Towards Ameliorating Measurement Bias in Evaluating Performance of Generated Code

ARM

Kristof Beyls

EuroLLVM
18/3/2016

Intro

- Evaluating performance of generated code
  - As a pre-commit test for a new optimization patch.
  - As post-commit performance tracking.
- Is the patch/commit OK?
- Measurements often give conflicting and sometimes misleading answers.

- Even when the benchmarking system is setup well to avoid CPU-external noise:
  - Programs pinned to a specific core.
  - Turn off daemon processes/OS services.
  - Make sure CPU frequency scaling doesn’t happen.

ARM

Codegen Change: Often Unclear if Good or Bad.

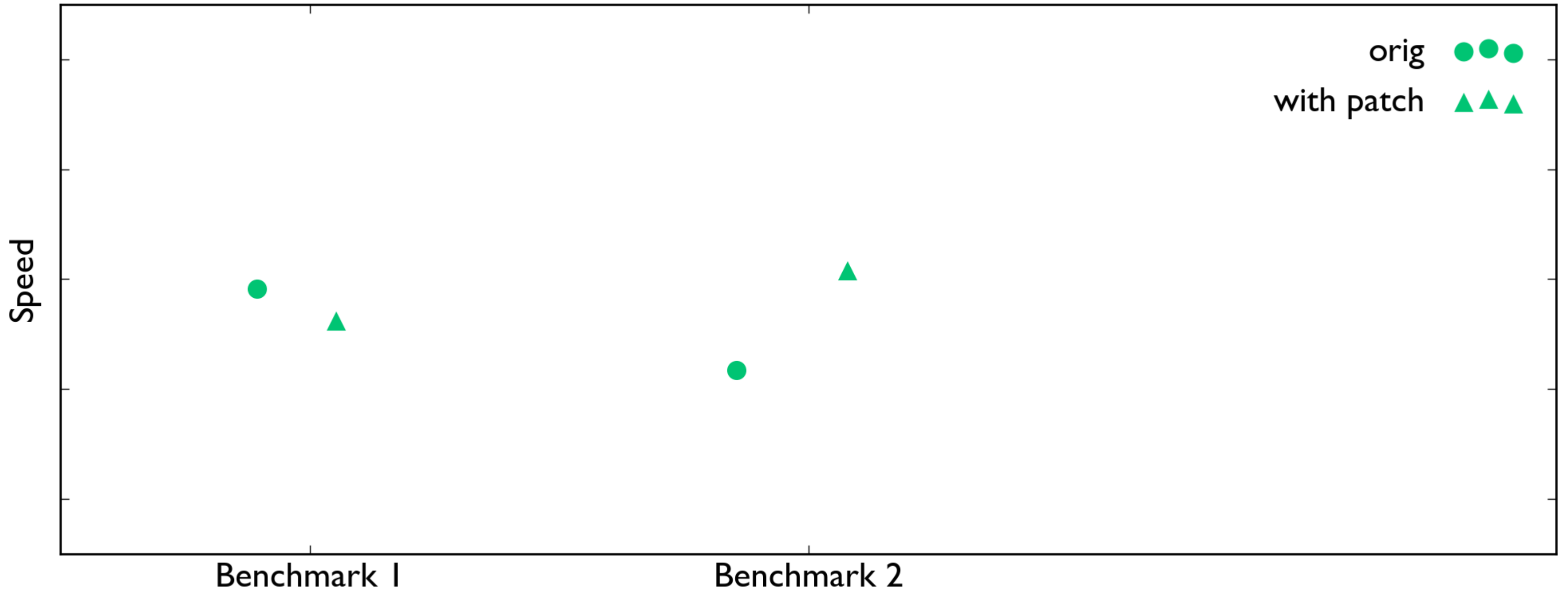
- **Which benchmarks matter?**
  Change is often good for one benchmark, bad for the other.

- **Which micro-architectures?**
  Change can be good for one micro-architecture, bad for the other.

- **Noisy system:**
  Same program running at different speeds when executed multiple times

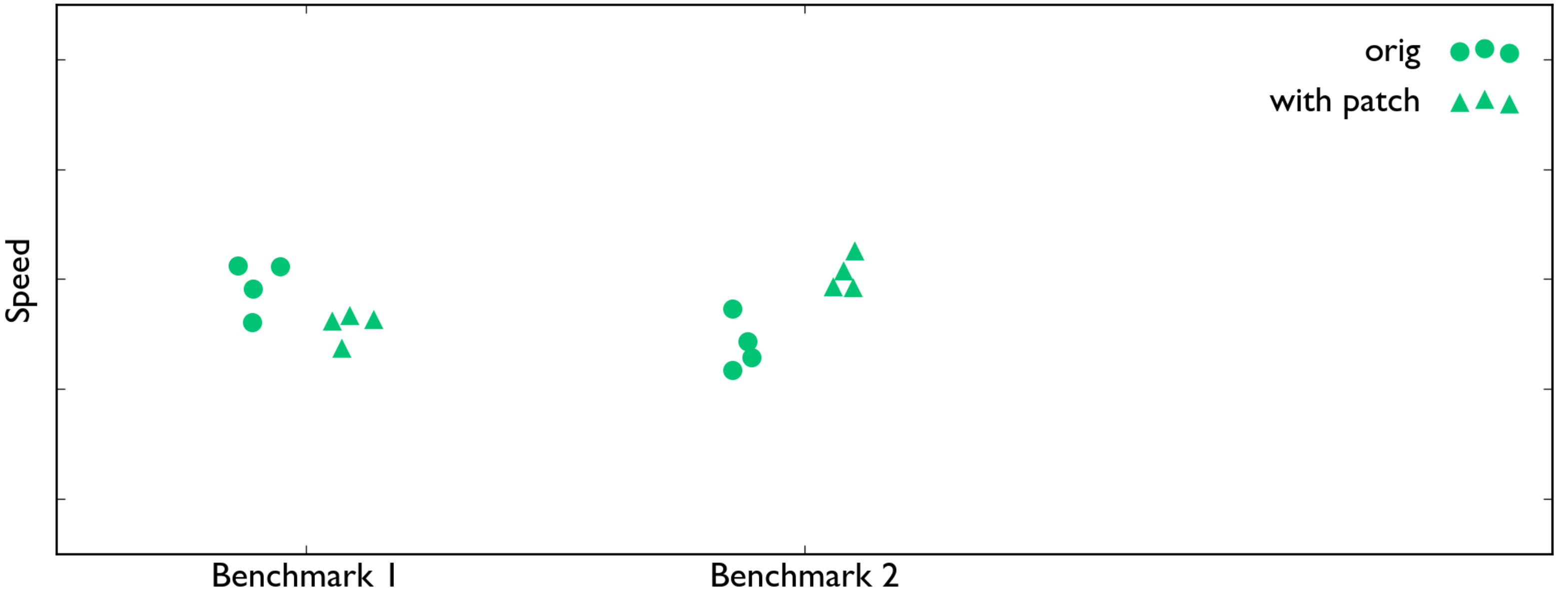
- **Chaotic system:**
  Small program change causes non-linear effect on execution speed.

Multiple Benchmarks Giving Different Results.



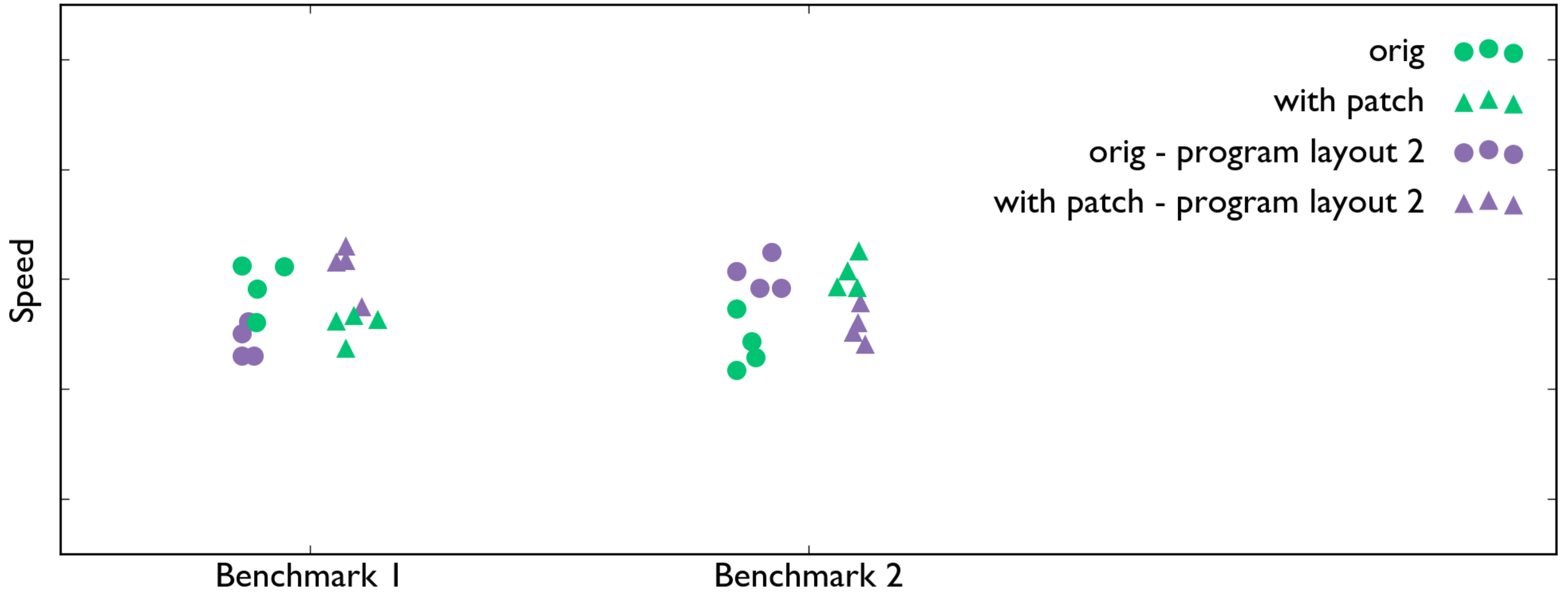
ARM

Noise

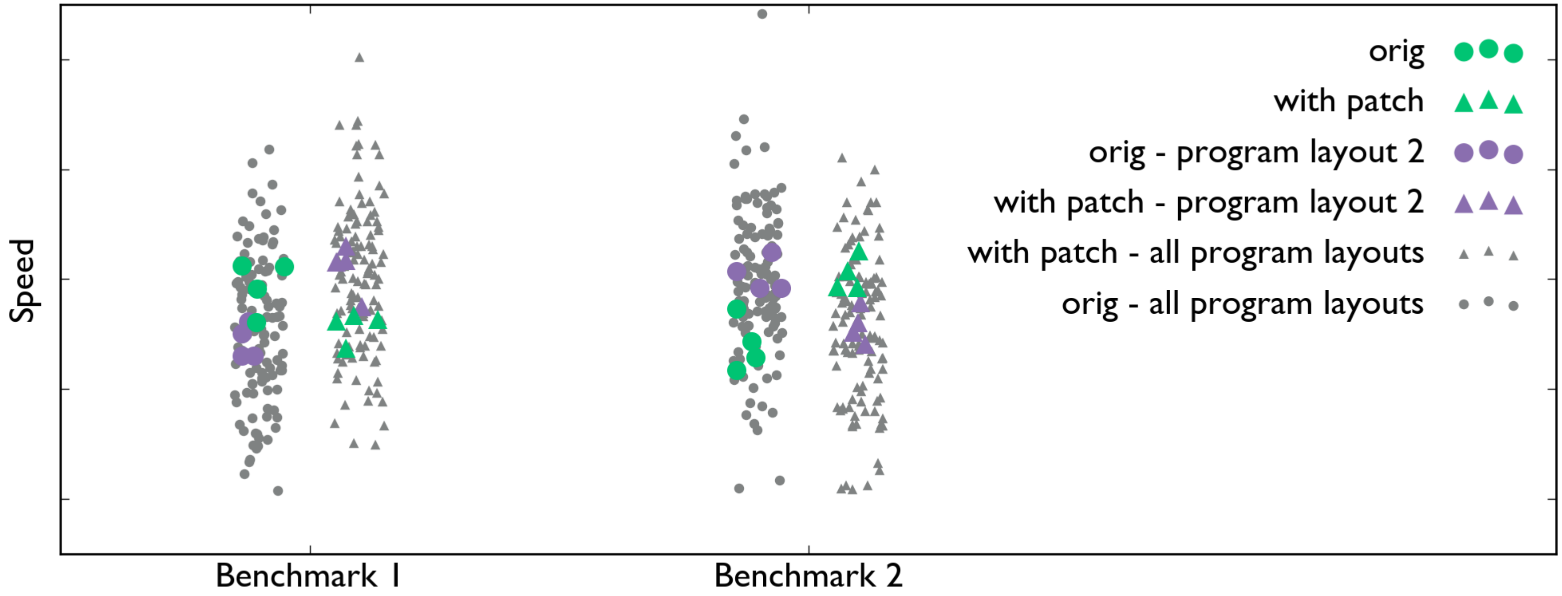


ARM

Noise + Chaotic Behavior



Getting the Whole Space to Avoid Drawing Wrong Conclusion



ARM

Codegen Change: Often unclear if Good or Bad.

- **Which benchmarks matter?**
  Change is often good for one benchmark, bad for the other.

- **Which micro-architectures?**
  Change can be good for one micro-architecture, bad for the other.

Solutions not purely technical

➔ *Not covered further here*

- **Noisy system:**
  Same program running at different speeds when executed multiple times
- **Chaotic system:**
  Small program causes non-linear effect on execution speed.

Solutions can be purely technical

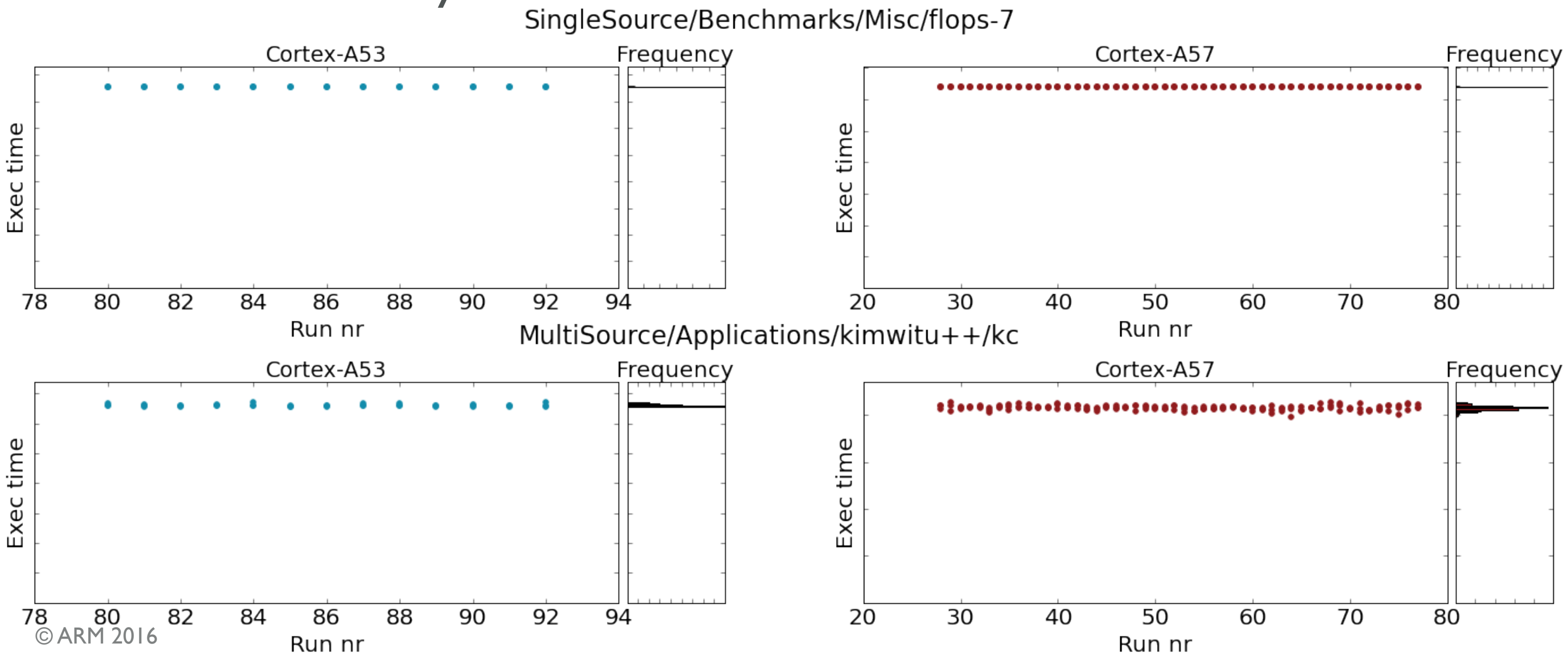
➔ *Topic of this presentation*

ARM

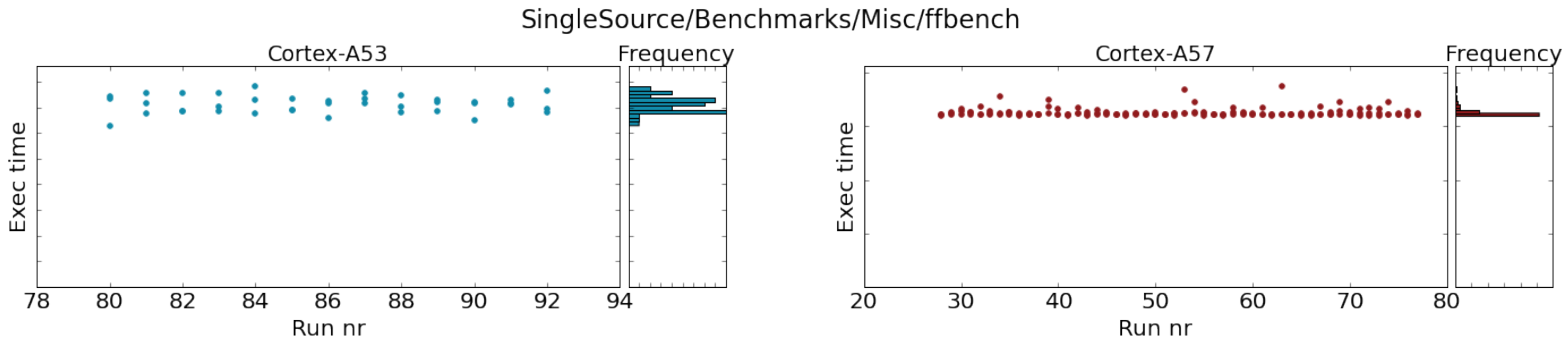
Some Characteristics of Noisy Performance

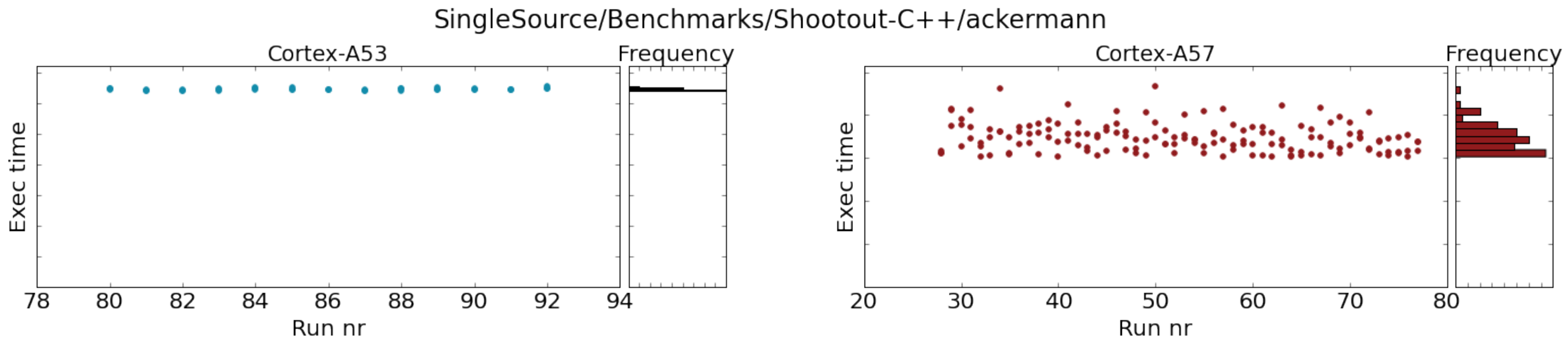
Typical Noise when Running a Binary Multiple Times

- Programs in the test-suite, running on Cortex-A53 and Cortex-A57.
- Most are relatively low-noise:

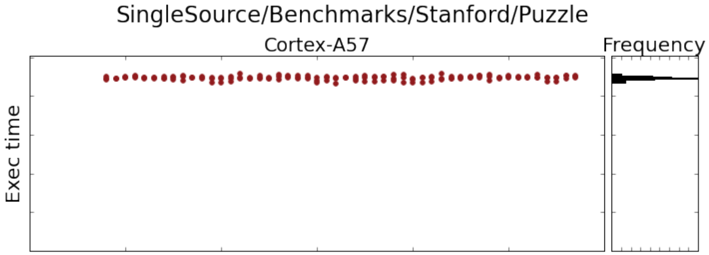


Noise is not Consistent Between Cores

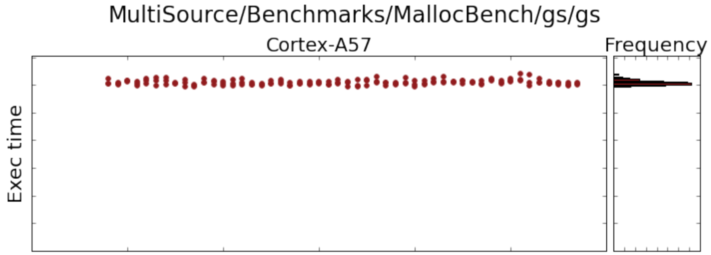




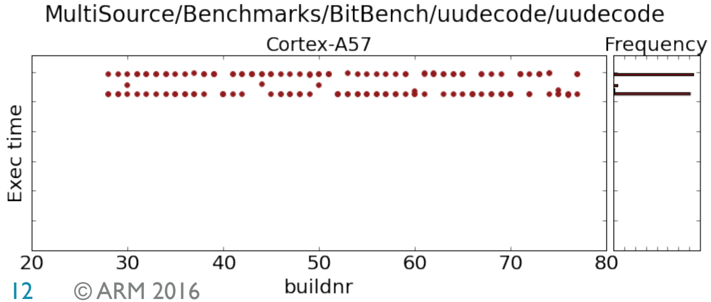
Noise is Distributed in Many Different Ways



Normal

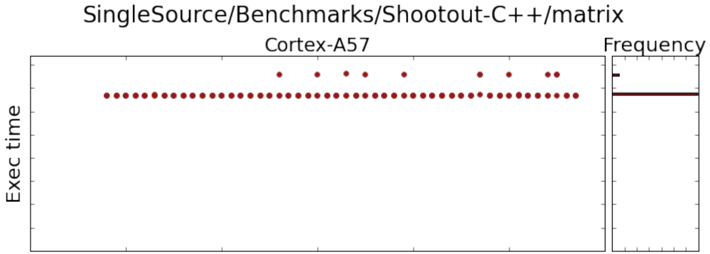


Skewed Normal

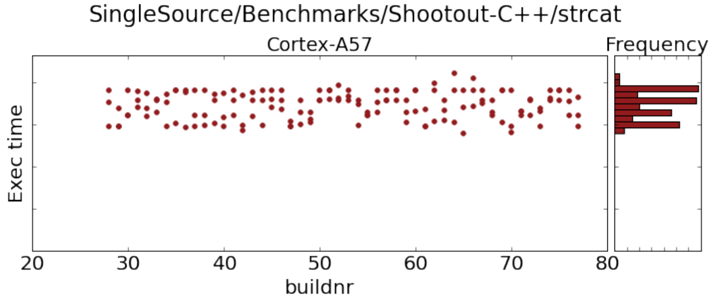


Bimodal

Skewed Bimodal



Quad-Modal?



Some Examples of Chaotic Performance

Rahman et al, WBIA 2009

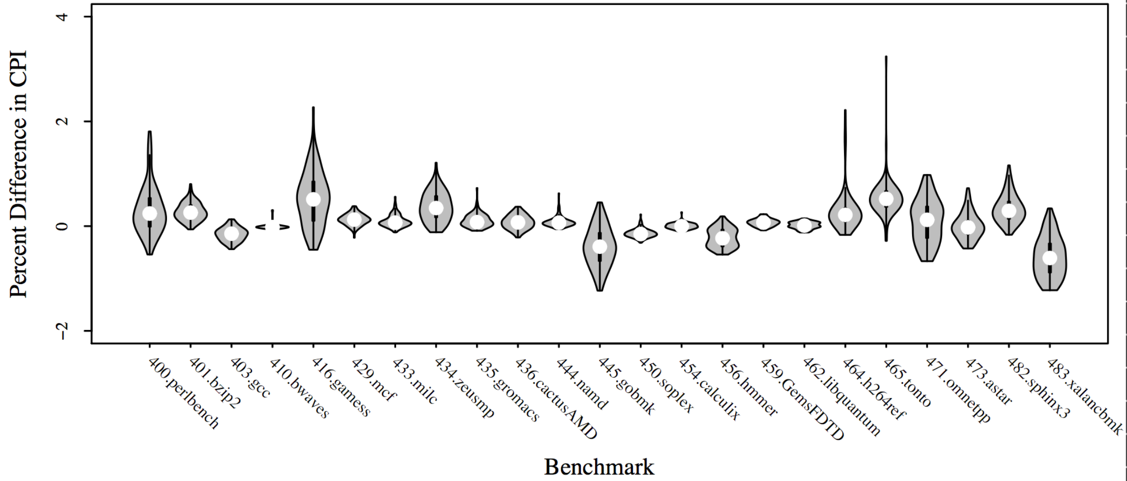


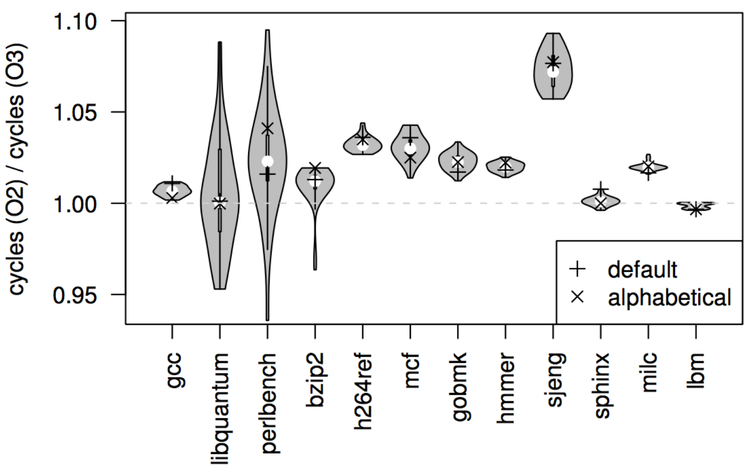
Figure 1: Violin plots for SPEC CPU 2006 percentage performance variation with object reorder

Benchmark Name	Event			
	Branch MPKI	L1 I-Cache Misses	L2 Cache Misses	Combined Estimator
400.perlbench	yes	yes	-	yes
401.bzip2	-	-	-	-
403.gcc	yes	yes	yes	yes
410.bwaves	-	-	-	-
416.gamess	yes	-	yes	yes
429.mcf	yes	-	-	yes
433.milc	-	-	-	-
434.zeusmp	-	-	yes	yes
435.gromacs	yes	-	-	yes
436.cactusADM	-	-	-	-
444.namd	-	-	yes	-
445.gobmk	yes	yes	-	yes
450.soplex	-	-	yes	-
454.calculix	-	-	yes	yes
456.hmmer	yes	-	-	yes
459.GemsFDTD	-	-	-	-
462.libquantum	yes	-	yes	yes
464.h264ref	yes	-	-	yes
465.tonto	yes	yes	-	yes
471.omnetpp	yes	-	-	yes
473.astar	yes	-	yes	yes
482.sphinx3	-	-	-	-
483.xalancbmk	yes	-	-	yes

Table 1: "Yes" means that the null hypothesis of "no correlation" is rejected with $p \leq 0.05$, i.e., with 95% probability, the given measurement is correlated with CPI.

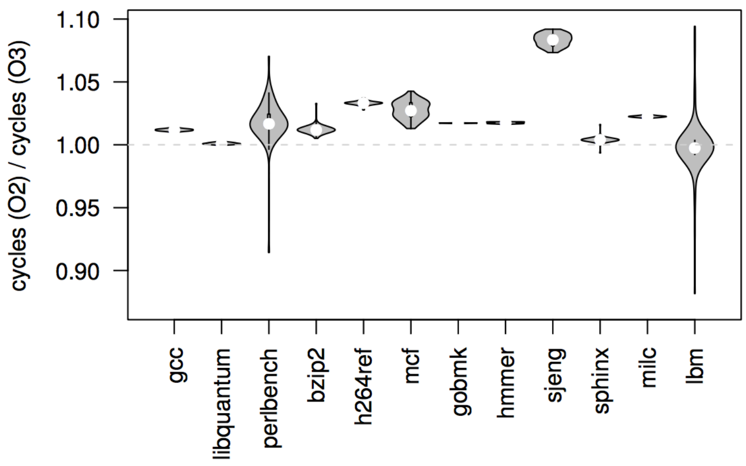
"Studying Microarchitectural Structures with Object Code Reordering"
http://doi.acm.org/10.1145/1791194.1791196

Mykowitz et. al, ASPLOS 2009



(b) All Benchmarks

Figure 2. The effect of link order on Core 2.



(b) All Benchmarks

Figure 3. The effect of UNIX environment size on the speedup of *O3* on Core 2.

"Producing Wrong Data Without Doing Anything Obviously Wrong"
http://doi.acm.org/10.1145/1508244.1508275

ARM

Kalibera et al, ISMM 2013

- Noise with code layout variation is typically a few times higher.

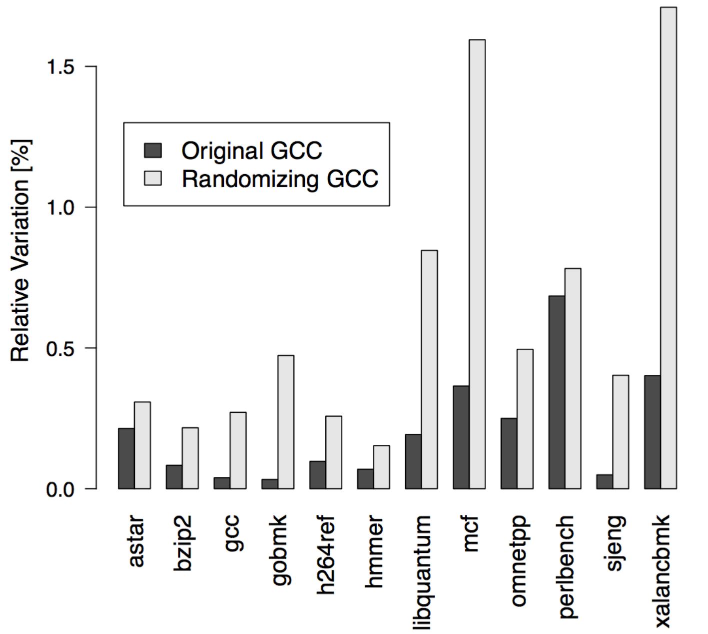


Figure 2. Relative variation with randomising and original gcc (reference size).

"Rigorous Benchmarking in Reasonably Time"
http://doi.acm.org/10.1145/2464157.2464160

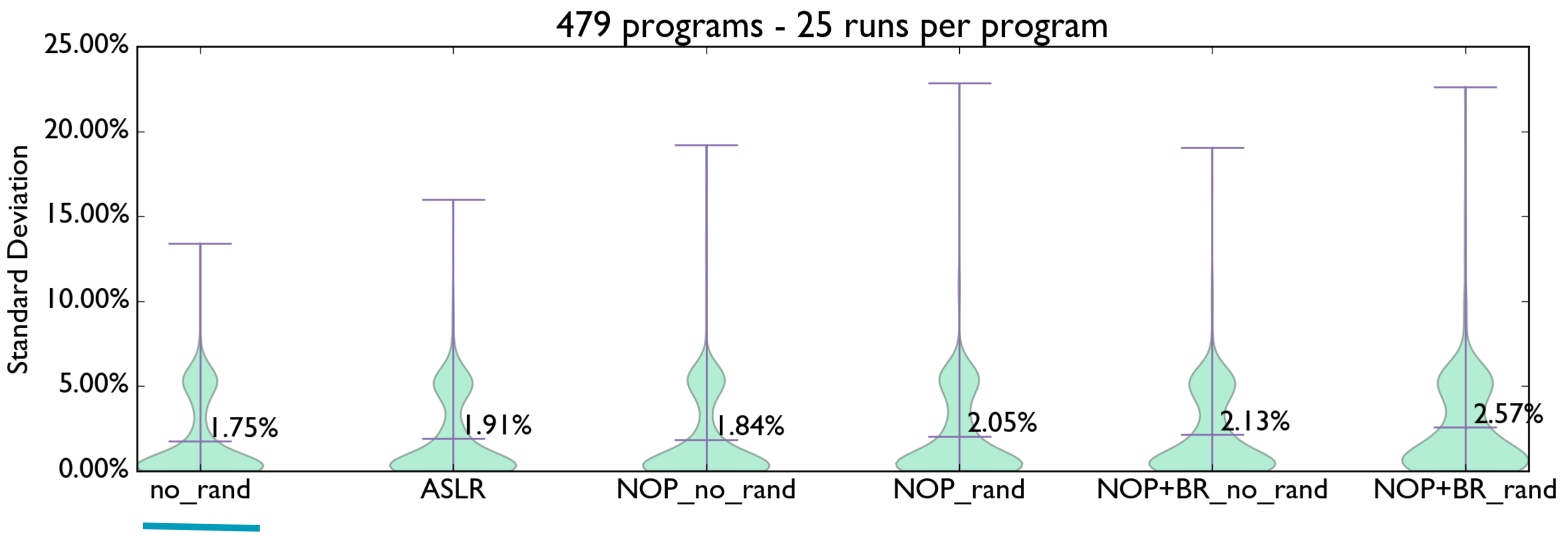
Own Experiments to Characterize Chaotic Performance

Do These Randomize Enough?

- The cited articles at best change the order of functions, i.e. offsets between functions.
- It shouldn't be that hard to also randomize intra-function offsets.
- Try out 2 approaches:
  - Insert random number of bytes after BB ending in unconditional branch.
  - Make all BB end in unconditional branch. Add random number of bytes after each BB.
- Implemented as a MachineFunctionPass for AArch64. Ran experiments using 479 programs in from test-suite, SPEC2000, and a number of other commercial benchmark suites.

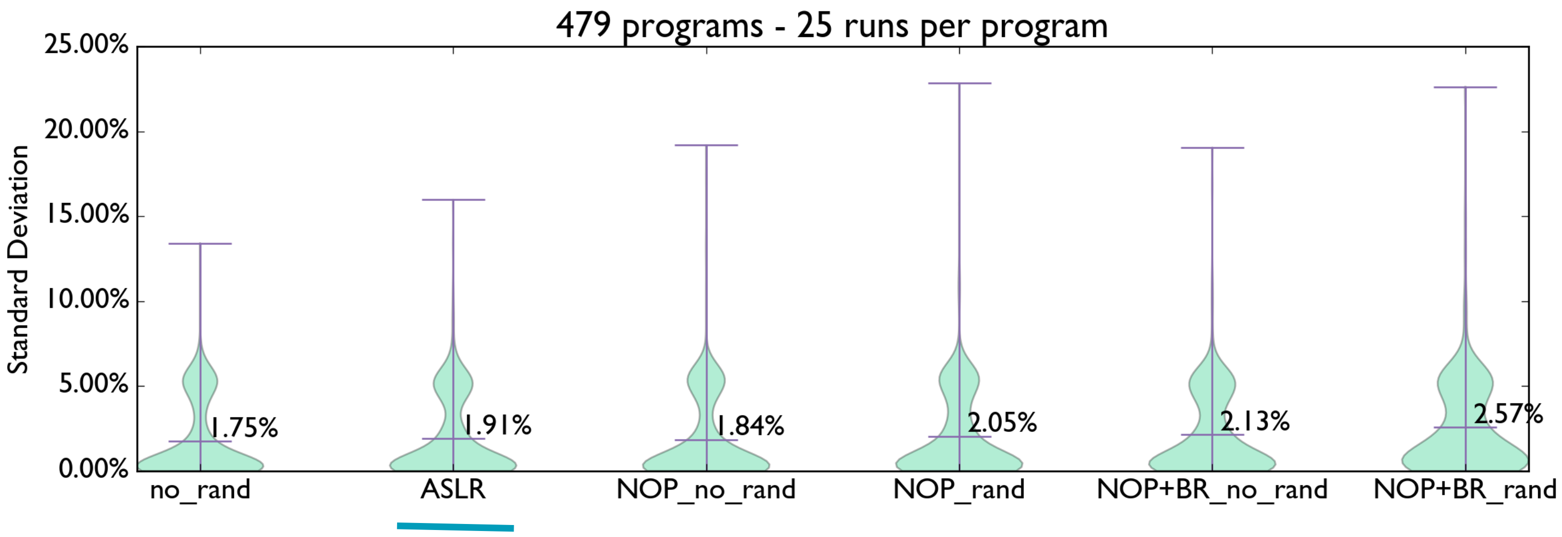
ARM

Relative Standard Deviation over 25 runs, for all 479 programs

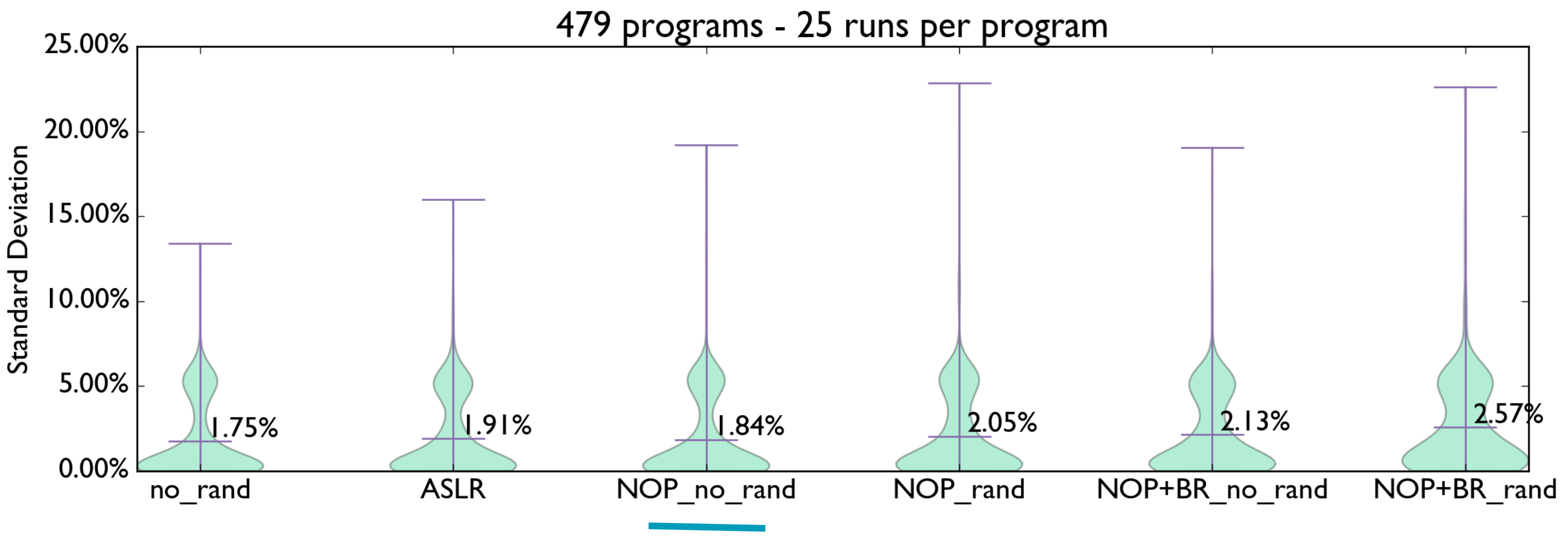


ARM

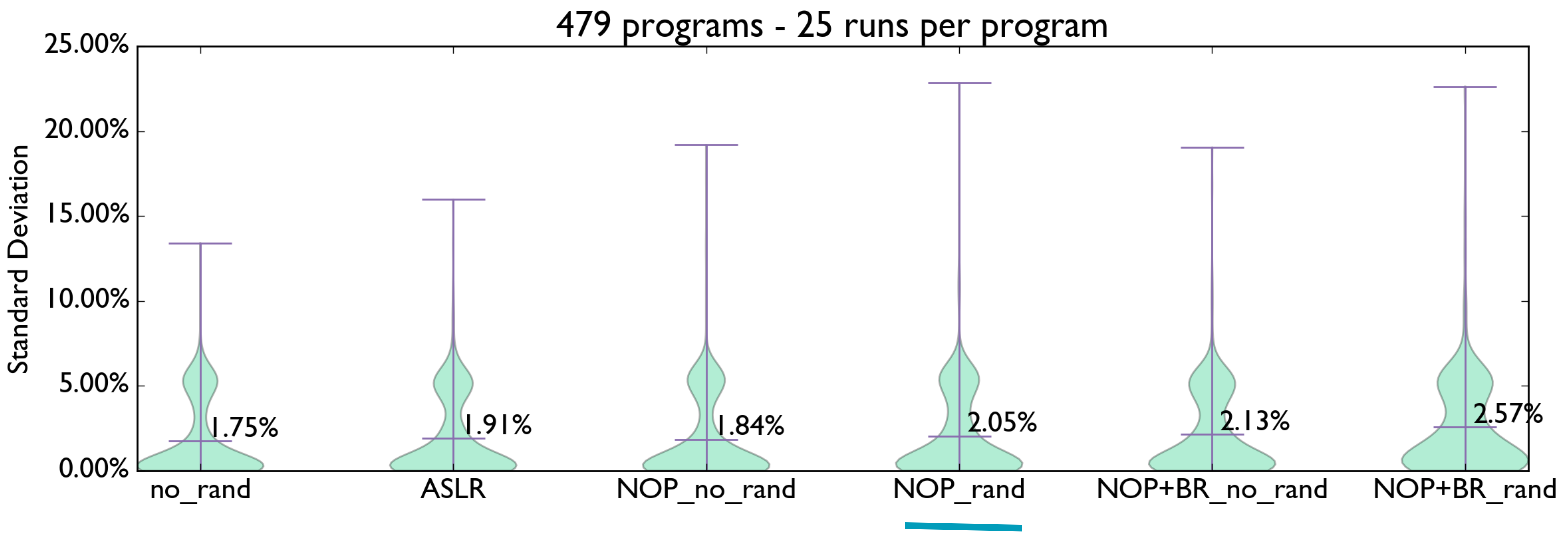
Relative Standard Deviation over 25 runs, for all 479 programs



Relative Standard Deviation over 25 runs, for all 479 programs

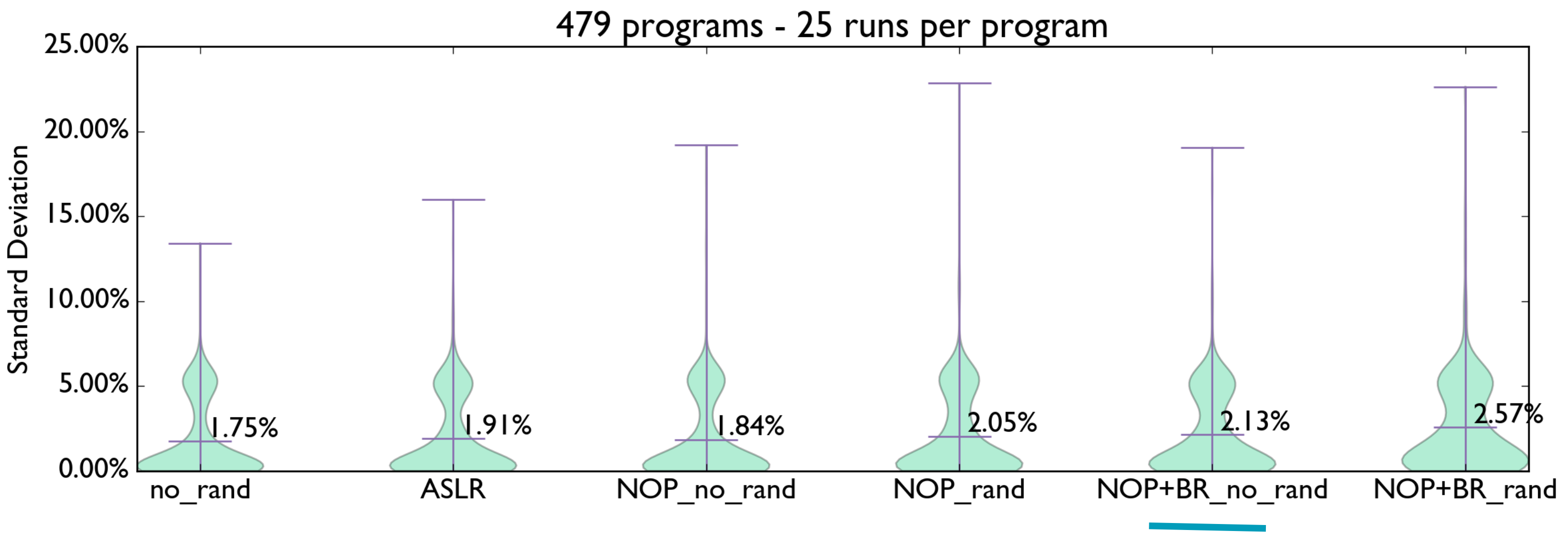


Relative Standard Deviation over 25 runs, for all 479 programs



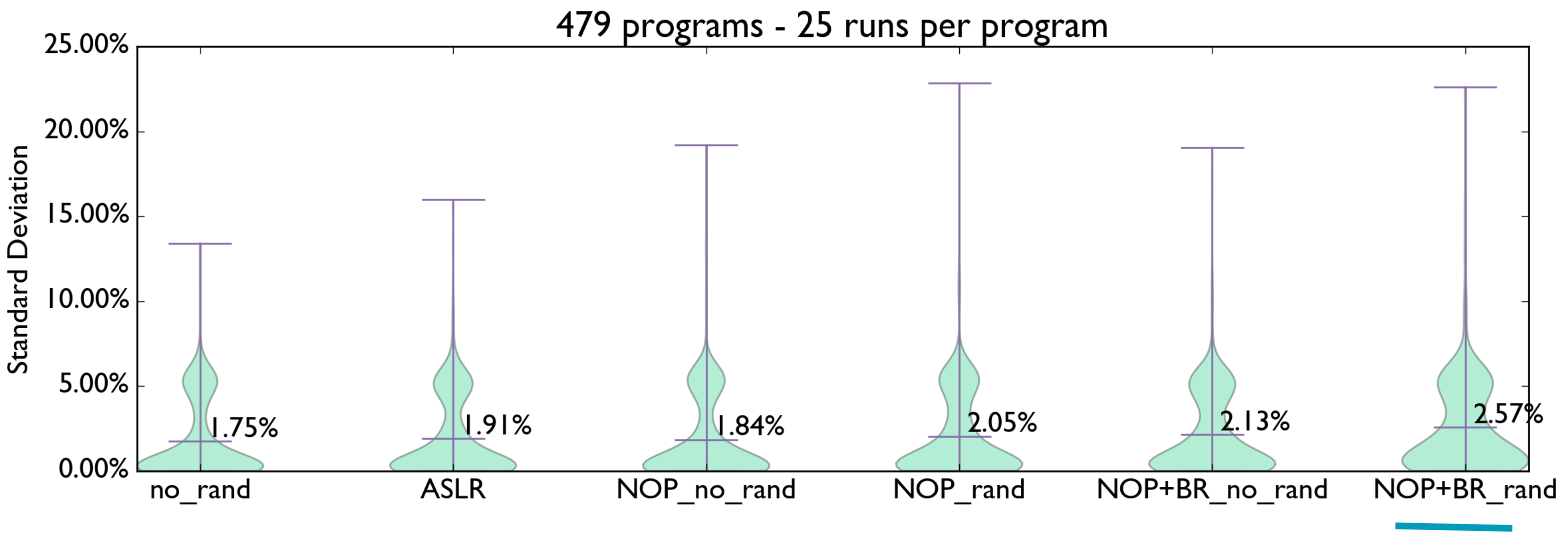
ARM

Relative Standard Deviation over 25 runs, for all 479 programs



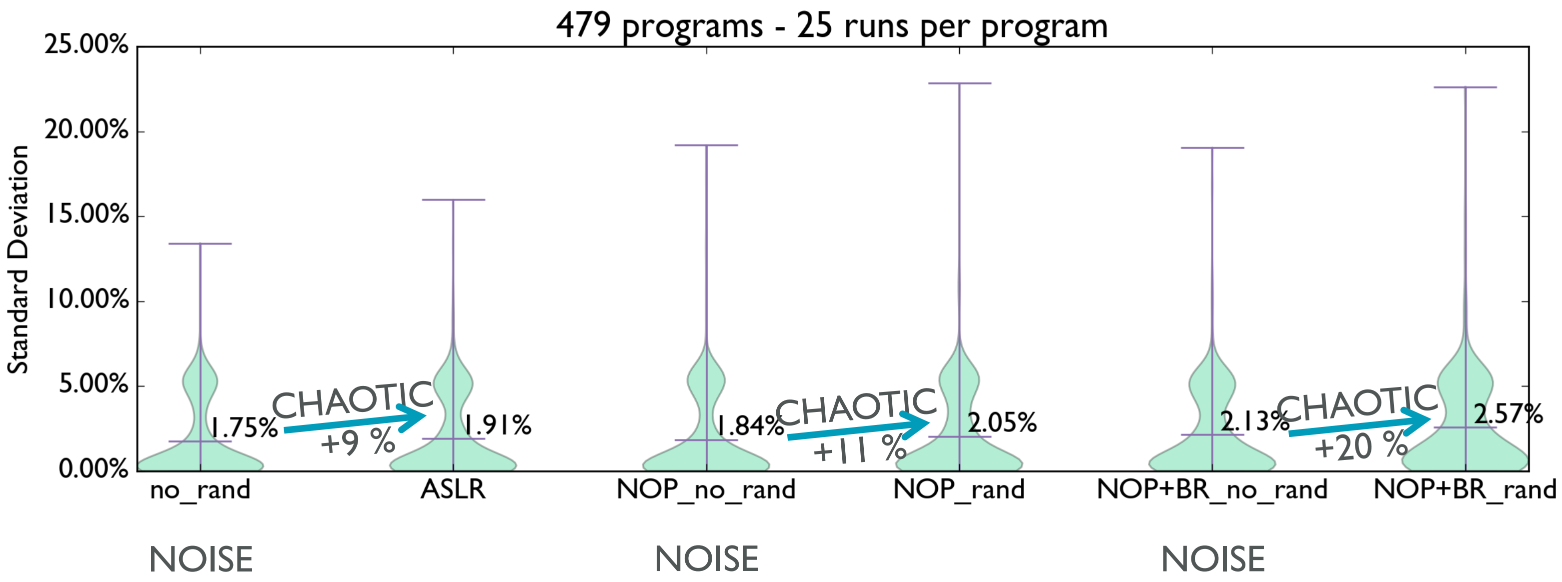
ARM

Relative Standard Deviation over 25 runs, for all 479 programs



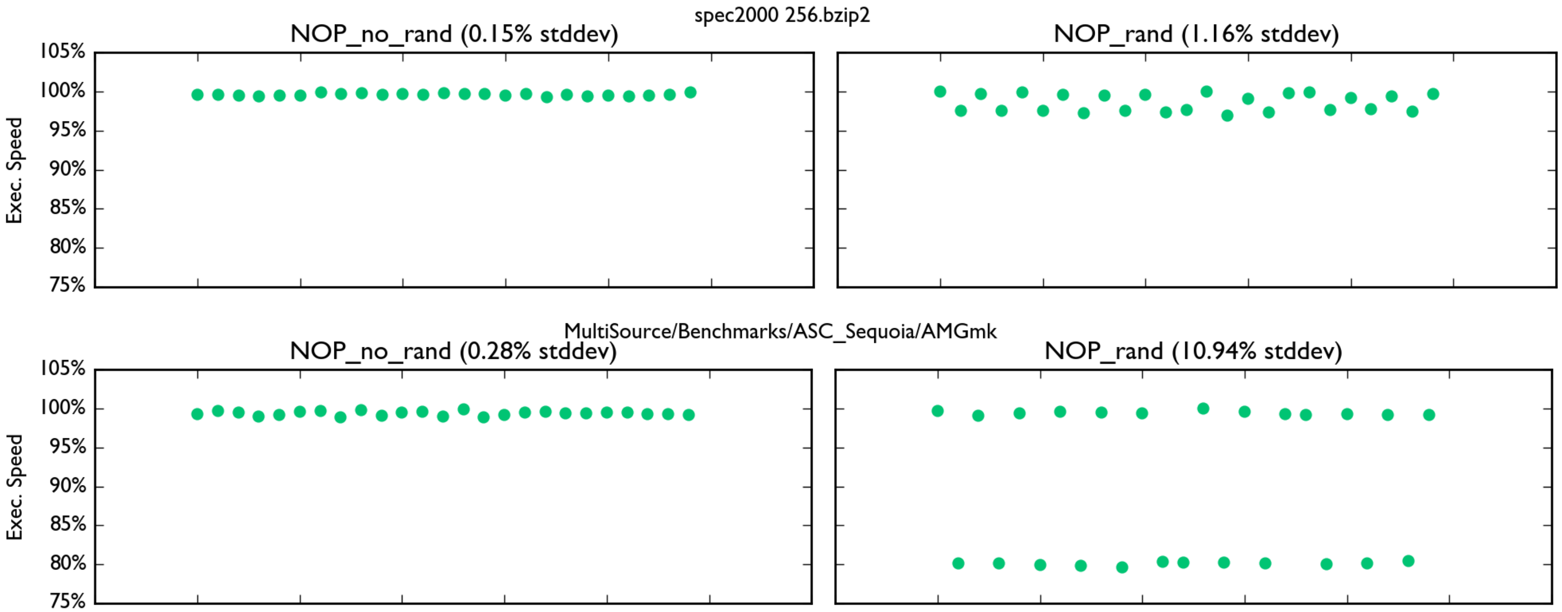
ARM

Relative Standard Deviation over 25 runs, for all 479 programs



ARM

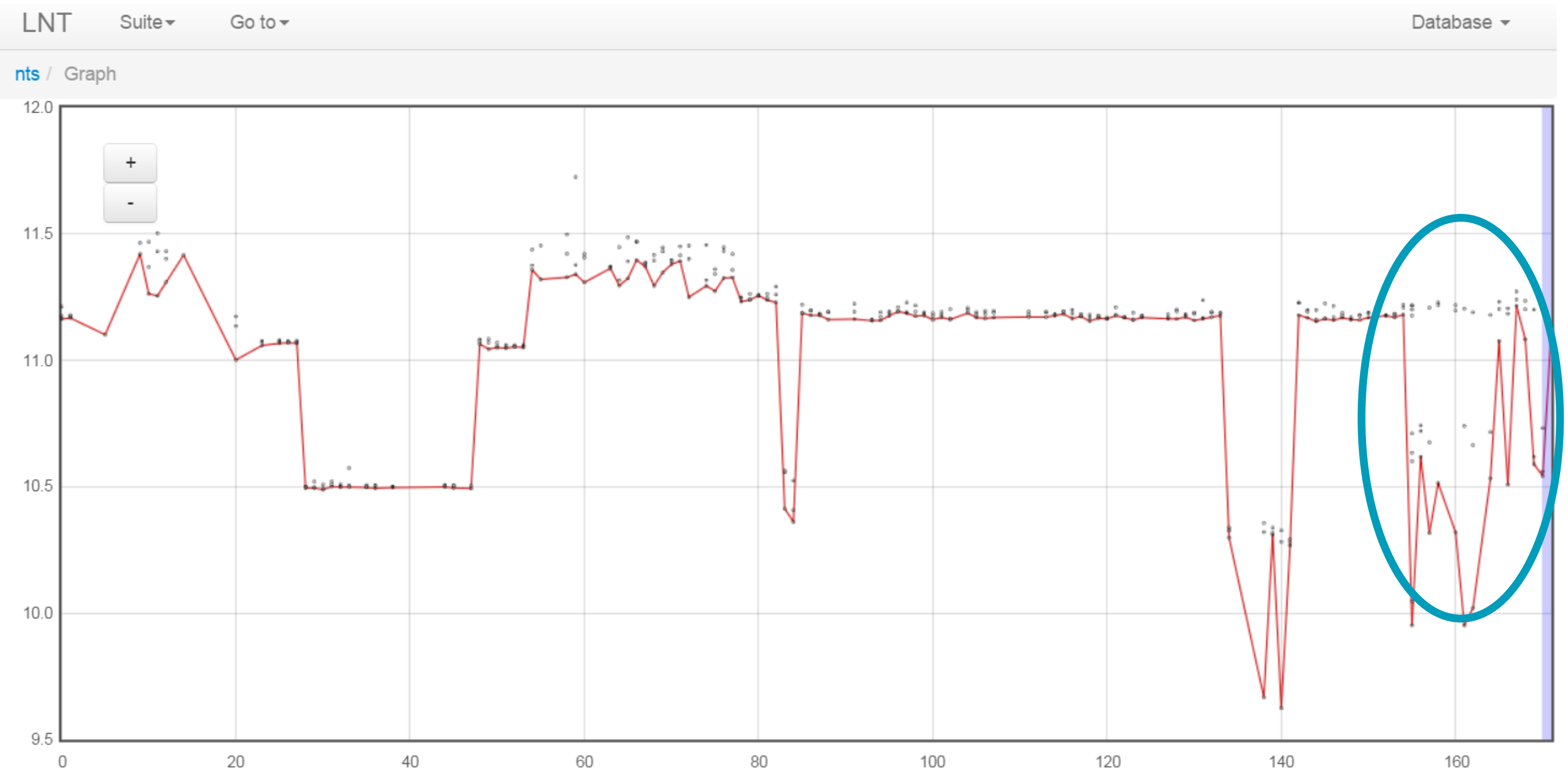
Highly Chaotic Performance on Some Programs



Even if it's only a few programs, each one requires manual investigation!

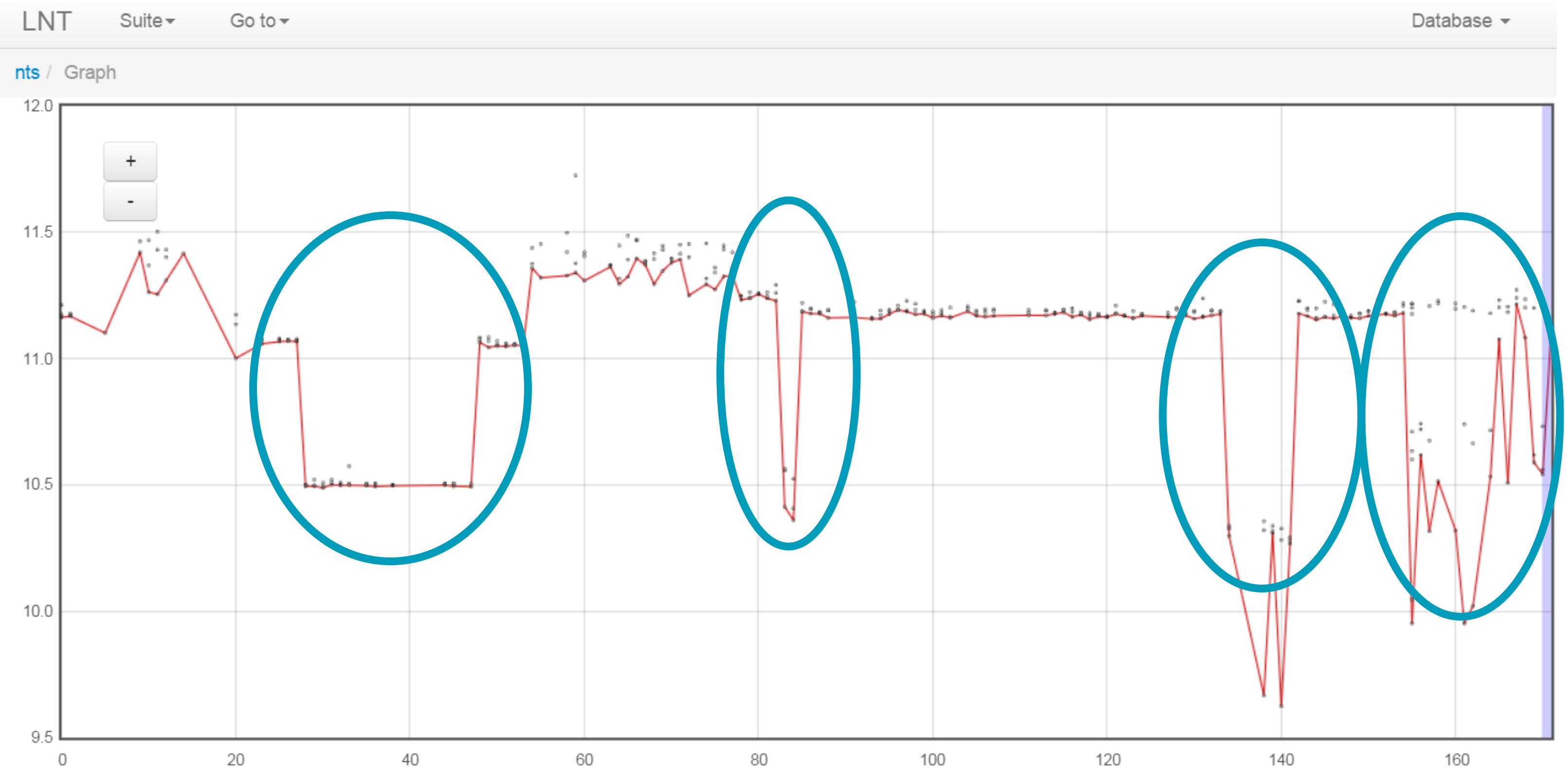
ARM

Did We See This in Trend Graphs But Didn't Notice?



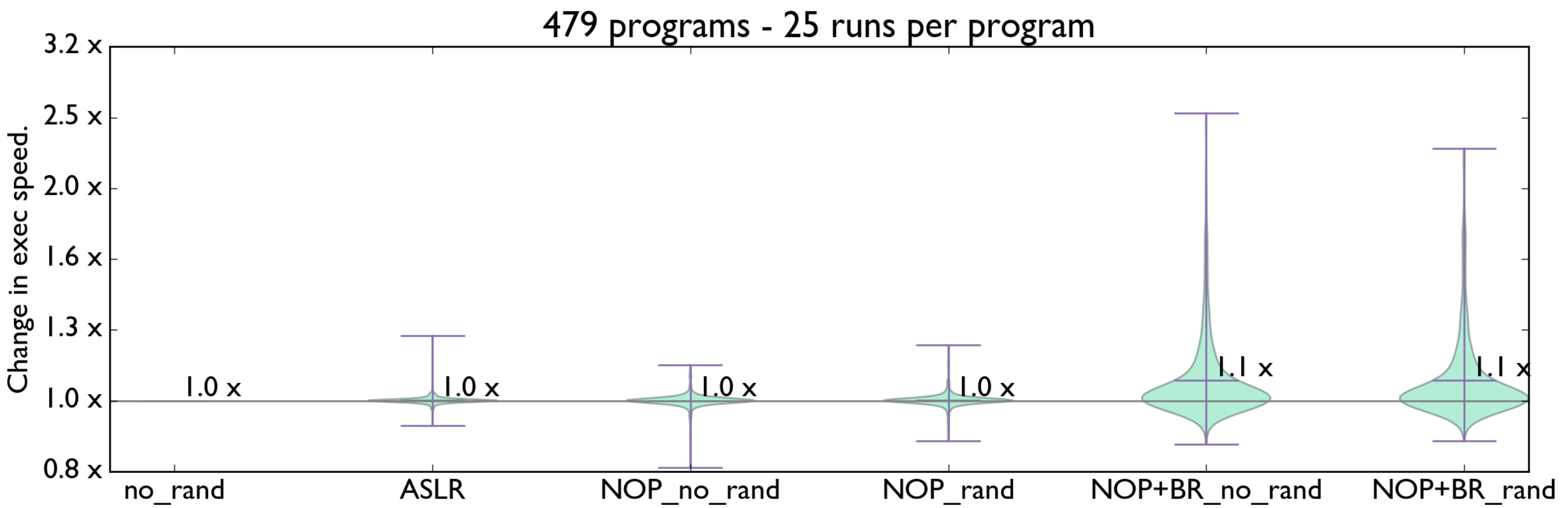
ARM

Did We See This in Trend Graphs But Didn't Notice?



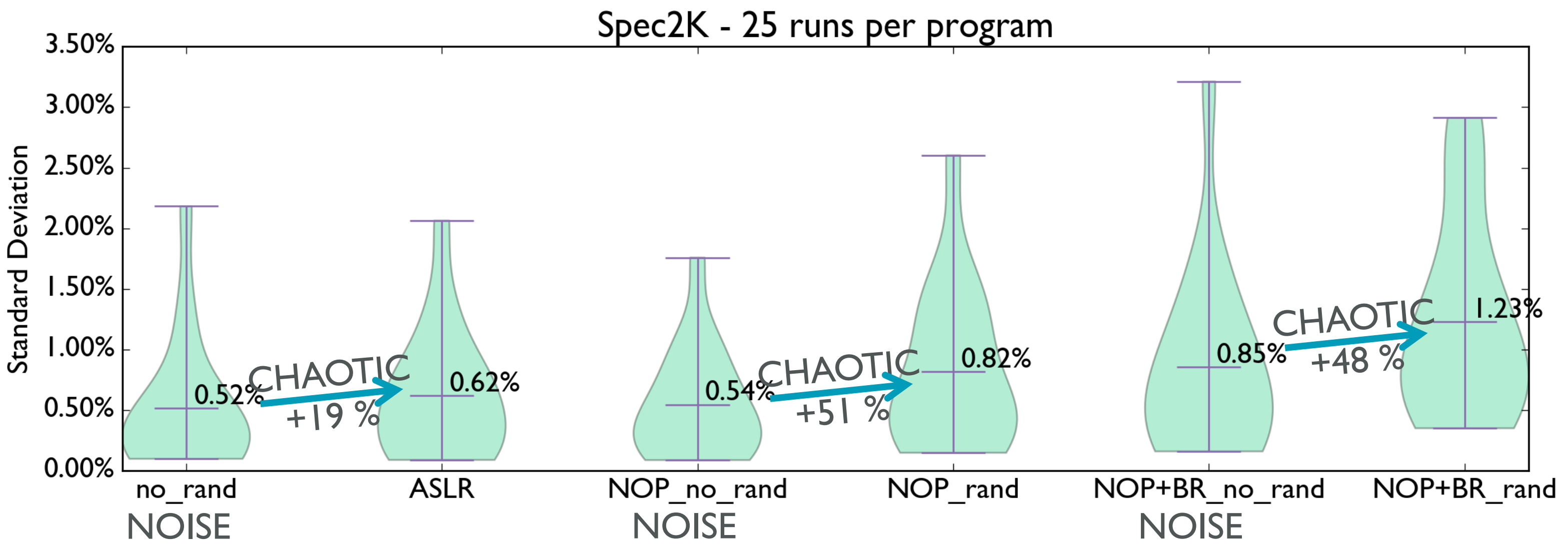
ARM

How does **performance** change with randomization (all 479 programs)?



ARM

Relative Standard Deviation of 25 runs, for SPEC2000(x programs)



SPEC2000 about 3x less noisy, chaotic behavior has more weight.

ARM

How to Measure Effect of Patch Correctly.

- Inject randomization so that the whole population of program layouts gets sampled.
- Do enough runs to get statistically valid results.

- … but isn't this going to be painfully slow?
- Our performance tracking bots already are too slow – when they only do a fraction of the necessary runs to get fully statistically valid results?

ARM

Can Coping with Noise and Measurement Bias be done Efficiently?

Suggestions in Literature

	Avoiding Bias			Increasing Speed		
	Many programs	**Randomize**	**Confidence Intervals**	**Est. sample size**	**Reduce inputs**	**Multiple revision**
Producing Wrong Data Without Doing Anything Obviously Wrong!, ASPLOS09	+	+	+			
Variability in Architectural Simulations of Multi-threaded Workloads, HPCA03		+	+	+		
A study of Performance Variations in the Mozilla Firefox Web Browser. ACSC13		-	+	+		+
Stabilizer: Statistically Sound Performance Evaluation. ASPLOS13		++				
Simulation of Comp. Arch.: Simulators, Benchmarks, Methodologies, Recommendations. IEEE TC 2006					+	
Rigorous Benchmarking in Reasonable Time. ISMM13	+	+	+	++	±	

ARM

What does LNT/test-suite already implement?

	Avoiding Bias			Increasing Speed		
	Many programs	Randomize	Confidence Intervals	Est. sample size	Reduce inputs	Multiple revision
LNT supporting test-suite/Externals	±					
Automatic reruns on changed result			?	?		
Test-suite has support for "SMALL"					±	
Multi-rev analysis						±
Multi-sampling (avoiding noise)			?	?		
Hash of binary						±

ARM

LNT/test-suite Ideas for Further Improvements

Post-commit (bot): 1 hour time.

1. Multi-revision analysis *with exponentially weighted average?*
2. Test-suite: Reduce SMALL size, see *Rigorous Benchmarking in Reasonable Time.*

Pre-commit: hours/days time.

3. Multi-run analysis *with layout randomization that doesn't break layout optimizations?*

4. Test-suite: add more benchmarks – not much seeming overlap between benchmark suite characteristics at the moment.
5. Further progress cmake/lit-ification of test-suite to easily apply techniques across all benchmarks.
6. Auto-tune number of samples to be (program,platform)-specific?

ARM

Summary

- Noisy and chaotic performance makes evaluating code generation changes harder.
- Randomizing program layout can be achieved with a simple late MachineFunctionPass, to avoid measurement bias.
- A few improvements to LNT/test-suite can probably go a long way to coping further with noise and chaotic performance without blowing up experimentation time.

ARM

ARM