Towards Ameliorating Measurement Bias in Evaluating Performance of Generated Code

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.
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.
Noise

The graph shows the comparison between the original ('orig') and the patched ('with patch') data across two benchmarks: Benchmark 1 and Benchmark 2. The y-axis represents speed, while the x-axis distinguishes between the two benchmarks. The data points indicate that the patch has a noticeable effect on improving speed.
Noise + Chaotic Behavior

![Graph showing comparison between original and patched programs for two benchmarks.]

- **orig**
- **with patch**
- **orig - program layout 2**
- **with patch - program layout 2**

**Y-axis:** Speed

**X-axis:**
- Benchmark 1
- Benchmark 2
Getting the Whole Space to Avoid Drawing Wrong Conclusion
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
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

SingleSource/Benchmarks/Misc/ffbench

Cortex-A53

Exec time

Run nr

Frequency

78 80 82 84 86 88 90 92 94

Cortex-A57

Exec time

Run nr

Frequency

20 30 40 50 60 70 80

SingleSource/Benchmarks/Shootout-C++/ackermann

Cortex-A53

Exec time

Run nr

Frequency

78 80 82 84 86 88 90 92 94

Cortex-A57

Exec time

Run nr

Frequency

20 30 40 50 60 70 80
Noise is Distributed in Many Different Ways

- Normal
- Skewed
- Bimodal
- Quad-Modal?
Some Examples of Chaotic Performance
Rahman et al, WBIA 2009

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

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

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.
Mykowtiz et. al, ASPLOS 2009

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

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
Kalibera et al, ISMM 2013

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

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

Figure 2. Relative variation with randomising and original gcc (reference size).
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.
Relative Standard Deviation over 25 runs, for all 479 programs
Relative Standard Deviation over 25 runs, for all 479 programs

479 programs - 25 runs per program

- no_rand: 1.75%
- ASLR: 1.91%
- NOP_no_rand: 1.84%
- NOP_rand: 2.05%
- NOP+BR_no_rand: 2.13%
- NOP+BR_rand: 2.57%
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
Relative Standard Deviation over 25 runs, for all 479 programs
Relative Standard Deviation over 25 runs, for all 479 programs

479 programs - 25 runs per program

- NOISE
  - no_rand: 1.75% +9%
  - ASLR: 1.91%
  - NOP_no_rand: 1.84% +11%
  - NOP_rand: 2.05%
  - NOP+BR_no_rand: 2.13% +20%
  - NOP+BR_rand: 2.57%

- CHAOTIC
  - NOP_no_rand: 1.84% +11%
  - NOP_rand: 2.05%
  - NOP+BR_no_rand: 2.13% +20%
  - NOP+BR_rand: 2.57%
Highly Chaotic Performance on Some Programs

Even if it’s only a few programs, each one requires manual investigation!
Did We See This in Trend Graphs But Didn’t Notice?
Did We See This in Trend Graphs But Didn’t Notice?
How does **performance** change with randomization (all 479 programs)?
Relative Standard Deviation of 25 runs, for SPEC2000(x programs)

SPEC2000 about 3× less noisy, chaotic behavior has more weight.
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?
Can Coping with Noise and Measurement Bias be done Efficiently?
### Suggestions in Literature

<table>
<thead>
<tr>
<th>Avoiding Bias</th>
<th>Increasing Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

**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
What does LNT/test-suite already implement?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Avoiding Bias</th>
<th>Increasing Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>LNT supporting test-suite/Externals</td>
<td>±</td>
<td></td>
</tr>
<tr>
<td>Automatic reruns on changed result</td>
<td></td>
<td>±</td>
</tr>
<tr>
<td>Test-suite has support for “SMALL”</td>
<td></td>
<td>±</td>
</tr>
<tr>
<td>Multi-rev analysis</td>
<td></td>
<td>±</td>
</tr>
<tr>
<td>Multi-sampling (avoiding noise)</td>
<td></td>
<td>±</td>
</tr>
<tr>
<td>Hash of binary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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?
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.