SFO15-301: Benchmarking Best Practices 101

Bernie Ogden
Maxim Kuvyrkov
Overview

- What is benchmarking?
- Design
  - Designing a benchmarking experiment
- Repeatability
  - Can we repeat the result?
- Reproducibility & Reporting
  - Can others repeat the result?
  - What does a good report look like?
What Is Benchmarking?
What is benchmarking?

- An experiment, like any other
- Scientific method
  - Form a hypothesis
  - Test, with control of variables
  - Report results…
  - …with enough detail for others to replicate
What is benchmarking?

- Slow, hard work
- If we don’t do it right
  - We waste effort
  - We fail to deliver member value
  - We look bad
- But if we over-do it, we also waste effort
  - No experiment is perfect
  - We must be aware of limitations, and understand and explain their consequences
Design
Goal

Establish goal: what am I trying to do?

- Measure performance improvement due to code change
- Compare performance of 32- and 64-bit builds of libfoo.so
Experiment

In light of goal, design experiment

- Identify question to ask
- Select testing domain
- Identify variables
- Consider how to control variables
Testing Domain (1/2)

Select appropriate testing domain for the effect being measured. For instance:

CPU-specific, CPU-bound effect

- Test on single implementation of that CPU
- Example: FP performance on Cortex-A57
Testing Domain (2/2)

Architecture specific, CPU-bound effect
- Test on range of CPUs that implement arch
- Example: FP performance on ARMv7-A

Architecture-generic, memory-bound
- Test on range of SoCs and implementations
- Example: AArch32 memcpy performance on v7-A & v-8A
Know Your Target

Know major hardware features

● Core, frequency, cache hierarchy...

Have a sense of ‘background activity’

● Determine post-boot ‘settling’ time
● Check background processes, memory use
● What interrupts are there, where do they go?
Know Your Benchmark

Purpose
- Static codegen, JIT

General characteristics
- Code size, mem load

What is it exercising?
- Pointer chasing, FP

What is it sensitive to?
- BP, memory system

Phase behaviour
- Physics, rendering

Run & reporting rules
Know The Intersection

Controlling all variables, study behaviour of benchmark on target

Run multiple times to determine variability

Should be able to converge on some average to a narrow interval with high confidence
Why Bother?

- Interpretation of results
- Identification of significant variables
- Identification of benchmark subsets
Repeatability
Repeatability

An experiment is repeatable if **one team** can repeatedly run the same experiment **over short periods of time** and get the same results.
## Control Variables

<table>
<thead>
<tr>
<th>Source</th>
<th>CPU frequency</th>
<th>Cache warmup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries</td>
<td>Core migration</td>
<td>Cache hierarchy</td>
</tr>
<tr>
<td>Toolchain</td>
<td>Interrupts</td>
<td>Code layout</td>
</tr>
<tr>
<td>Build env</td>
<td>Thermal throttling</td>
<td>Memory controller</td>
</tr>
<tr>
<td>OS image</td>
<td>Power management</td>
<td>Etc etc etc etc...</td>
</tr>
<tr>
<td>Firmware</td>
<td>ASLR</td>
<td></td>
</tr>
<tr>
<td>Hardware</td>
<td>MMU effects</td>
<td></td>
</tr>
</tbody>
</table>
Countering Noise: Mitigation

Improves run-to-run consistency

Reduces realism

- Reboot for every run
- Warm-up period
- Fix CPU frequency
- Power management/thermal control
- Bind processes to cores
Countering Noise: Statistics

- Some variables cannot be controlled
- Controlling variables reduces realism
- Multiple runs required to show effect of controlling variables
- Multiple runs required for consistency of results
- Changes may affect variance as well as mean
Combined Approach

Reduce target noise sources

- To threshold of unacceptable irrealism
- To point where no further reduction can be achieved

Increase number of runs

- Until effect is repeatable to some acceptable confidence interval
How Much Noise Is Acceptable?

Roughly: the effect size should be larger than some confidence interval

- 0.95 is popular, but won’t include the true mean 1 time in 20

YMMV, depending on the experiment.
Reproducibility
Reproducibility

An experiment is reproducible if external teams can run the same experiment over large periods of time and get commensurate (comparable) results. Achieved if others can repeat what we did and get the same results as us, within the given confidence interval.
Recording

Record *everything*

- Beware of implicit knowledge
- We don’t know what we don’t know
- Recording is cheap
- Future analysis
Recording

Record everything… but the following points are especially important.

- **Full details** of target hardware & OS
- **Exact** toolchain used
- **Exact** benchmark sources
- **Full** build logs
Reporting
Clear, concise reporting allows others to utilise benchmark results

Reports for one audience can slip to others

Do not assume knowledge
  - The reader may not know what your board is...

Include relevant data
  - Make sure all data are available

Define terms
Reporting: Goal

Explain the goal of the experiment

- What decision will it help you to make?
- What improvement will it allow you to deliver?

Explain the question that the experiment asks

Explain how the answer to that question helps you to achieve the goal
Reporting

- Method: Sufficient high-level detail
  - Target, toolchain, build options, source, mitigation
- Limitations: Acknowledge and justify
  - What are the consequences for this experiment?
- Results: Discuss in context of goal
  - Co-locate data, graphs, discussion
  - Include units - numbers without units are useless
  - Include statistical data
  - Use the benchmark’s metrics
Conclusion
It’s a lot of work...

But we have to do it to get meaningful, shareable benchmarking results.

We can (should) limit the amount of work, as long as we understand the consequences and are explicit about them.
Actions?
END
BACKUP/REFERENCE
Graphs: Strong Suggestions
Speedup Over Baseline (1)

Misleading scale
- A is about 3.5% faster than it was before, not 103.5%

Obfuscated regression
- B is a regression
Speedup Over Baseline (2)

Baseline becomes 0
Title now correct
Regression clear
But, no confidence interval.
Speedup Over Baseline (3)

Error bars tell us more: effect D can be disregarded, A is a real, but noisy, effect.

Watch out for scale change
Labelling (1/2)

What is the unit?
What are we comparing?
Labelling (2/2)
Graphs:
Weak Suggestions
Speedup Over Baseline (4)

Can add a mean
Direction of ‘Good’ (1)

Inconsistent
Might be necessary
Direction of ‘Good’ (2)

If you have to change the direction of ‘good’, flag the direction (everywhere)

Can be helpful to flag it anyway
Consistent Order

Presents improvements neatly

But, hard to compare different graphs in the same report
A few high scores make other results hard to see
A couple of alternatives may be more clear...
Scale (2/2)

Separate Outliers

Truncate

A: 2 4 1 1 5 6 7 2 97 85 90

Linaro connect
San Francisco 2015
Noise Mitigation
Mitigation: Settling and warm-up

Monitor `/proc/loadavg` to determine how long system takes to ‘settle down’ after boot.

Run one or two iterations of benchmark to initialize cache, branch predictors, etc, before beginning timing.

Or run the benchmark so many times that warm-up effects are insignificant.
Mitigation: Other Processes/Migration

Use a minimal OS
Shut down non-essential processes
  ● Tricky to generalize reliably
Set CPU affinity
  ● One CPU runs the benchmark, another runs ‘everything else’
Mitigation: Interrupts

Disable/monitor/constrain irqbalance daemon

/proc/irq/*/smp_affinity: where interrupts can go (as far as kernel knows)

/proc/interrupts: where interrupts are going

Disable network

- Fiddly, but doable
- At least disable accidental access
Mitigation: DVFS

cpufreq can set a fixed frequency
Watch out for broken thermal throttling
Don’t try to extrapolate results to different frequencies (you’ve thrown off relative timings to rest of system)
Mitigation: ASLR

ASLR randomizes base of heap and stack
Affects alignment and relative position of data
May cause cache thrashing

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```
Mitigation: MMU

Use largest available page size

- Fewer TLB misses, potentially fewer page faults
- Intuitively: Better performance, less noise

AArch64 supports 4KB & 64KB page size
AArch32+LPAE, and AArch64 support huge pages

- 4K page -> 2MB huge page
- 64k page -> 512MB huge page
Mitigation: Huge Page Gotchas (1/2)

Huge pages can have downsides

- Increased chance of cache thrashing
  - Large address == less random way selection
- Potentially similar effects elsewhere in system, e.g. channel selection in memory controller
Mitigation: Huge Page Gotchas (2/2)

THP collapses pages into huge pages

- Can happen at any time, potentially a noise source

libHugeTLBfs can back heap with huge pages

- Which will affect alignment, introducing noise
Bias: Code Layout

Code layout effects may dominate the effect we are trying to measure.
Can cause cache thrashing, branch mispredicts.
Varies statically and dynamically.
Easily perturbed:
- Link order
- Environment variables
Mitigation: Layout Bias

- Vary experimental conditions
  - PIE, ASLR help in this respect
  - Vary link order, environment size
  - Tooling, e.g. …

- Statistically isolate size of effect, to within some confidence interval
Bare Metal vs Rich OS
Bare Metal vs Rich OS

Bare Metal

● High control / low realism
● Configuration trades control for realism

Rich OS

● High realism / low control
● Configuration trades realism for control
Bare Metal vs Rich OS

Pragmatic considerations

- Some benchmarks hard to run BM
- Longer-running benchmarks less perturbable
- Infrastructure and skills
  - In any given organisation, may be more oriented towards BM or Rich OS
Regression Tracking
Regression Tracking

You’ll have a few bots running a few point builds
Results will be noisy and incomplete
Look out for (informally) significant, and lasting, changes
Regression Tracking

Investigate

Possibly investigate

http://llvm.org/perf/db_default/v4/nts/graph?plot.0=39.1304.3&highlight_run=27797
Regression Tracking

Chrome Performance Dashboard

Brief regressions, ignore