NICTA

The Dos and Don'ts of Benchmarking

Gernot Heiser
NICTA and UNSW

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

… or how not to lie with benchmarks

Benchmarking in Research

- Generally one of two objectives:
  - Show new approach *improves* performance
  - Show otherwise attractive approach *does not undermine* performance

- Requirement: objectivity/fairness
  - Selection of baseline
  - Inclusion of relevant alternatives
  - Fair evaluation of alternatives

- Requirement: analysis/explanation of results
  - Model of system, incorporating relevant parameters
  - Hypothesis of behaviour
  - Results must support hypothesis

Lies, Damned Lies, Benchmarks

NICTA

- Micro- vs macro-benchmarks
- Standard vs ad-hoc
- Benchmark suites, use of subsets
- Completeness of results
- Significance of results
- Baseline for comparison
- Benchmarking ethics
- What is good — analysing the results

Micro- vs Macro-Benchmarks

- Macro-benchmarks
  - Use realistic workloads
  - Measure real-life system performance (hopefully)
- Micro-benchmarks
  - Exercise particular operation, e.g. single system call
  - Good for analysing performance / narrowing down down performance bottlenecks
    - critical operation is slower than expected
    - critical operation performed more frequently than expected
    - operation is unexpectedly critical (because it's too slow)

Micro- vs Macro-Benchmarks

Benchmarking Crime: Micro-benchmarks only

- Pretend micro-benchmarks represent overall system performance

- Real performance can generally not be assessed with micro-benchmarks
- Exceptions:
  - Focus is on improving particular operation known to be critical
  - There is an established base line

Note: My macro-benchmark is your micro-benchmark

- Depends on the level on which you are operating
- Eg: lmbench
  - … is a Linux micro-benchmark suite
  - … is a hypervsior macro-benchmark

Synthetic vs "Real-world" Benchmarks

- Real-world benchmarks:
  - real code taken from real problems
    - Livermore loops, SPEC, EEMBC, …
  - execution traces taken from real problems
  - distributions taken from real use
    - file sizes, network packet arrivals and sizes
  - Caution: representative for one scenario doesn't mean for *every* scenario!
    - may not provide complete coverage of relevant data space
    - may be biased
- Synthetic benchmarks
  - created to simulate certain scenarios
  - tend to use random data, or extreme data
  - may represent unrealistic workloads
  - may stress or omit pathological cases

Standard vs Ad-Hoc Benchmarks

Why use ad-hoc benchmarks?

- There may not be a suitable standard benchmark
  - Example: lack of standardised multi-tasking workloads
- Cannot run standard benchmarks
  - Limitations of experimental system

Why not use ad-hoc benchmarks?

- Not comparable to other work (unless they use the same)
- Poor reproducibility

Facit: Only use ad-hoc benchmarks if you have no other choice

- Justify well what you're doing

Benchmark Suites

- Widely used (and abused!)
- Collection of individual benchmarks, aiming to cover all of relevant data space
- Examples: SPEC CPU{92|95|2000|2006}
  - Originally aimed at evaluating processor performance
  - Heavily used by computer architects
  - Widely (ab)used for other purposes
  - Integer and floating-point suite
  - Some short, some long-running
  - Range of behaviours from memory-intensive to CPU-intensive
    - behaviour changes over time, as memory systems change
    - need to keep increasing working sets to ensure significant memory loads

Obtaining an Overall Score for a BM Suite

NICTA

- How can we get a single figure of merit for the whole suite?
- Example: comparing 3 systems on suite of 2 BMs

Normalise to System Y

Normalise to System X

Geometric mean?

Benchmark	System X		System Y		System Z	
	Abs	Rel	Abs	Rel	Abs	Rel
1	20	2.00	10	1.00	40	4.00
2	40	0.50	80	1.00	20	0.25
Geom. mean		**1.00**		**1.00**		**1.00**

Invariant under normalisation!

...e sense?

Arithmetic mean is meaningless for relative numbers

Rule: *arithmetic* mean for *raw* numbers,
geometric mean for *normalised!* [Fleming & Wallace, '86]

Benchmark Suite Abuse

Benchmarking Crime: Select subset of suite

- Introduces bias
  - Point of suite is to cover a range of behaviour
  - Be wary of “typical results”, “representative subset”
- Sometimes unavoidable
  - some don't build on non-standard system or fail at run time
  - some may be too big for a particular system
    - eg, don't have file system and run from RAM disk...
- Treat with extreme care!
  - can only draw limited conclusion from results
  - cannot compare with (complete) published results
  - need to provide convincing explanation why only subset

Other SPEC crimes include use for multiprocessor scalability
  - run multiple SPECs on different CPUs
  - what does this prove?

Partial Data

- Frequently seen in I/O benchmarks:
  - Throughput is degraded by 10%
    - "Our super-reliable stack only adds 10% overhead"

Almost certainly not true!

  - Why is throughput degraded?
    - latency too high
    - CPU saturated?
  - Also, changes to drivers or I/O subsystem may affect scheduling
    - interrupt coalescence: do more with fewer interrupts
  - *Throughput on its own is useless!*

Throughput Degradation

NICTA

- Scenario: Network driver or protocol stack
  - New driver reduces throughput by 10% — why?
  - Compare:
    - 100 Mb/s, 100% CPU vs 90 Mb/s, 100% CPU
    - 100 Mb/s, 20% CPU vs 90 Mb/s, 40% CPU
  - Correct figure of merit is *processing cost per unit of data*
    - Proportional to *CPU load divided by throughput*
  - Correct overhead calculation:
    - 10 µs/kb vs 11 µs/kb: *10% overhead*
    - 2 µs/kb vs 4.4 µs/kb: *120% overhead*

CPU limited

Latency limited

Benchmarking crime: Show throughput degradation only

- … and pretend this represents total overhead

Significance of Measurements

All measurements are subject to random errors

- Standard scientific approach: Many iterations, *collect statistics*
- Rarely done in systems work — why?
- Computer systems tend to be *highly deterministic*
  - Repeated measurements often give identical results
  - Main exception are experiments involving WANs
- However, it is dangerous to rely on this without checking!
  - Sometimes “random” fluctuations indicate *hidden parameters*

Benchmarking crime: results with no indication of significance

Non-criminal approach:

- Show at least standard deviation of your measurements
- … or state explicitly it was below a certain value throughout
- Admit results are insignificant unless well-separated std deviations

How to Measure and Compare Performance

Bare-minimum statistics:

- At minimum report the mean (μ) and standard deviation (σ)
  - Don't believe any effect that is less than a standard deviation
    - 10.2±1.5 is not significantly different from 11.5
  - Be highly suspicious if it is less than two standard deviations
    - 10.2±0.8 may not be different from 11.5
- Be *very suspicious* if reproducibility is poor (i.e. σ is *not* small)
  Distrust standard deviations of small iteration counts
  - standard deviations are meaningless for small number of runs
  - … but ok if effect ≫ σ
  - The proper way to check significance of differences is Student's t-test!

How to Measure and Compare Performance

NICTA

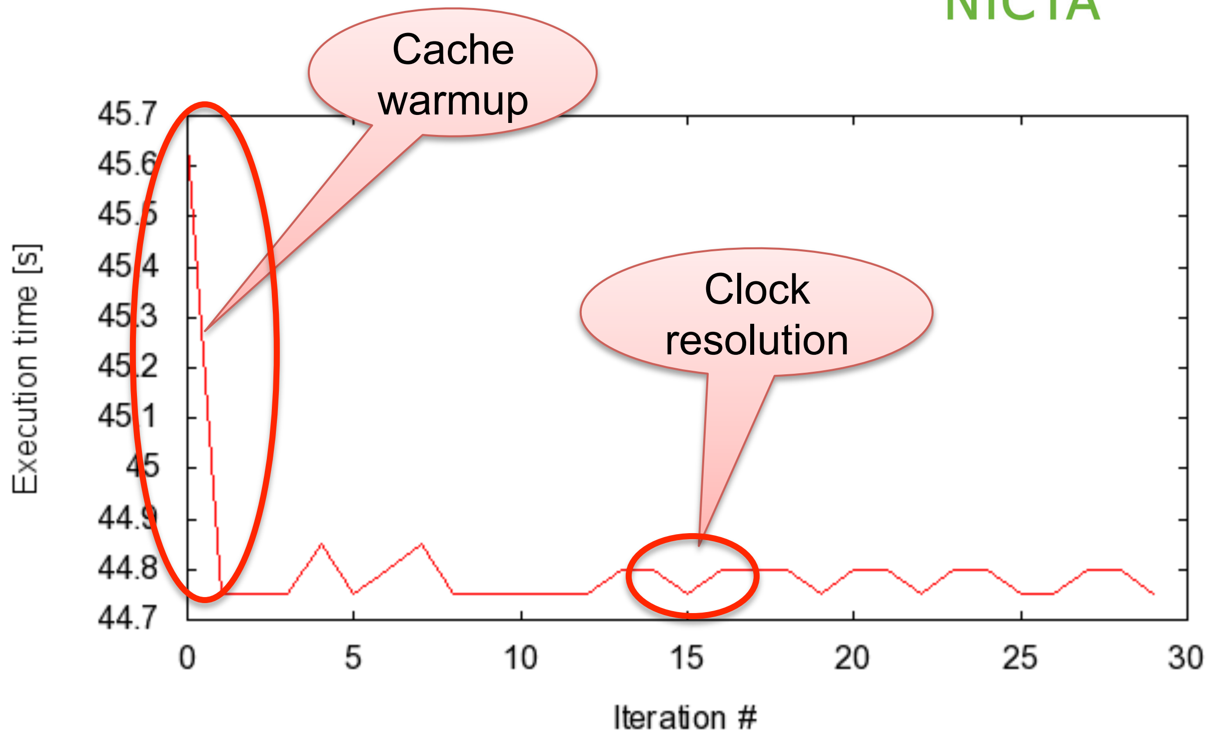
Obtaining meaningful execution times:

- Make sure execution times are long enough
  - What is the granularity of your time measurements?
  - make sure the effect you're looking for is much bigger
  - many repetitions won't help if your effect is dominated by clock resolution
  - do many repetitions in a tight loop if necessary

Example: gzip from SPEC CPU2000

Observations?

- First iteration is special
- 20 Hz clock
  - will not be able to observe any effects that account for less than 0.1 sec



Lesson?

- Need a mental model of the system
  - Here: repeated runs should give the same result
- Find reason (hidden parameters) if results do not comply!

How to Measure and Compare Performance

Noisy data:

- sometimes it isn't feasible to get a “clean” system
  - e.g. running apps on a “standard configuration”
  - this can lead to very noisy results, large standard deviations

Possible ways out:

- ignoring lowest and highest result
- taking the floor of results
  - makes only sense if you're looking for minimum
    - but beware of difference-taking!

Both of these are dangerous, use with great care!

- Only if you know what you are doing
  - need to give a convincing explanation of why this is justified
- Only if you explicitly state what you've done in your paper/report

Real-World Example

Benchmark:

- `300.twolf` from SPEC CPU2000 suite

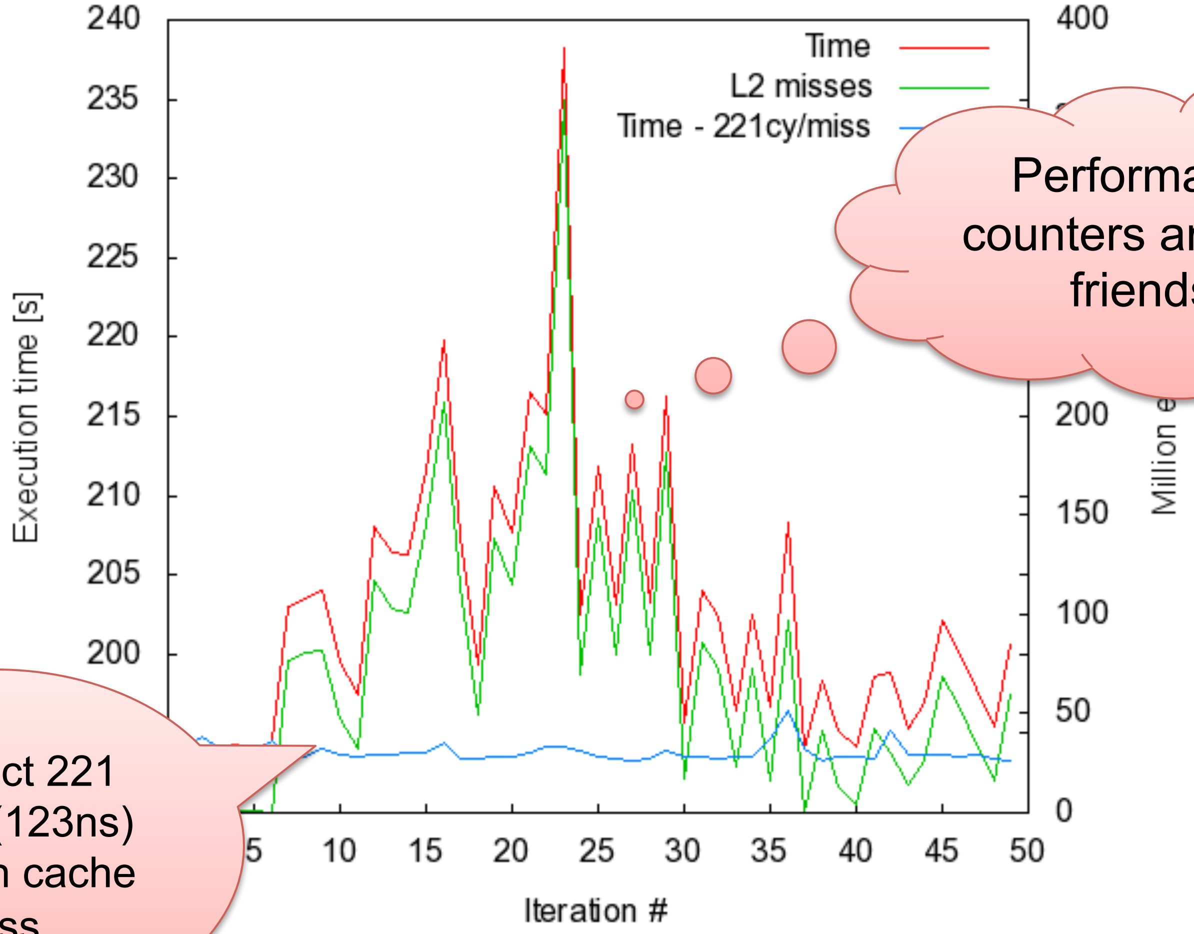
Platform:

- Dell Latitude D600
  - Pentium M @ 1.8GHz
  - 32KiB L1 cache, 8-way
  - 1MiB L2 cache, 8-way
  - DDR memory @ effective 266MHz
- Linux kernel version 2.6.24

Methodology:

- Multiple identical runs for statistics...

`twolf` on Linux: What's going on?



Performance counters are your friends!

Subtract 221 cycles (123ns) for each cache miss

`twolf` on Linux: Lessons?

- Pointer to problem was standard deviation
  - σ for "twolf" was much higher than normal for SPEC programs
- Standard deviation did not conform to mental model
  - Shows the value of verifying that model holds
  - Correcting model improved results dramatically
- Shows danger of assuming reproducibility without checking!

Conclusion: *Always* collect and analyse standard deviations!

How to Measure and Compare Performance

NICTA

Avoid incorrect conclusions from pathological cases

- Typical cases:
  - sequential access optimised by underlying hardware/disk controller...
  - potentially massive differences between sequentially up/down
    - pre-fetching by processor, disk cache
  - random access may be an unrealistic scenario that destroys performance
    - for file systems
  - powers of two may be particularly good or particularly bad for strides
    - often good for cache utilisation
      - minimise number of cache lines used
    - often bad for cache utilisation
      - maximise cache conflicts
  - similarly just-off powers (2^n-1, 2^n+1)
- What is "pathological" depends a **lot** on what you're measuring
  - e.g. caching in underlying hardware

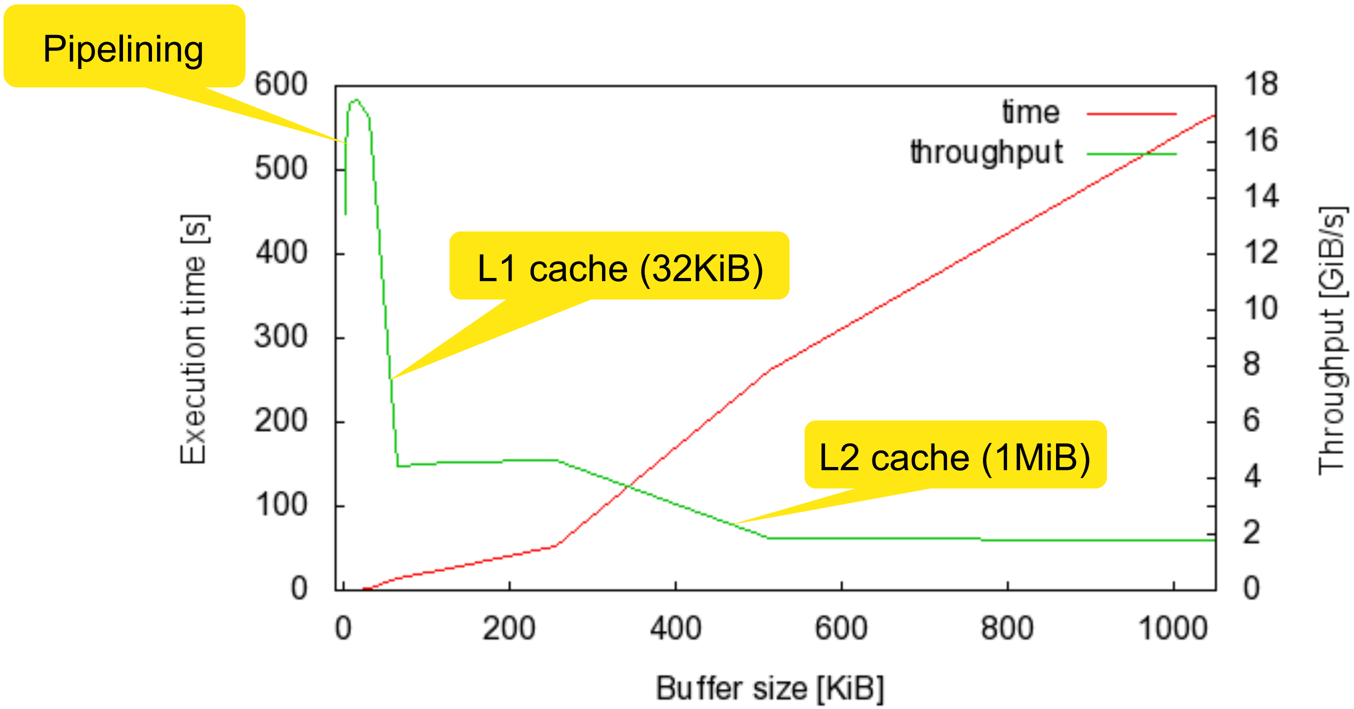
How to Measure and Compare Performance

NICTA

Use a model

- You need a (mental or explicit) model of the behaviour of your system
  - benchmarking should aim to support or disprove that model
  - need to think about this in selecting data, evaluating results
  - eg: I/O performance dependent on FS layout, caching in controller...
  - cache sizes (HW & SW caches)
  - buffer sizes vs cache size
- Should tell you the size of what to expect
  - you should understand that a 2ns cache miss penalty can't be right

Example: Memory Copy



How to Measure and Compare Performance

NICTA

Understand your results!

- Results you don't understand will almost certainly hide a problem
  - Never publish results you don't understand
    - chances are the reviewers understand them, and will reject the paper
    - maybe worse: someone at the conference does it
      - this will make you look like an idiot

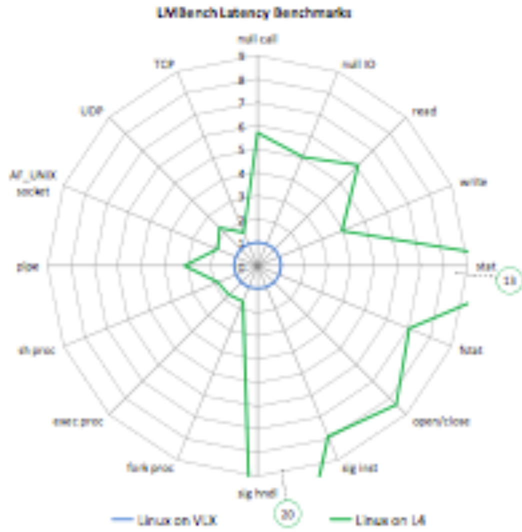
Of course, if this happens you ***are*** an idiot!

Relative vs Absolute Data

NICTA

From a real paper (IEEE CCNC'09):

- No data other than this figure
- No figure caption
- Only explanation in text:
  - "The L4 overhead compared to VLX ranges from a 2x to 20x factor depending on the Linux system call benchmark"
- No definition of "overhead factor"
- No native Linux data



Benchmarking crime: Relative numbers only

- Makes it impossible to check whether results make sense
- How hard did they try to get the competitor system to perform?
  - Eg, did they run it with default build parameters (debugging enabled)?

Benchmarking Ethics

- Do compare with published competitor data, but…
  - Ensure comparable setup
    - Same hardware (or *convincing* argument why it doesn't matter)
  - you may be looking at an aspect the competitor didn't focus on
    - eg: they designed for large NUMA, you optimise for embedded
- Be ultra-careful when benchmarking competitor's system yourself
  - Are you sure you're running the competitor system optimally?
    - you could have the system mis-configured (eg debugging enabled)
    - Do your results match their (published or else) data?
  - Make sure you understand exactly what is going on!
    - Eg use profiling/tracing to understand source of difference
    - Explain it!

Benchmarking crime: Unethical benchmarking of competitor

- Lack of care is unethical too!

What Is "Good"?

- Easy if there are established and published benchmarks
  - Eg your improved algorithm beats best published Linux data by *x*%
  - But are you sure that it doesn't lead to worse performance elsewhere?
    - important to run complete benchmark suites
    - think of everything that could be adversely effected, and *measure*!
- Tricky if no published standard
  - Can run competitor/incumbent
    - eg run lmbench, kernel compile etc on your modified Linux and standard Linux
    - but be *very careful* to avoid running the competitor sub-optimally!
  - Establish performance limits
    - ie compare against optimal scenario
    - micro-benchmarks or profiling can be highly valuable here!

Real-World Example: Virtualization Overhead

Good or bad?

- Symbian null-syscall microbenchmark:
  - native: 0.24μs, virtualized (on OKL4): 0.79μs
  - 230% overhead
- ARM11 processor runs at 368 MHz:
  - Native: 0.24μs = 93 cy
  - Virtualized: 0.79μs = 292 cy
  - Overhead: 0.55μs = 199 cy
  - Cache-miss penalty ≈ 20 cy
- Model:
  - native: 2 mode switches, 0 context switches, 1 x save+restore state
  - virtualized: 4 mode switches, 2 context switches, 3 x save+restore state

Expected overhead?

Performance Counters are Your Friends!

Good or bad?

Counter	Native	Virtualized	Difference
Branch miss-pred	1	1	0
D-cache miss	0	0	0
I-cache miss	0	1	1
D-µTLB miss	0	0	0
I-µTLB miss	0	0	0
Main-TLB miss	0	0	0
Instructions	**30**	**125**	**95**
D-stall cycles	0	27	27
I-stall cycles	0	45	45
Total Cycles	**93**	**292**	**199**

More of the Same...

First step: improve representation!

Benchmark	Native	Virtualized
Context switch [1/s]	615046	444504
Create/close [µs]	11	15
Suspend [10ns]	81	154

Further Analysis shows guest dis-&enables IRQs 22 times!

Second step: overheads in appropriate units!

Benchmark	Native	Virt.	Diff [µs]	Diff [cy]	# sysc	Cy/sysc
Context switch [µs]	1.63	2.25	0.62	230	1	230
Create/close [µs]	11	15	4	1472	2	736
Suspend [µs]	0.81	1.54	0.73	269	1	269

Yet Another One...

Good or bad?

NICTA

Benchmark	Native [µs]	Virt. [µs]	Overhead	Per tick
TDes16_Num0	1.2900	1.2936	0.28%	2.8 µs
TDes16_RadixHex1	0.7110	0.7129	0.27%	2.7 µs
TDes16_RadixDecimal2	1.2338	1.2373	0.28%	2.8 µs
TDes16_Num_RadixOctal3	0.6306	0.6324	0.28%	2.8 µs
TDes16_Num_RadixBinary4	1.0088	1.0116	0.27%	2.7 µs
TDesC16_Compare5	0.9621	0.9647	0.27%	2.7 µs
TDesC16_CompareF7	1.9392	1.9444	0.27%	2.7 µs
TdesC16_MatchF9	1.1060	1.1090	0.27%	2.7 µs

- Note: these are purely user-level operations!
  - What's going on?

Timer interrupt virtualization overhead!

Lessons Learned

- Ensure stable results
  - repeat for good statistics
  - investigate source of apparent randomness
- Have a model of what you expect
  - investigate if behaviour is different
  - unexplained effects are likely to indicate problems — don't ignore them!
- Tools are your friends
  - performance counters
  - simulators
  - traces
  - spreadsheets

Annotated list of benchmarking crimes: http://www.gernot-heiser.org/

Thank You!

mailto:gernot@nicta.com.au

Twitter: @GernotHeiser