B4M36ESW: Efficient software

Lecture 3: Benchmarking

Michal Sojka
`michal.sojka@cvut.cz`

CZECH TECHNICAL UNIVERSITY IN PRAGUE

March 4, 2019

Outline

Outline

Benchmark

Wikipedia defines benchmark as:

1. the act of running a computer program, a set of programs, or other operations, in order to **assess the relative performance** of an **object**, normally by running a number of standard tests and trials against it.
2. a benchmarking program itself (i.e. “XY is a free benchmark that tests your computer’s performance.”)

Object examples:

- Hardware
- Compiler
- Algorithm
- ...

Types of benchmarks:

- Micro-benchmarks (synthetic)
- Application benchmarks

Types of benchmark

1. Micro-benchmark
   - Evaluates very little part of an application
   - It is easy to determine source of speed-up/slow down
   - Typically, improvements in micro-benchmark do not imply improvements application performance
2. Application benchmarks
   - Evaluate performance of the whole applications
   - Performance is influenced by many read-world factors
   - For complex applications, it might difficult to determine the source of speed-up/slow-down

How to measure software performance?

- What to measure?
  - Execution time
  - Memory consumption
  - Energy
- How to measure?
  - Not as easy as it sounds
  - See the rest of the lecture

Outline

Measuring energy

- Connect power meter to your computer/board
- Use hardware-provided interfaces for power/energy measurement/control
  - These are more and more common these days

Example

Intel RAPL (Running Average Power Limit)

- Allows to monitor and/or limit power consumption of individual components
- Package domain, memory domain (DRAM)
- Interface via MSRs
- See Intel Software Developer's Manual: System Programming Guide

Outline

Measuring memory consumption

- Under modern OSes, measuring memory usage is surprisingly complex
- How programs consume memory?
  1. Program memory
     - Code, static data, heap, stack
     - Stack is allocated for each thread
  2. Operating system kernel memory
     - Allocated by the OS kernel on behalf of the program
     - network buffers, disk and file system caches, system objects (timers, semaphores, …)
     - Sometimes, it is not possible to account this memory to an individual process – e.g. network receive buffers.
  3. Shared libraries
     - How to account memory consumed by libraries shared by multiple programs?

Basics of Linux Memory Statistics

- Tools like `top` or `htop` report several memory statistics

VIRT
: Total amount of virtual memory reserved by the process. Not all this memory needs to be backed by physical memory. It does not include kernel memory.
  - Example: Allocate 1 GiB of virtual memory without allocating physical memory immediately.

```
mmap(NULL, 1ULL << 34, PROT_READ | PROT_WRITE,
     MAP_ANONYMOUS | MAP_SHARED, -1, 0);
```

RES
: Currently resident (physical) memory

SHR
: Memory shared with other processes (data, .so)

Outline

Outline

Measuring execution time

Timestamping

1. Use system calls
   - Linux: gettimeofday, clock_gettime(CLOCK_MONOTONIC)
   - Resolution: depends on available hardware (down to 1 ns), earlier it was a system tick period (1–10 ms)
   - Overhead – hundreds of CPU cycles (but see next slide)
2. Use hardware directly (e.g. timestamp counter)
   - TSC register on x86 (resolution 1 clock cycle, overhead few ($\approx$ 8) clock cycles
   - Similar registers on other architectures
   - Cons: Can be subject to CPU frequency scaling, TSC counters on different CPU cores/sockets may not be synchronized

   ```
   static inline uint64_t rdtsc() {
       uint64_t ret;
       asm volatile ( "rdtsc" : "=A" (ret) );
       return ret;
   }
   ```
3. Combine both: Virtual syscall

Virtual syscall for fast timestamping

- Reading TSC is fast, but HW/frequency/socket dependent
  - Problematic when two timestamps need to be subtracted
  - OS kernel knows everything about HW/frequency/socket but calling kernel has overhead
- Idea: OS kernel publishes enough information for user space to reliably convert TSC value to wall-clock time without calling the kernel
  - time_ns = rdtsc() * tsc_scale + tsc_offset
- Virtual Dynamic Shared Object – VDSO
  - Kernel memory mapped to process address space
  - Looks like shared library
  - Application can call ordinary functions from there
  - `cat /proc/$$/maps|grep vdso`
  - gettimeofday, clock_gettime are functions implemented in VDSO

Outline

Measuring execution time

- Execution time exhibits variations
- Influenced by many factors:
  - Hardware, input data, compiler, memory layout, measuring overhead, rest of the system, network load, ... you name it
  - Same factors can be controlled, others cannot
- Repeatability of measurements
- How to design benchmark experiments properly?
- How to measure *speedup*?

Example

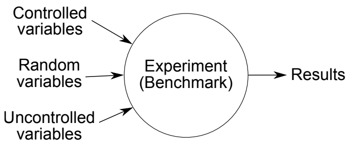
The Challenge of Reasonable Repetition

- Variations
- Measurements must be repeated
- We want to eliminate the influence of random (non-deterministic) factors
- Statistics

Controlled variables (e.g. compiler flags, hardware, algorithm changes) – we are interested how they impact the results

Random variables (e.g. hardware interrupts, OS scheduler) – we are interested in statistical properties of our results in face of these variables

Uncontrolled variables – mostly fixed, but can cause bias of the results



Benchmark goal

- Estimate (a confidence interval for) the mean of execution time of a given benchmark on one or more platforms.
- The mean is the property of the probability distribution of the random execution times
- We can only estimate the mean value from the measurements
- Confidence interval is important
  - CI of 95% $\Rightarrow$ in 95% of cases, the true mean will be within the interval.

Levels of repetition

- Results variance occurs typically at multiple levels, e.g.:
  - (re)compilation
  - execution
  - iteration inside a program
- Sound benchmarking methodology should evaluate all the levels with random variations

Next slides give answer to:

- How many times to repeat the experiment at each level?
  - As little times as possible to not waste time
  - As many times as possible to get reasonable confidence in results
- How to summarize the results?

Outline

Significance testing

Is it likely that two systems have different performance?

- Statistics can answer this with **Significance testing**
- However, this technique has problems, especially when used with results of computer benchmarks – see Kalibera's paper.
  - It is better to ask what is the speedup.
- Significance testing is implemented in the `ministat` tool (FreeBSD)

From ministat man page

The ministat command was written by Poul-Henning Kamp out of frustration over all the bogus benchmark claims made by people with no understanding of the importance of uncertainty and statistics.

ministat examples

```
+-----------------------------------------------------------------------------+
|     +++ + +   x       x    x                                                |
|   +++++ +++   x  x    x    x                                                |
|   +++++ +++   x  xx xxx x  x                                                |
|   +++++ +++   x xxxxxxx x  x x                                              |
|   +++++ +++++ xxxxxxxxx x xx x                           +                 x|
|             |________MA_________|                                          |
||______M_A_______|                                                         |
+-----------------------------------------------------------------------------+
    N           Min           Max        Median           Avg        Stddev
x  40         88.92       122.527        92.594      93.34845     5.3399441
+  40        82.313       112.625         84.52     85.447325     4.6810848
Difference at 95.0% confidence
        -7.90112 +/- 2.2355
        -8.46412% +/- 2.39479%
        (Student's t, pooled s = 5.02133)
Difference at 99.5% confidence
        -7.90112 +/- 3.59073
        -8.46412% +/- 3.84658%
```

Too little data with too similar distribution:

```
+---------------------------------------------------------------------------+
|                                   +  +                                    |
|+ x       +   +    +  x            +  +    * x x        x          x  xxx +|
|                          |______________________A______M_______________|  |
|          |____________________A___M________________|                      |
+---------------------------------------------------------------------------+
    N           Min           Max        Median           Avg        Stddev
x  10       151.527       155.963       154.936      154.5278     1.4673007
+  10       151.371       156.096       153.618      153.3248     1.3398755
No difference proven at 95.0% confidence
```

Confidence interval

- We want to estimate the mean of a probability distribution
- We only have a limited set of r measurements and know almost nothing about the distribution
- We calculate the average value $\bar{Y}$ from the measurements
- How is the average different from the true mean value?
- $\bar{Y} \pm \frac{S_Y}{\sqrt{r}} q_{t(r-1)}(1 - \frac{\alpha}{2})$, where
  - $q_{t(r-1)}(1 - \frac{\alpha}{2})$ is $(1 - \frac{\alpha}{2})$-quantile of the Student's t-distribution with $r - 1$ degrees of freedom.
  - α is significance level (e.g. 5%)
- We say: Execution time of our benchmark is 25.4 ± 3.2 ms with 95% confidence.
- This means that the true mean is somewhere between 22.2 and 28.6 with probability of 95%.

- `https://stackoverflow.com/questions/15033511/compute-a-confidence-interval-from-sample-data`

Visual tests

- Calculate and visualize confidence intervals.
- Do the two confidence intervals overlap?
- No $\Rightarrow$ different performance is likely
- Yes $\Rightarrow$ more statistics needed
- Hard to estimate speedup and its confidence interval
- Note: `ministat` does not calculate confidence intervals, but standard deviations, i.e. S_Y

Recommendation

Analysis of results should be statistically rigorous and in particular should quantify any variation. Report performance changes with effect size confidence intervals.

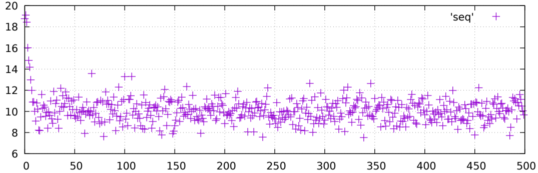
Outline

Repeating iterations

- Iteration = one execution of a loop body
- We are interested in *steady state performance*
- Initialization phase
  - First few iterations typically include the initialization overheads
  - Warming up caches, teaching branch predictor, memory allocations
- Independent state
  - Ideally, measurements should be *independent, identically distributed* (i.i.d.)
  - Independent: measurement does not depend on any a previous measurement
  - Independent $\Rightarrow$ initialized

When a benchmark reaches independent state?

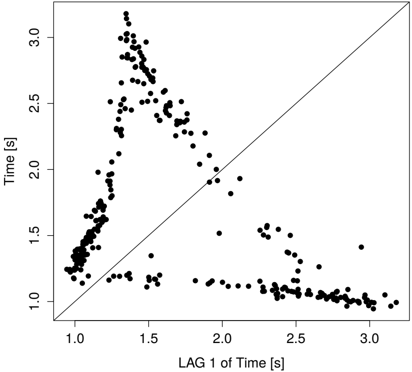
- Manual inspection of graphs from measured data
  1. run-sequence plot ⇒ easy identification of initialization phase ⇒ strip

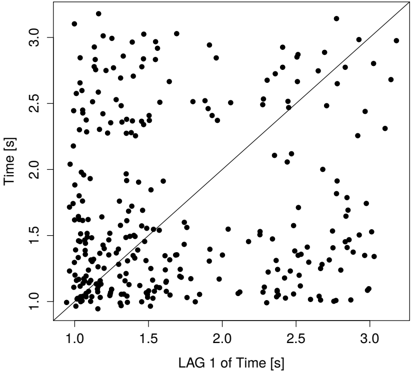


  2. Independence assessment – plot the following plots on original and randomly reordered sequence
     - lag plot (for several lags – e.g. 1–4)
     - auto-correlation function
  3. Any visible pattern suggests the measurements are not independent

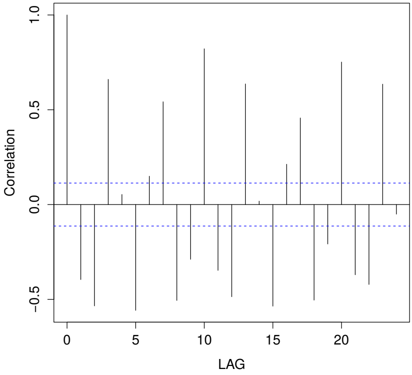
Lag plot

Dependency of a measured values on the previously measured value.

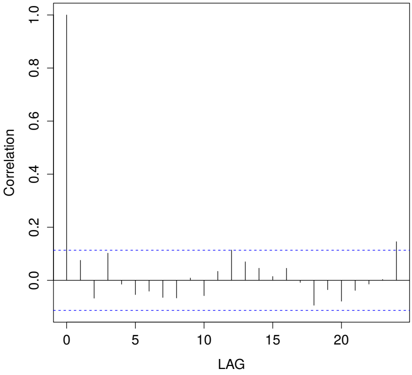




Auto-correlation function



dependent



independent

Recommendations

Use this manual procedure just once to find how many iterations each benchmark, VM and platform combination requires to reach an independent state.

If a benchmark does not reach an independent state in a reasonable time, take the same iteration from each run.

Outline

Repeating executions

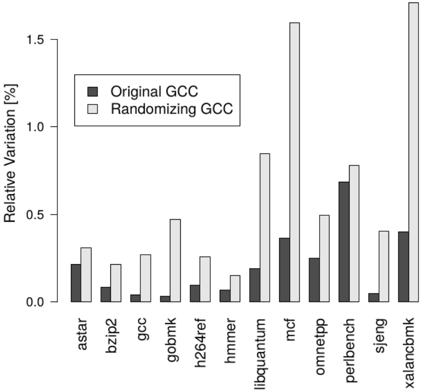
- Running a benchmark program multiple times
  - Effect of JIT compiler etc.
- Example: Variance in % of different benchmarks from DaCapo/OpenJDK benchmark suite

	bloat6	*eclipse9*	*lusearch9*	*tradebeans9*	*tradesoap9*	*xalan6*	*xalan9*
Iteration	14.1	0.8	3.3	1.5	0.8	7.0	3.5
Execution	3.7	0.4	30.3	0.4	0.4	9.1	1.0

- What if different executions exhibit higher variance than iterations? (see lusearch9)
- Determine initialized and independent state for executions as for iterations.

Repeating compilation

- Sometimes even a compiler can influences the benchmark results.
- Experiment: Code layout generated by the compiler: original vs. randomized



- Why code layout makes a difference?
- If you cannot control the factor, make it random!

Outline

Multi-level repetition

- We have to repeat the experiments to narrow confidence interval
- If the variance occurs at higher levels (execution, compilation), we need to repeat at least at that level.
- Repeating at lower level may be cheaper (no execution overhead, compilation overhead, etc.)
  - Time can be saved by repeating at lower levels.
- How to find required number of repetitions at each level to reach given confidence interval?
  - Can be formulated mathematically.
  - If you repeat too little, you have wide confidence intervals.
  - If you repeat too much, you waste your time with running unnecessary experiments.

Notation

- Levels
  - Lowest level (iteration) = 1
  - Highest level (e.g. compilation) = n
- *Initial experiment*
  - bold letters
  - $\mathbf{r_1}, \mathbf{c_1}$
- *Real experiment*
  - normal letters
  - r_1, c_1

Initial experiment

Goal is to find the required number of iterations at each level.

- Select number of repetitions (exclusive of warm-up) $\mathbf{r_1}, \mathbf{r_2}$, ... to be arbitrary but sufficient value, say 20.
- Gather the cost of repetition at each level (time added exclusively by that level, e.g. compile time)
  - $\mathbf{c_1}$ iteration duration
  - $\mathbf{c_2}$ time execute benchmark up to independent state
  - $\mathbf{c_3}$ compilation time
- Measurement times: $\mathbf{Y_{j_n \dots j_1}}, \quad j_1 = 1 \dots r_1, j_2 = 1 \dots r_2, \dots$
- Calculate arithmetic means for different levels:
  $\bar{\mathbf{Y}}_{j_n \bullet \cdots \bullet}$

Variance estimators

- After initial experiments, we will calculate n unbiased variance estimators $\mathbf{T_1^2}, \ldots, \mathbf{T_n^2}$
- They describe how much each level contributes independently to variability in the results
- Start with calculating $\mathbf{S_i^2}$ – biased estimator of the variance at each level $i, 1 \leq i \leq n$:

$$\mathbf{S_i^2} = \frac{1}{\prod_{k=i+1}^{n} \mathbf{r_k}} \frac{1}{\mathbf{r_i} - 1} \sum_{j_n=1}^{r_n} \cdots \sum_{j_i=1}^{r_i} \left(\bar{\mathbf{Y}}_{j_n \ldots j_i \bullet \cdots \bullet} - \bar{\mathbf{Y}}_{j_n \ldots j_{i+1} \bullet \cdots \bullet}\right)^2$$

- Then obtain T_i^2:

$$\begin{aligned} T_1^2 &= \mathbf{S}_1^2 \\ \forall i, 1 < i \leq n, T_i^2 &= \mathbf{S}_i^2 - \frac{\mathbf{S}_{i-1}^2}{\mathbf{r_{i-1}}} \end{aligned}$$

- If $T_i^2 \leq 0$, this level induces little variation and repetitions can be skipped.

Real Experiment: Confidence Interval

- Optimum number of repetitions at different levels $r_1, \ldots, r_{n-1}$ can be calculated as:

$$\forall i, 1 \leq i < n, \quad r_i = \left\lceil \sqrt{\frac{c_{i+1}}{c_i} \frac{T_i^2}{T_{i+1}^2}} \right\rceil$$

- Then recalculate: S_n^2 and $\bar{Y}_{j_n \bullet \ldots \bullet}$ as before but with data from real experiment.
- Asymptotic confidence interval with confidence $(1-\alpha)$ is:

$$\bar{Y} \pm t_{1-\frac{\alpha}{2},\nu} \sqrt{\frac{S_n^2}{r_n}}$$

where $t_{1-\frac{\alpha}{2},\nu}$ is $(1-\frac{\alpha}{2})$-quantile of the t-distribution with $\nu = r_n - 1$ degrees of freedom.

Recommendation

For each benchmark/VM/platform, conduct a dimensioning experiment to establish the optimal repetition counts for each but the top level of the real experiment. Re-dimension only if the benchmark/VM/platform changes.

Outline

Measuring speedup

- Speedup: "With my optimization, the program runs 10% faster."
- Speedup is a ratio of two execution times (random variables)
- What is the speedup confidence interval?
  E.g. 10%±2% faster with confidence of 99%
- How many times to repeat the speedup experiments?

Speedup confidence interval

- $\bar{Y}$ – old system execution time (average of measured times)
- $\bar{Y}'$ – new system execution time
- Speedup: $\bar{Y}' / \bar{Y}$
- Speedup confidence interval:

$$\frac{\bar{Y} \cdot \bar{Y}' \pm \sqrt{(\bar{Y} \cdot \bar{Y}')^2 - (\bar{Y}^2 - h^2)(\bar{Y}'^2 - h'^2)}}{\bar{Y}^2 - h^2}$$

$$h = \sqrt{t^2_{\frac{\alpha}{2},\nu} \frac{S_n^2}{r_n}} \quad h' = \sqrt{t^2_{\frac{\alpha}{2},\nu} \frac{S_n'^2}{r_n}}$$

Repetition count

- Relation of confidence interval of the speedup to confidence interval on individual measurements:

$$\mathbf{e_s} \approx \frac{\bar{Y}'}{\bar{Y}} \sqrt{e^2 + e'^2}$$

- $\mathbf{e_s}$, e, e' relative half-width of the speedup resp. old resp. new confidence interval, i.e. $e = h/\bar{Y}$

- Old system: 10±1 s, e=0.1 (10%)
- New system: 9±0.9 s, e'=0.1
- Speedup: ≈0.9±0.13
- Outcome: Speedup can be 1, i.e. no speedup!

Recommendation

Always provide effect size confidence intervals for results. Either for single systems or for speedups.

References

- Kalibera, T. and Jones, R. E. (2013) **Rigorous Benchmarking in Reasonable Time**. In: ACM SIGPLAN International Symposium on Memory Management (ISMM 2013), 20–12 June, 2013, Seattle, Washington, USA. `http://kar.kent.ac.uk/33611/`
- `https://www.cse.unsw.edu.au/~gernot/benchmarking-crimes.html`
- Erik van der Kouwe et al. (2018) **Benchmarking Crimes: An Emerging Threat in Systems Security**, `https://arxiv.org/abs/1801.02381`