.65 |      bne.n  10d0c
  ...
```

b54 (r248094)

```
 ...
  4.93 |      ldrb.w r8, [ip, r5, lsl #1]
  5.96 |      cmp.w  r8, #0
  3.32 |      ittt   ne
  3.32 |      ldrbne r4, [r6, r5]
  4.35 |      eorne. lr, lr, r4
  5.47 |      strbne lr, [r0, r1]
  8.98 |      adds   r5, #1
  8.64 |      uxth   r4, r5
  8.35 |      cmp    r4, r2
  6.15 |      bne.n  10d0c
  ...
```

F2. Show annotated assembly diffs – e.g linux perf output



b53 (r247972)

3114M Cycles

```
ldrb.w r8, [ip, r5, lsl #1]
cmp.w  r8, #0
beq.n  10d20
ldrb   r4, [r6, r5]
eor.w  lr, lr, r4
strb.w lr, [r0, r1]
adds   r5, #1
uxth   r4, r5
cmp    r4, r2
bne.n  10d0c
```

4681M Cycles, 150%

b54 (r248094)

```
ldrb.w r8, [ip, r5, lsl #1]
cmp.w  r8, #0
ittt   ne
ldrbne r4, [r6, r5]
eorne. lr, lr, r4
strbne lr, [r0, r1]
adds   r5, #1
uxth   r4, r5
cmp    r4, r2
bne.n  10d0c
```

Danger of re-inventing performance analysis tools in LNT's web-ui?

F3. Driving the test-suite using cmake & lit

- Instead of the current Makefiles.
- Main goal is to make it easy to plug in other benchmark suites under "Externals"
  - … and have all extra features to build and run the test-suite work automatically for every benchmark suite under externals too. Examples:
  - Recording hash of binary
  - Running the program under perf or other profiling tools
  - Measuring code size
  - … any other improvement to measuring program build or execution we can think of in the future.
- Work in progress

Overview

- Analysis of noise observed on a big.LITTLE Cortex®-A57/Cortex®-A53 system.
- Improvements made to test-suite and LNT based on those insights.
- Other improvements made in the last year?
- Further ideas.
- **Conclusions**

Conclusion

- Some really good progress this year:
  - Signalling issues quickly and reliably ↑
  - With low false positive and low false negative rate ↑
  - In a way that is actionable ↗
  - Requiring as little as possible human effort ↑
  - Enabling a culture of acting on deltas →
- Consider using LNT as your performance tracking infrastructure for down-stream changes too. It’s not perfect yet, but amongst the best available.