A practical course on

Advanced systems programming in C/Rust

Maurice Bailleu

Today’s topic!

Performance

Outline

- What do we mean by performance?
- Why does performance matter?
- Why does my code not perform as expected?
- How to fix performance problems?
- Profilers
- Linux Perf
- Common performance optimization pitfalls

Performance?

- Throughput (Op/s, B/s, Tx/s)
  - DB, KV, network, storage
- Latency (s, cycles)
  - DB, KV, network, storage, HFT
- Memory (B, pages)
  - Embedded systems
- Energy Usage (V, A, W, W/op)
  - Embedded systems, laptops, smartphones, tablets, etc.
- Binary size (B)
  - Embedded systems, loading from remote
- Compiling time (s)
  - Testing, development

Performance?

- Different metrics often require tradeoffs
    - Multithreading increase throughput but makes latency worse
    - Less memory usage means less caching more recalculations -> worse throughput
    - Less compile time means less compile time calculation -> more runtime calculations

Does Performance matter?

> “Performance doesn't matter, unless you can prove that it does.”[1]

- Application performance determines hardware requirements
- Performance can be a feature: HFT, UI
- Systems should avoid to be the limiting factor for an application

[1] Mathew Jones: https://exceptionnotfound.net/performance-doesnt-matter-unless-you-can-prove-that-it-does/

Why does my code not perform as expected?

> “The field of performance is littered with myth and bogus folklore. Modern hardware and optimizers defy naive assumptions; even experts are regularly surprised.”[2]

Hardware:

- CPU Caches
  - Cache size
  - Caching protocol
  - False sharing
  - Cache misses
  - Prefetcher
  - Page misses
- Branch predictor
  - Pipes
- Interrupts
- Simultaneous multithreading
- Dynamic frequency scaling
- NUMA

Software:

- Software Caches
  - Page cache
- Scheduler
  - Energy aware scheduling
  - Real time scheduling
  - Overcommitment
  - Process priority
- Context switch
- Swapping
- Compiler/Linker optimizations
- ASLR
- CPU affinity
- CPU governor

[2] Bjarne Stroustrup and Herb Sutter: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#per6-dont-make-claims-about-performance-without-measurements

How to fix performance problems?

1. Establish that you have a performance problem
2. Do measurements!!
3. Do more measurements!!
4. Be sure you understand the problem
5. Fix the problem
6. Repeat 2. and 3. to be sure that you actually fixed it
7. Goto 1.

Remember: Performance optimization is a diminishing returns situation

How to get consistent results?

1. Disable turboboost

```
Intel: echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
AMD:   echo 0 > /sys/devices/system/cpu/cpufreq/boost
```

2. Disable SMT

```
echo 0 > /sys/devices/system/cpu/cpuX/online
```

3. Set scaling_governor to performance

```
echo performance > /sys/devices/system/cpu/cpuX/cpufreq/scaling_governor
```

4. Set cpu affinity

```
taskset -c X sleep 10
```

5. Set process priority

```
nice -n -N sleep 10
```

6. Drop file system cache

```
echo 3 | sudo tee /proc/sys/vm/drop_caches && sync
```

7. Disable ASLR

```
setarch -R sleep 10
```

8. Measure multiple runs

Measurements (1/4)

HW Counter:

Architecture depending counters

Count specific events

e.g . CPU cycles, L1 Cache misses, Energy

Tools:

- **perf**
- Intel VTune
- AMD µPerf

Measurements (2/4)

Syscall wrapper:

Libraries intercepting syscalls

Counting events and collecting stack traces

Mainly used for memory profiling

Tools:

- Gperftools
- Heaptrack

Measurements (3/4)

VM based:

Puts application in a VM

Can collect very detailed information

Very slow

Tools:

- Valgrind

Measurements (4/4)

Instrumentation:

Inject code to function calls

Code takes necessary measurements

Either binary or source code changes

Tools:

- Orbit
- gprof
- Google Benchmark/Quick Bench

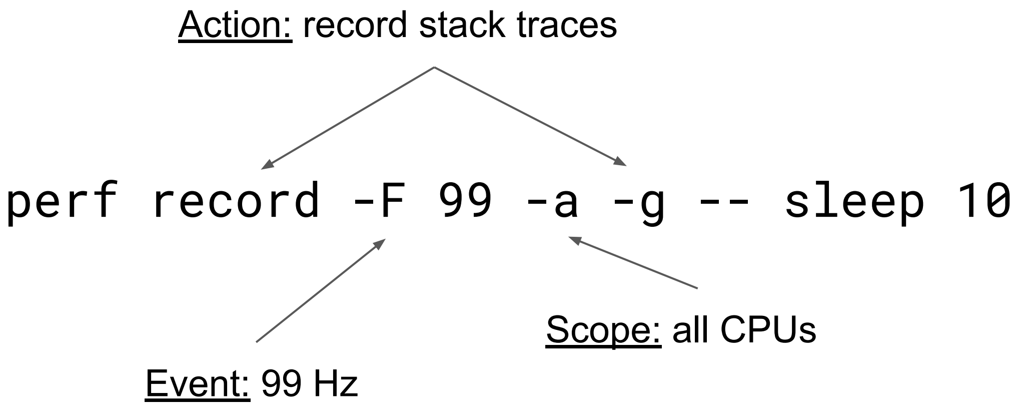
Linux Perf

- The official Linux profiler
- Supports many profiling/tracing features:
  - CPU Performance Monitoring Counters (PMCs)
  - Statically defined tracepoints
  - User and kernel dynamic tracing
  - Kernel line and local variable tracing
  - Stack tracing, libunwind
  - Code annotation

Perf basic workflow

1. `list ->` find events
2. `stat ->` count them
3. `record ->` write event data to file
4. `report ->` browse summary
5. `script ->` event dump for post processing

Perf stat/record format



Action: record stack traces

```
perf record -F 99 -a -g -- sleep 10
```

Event: 99 Hz

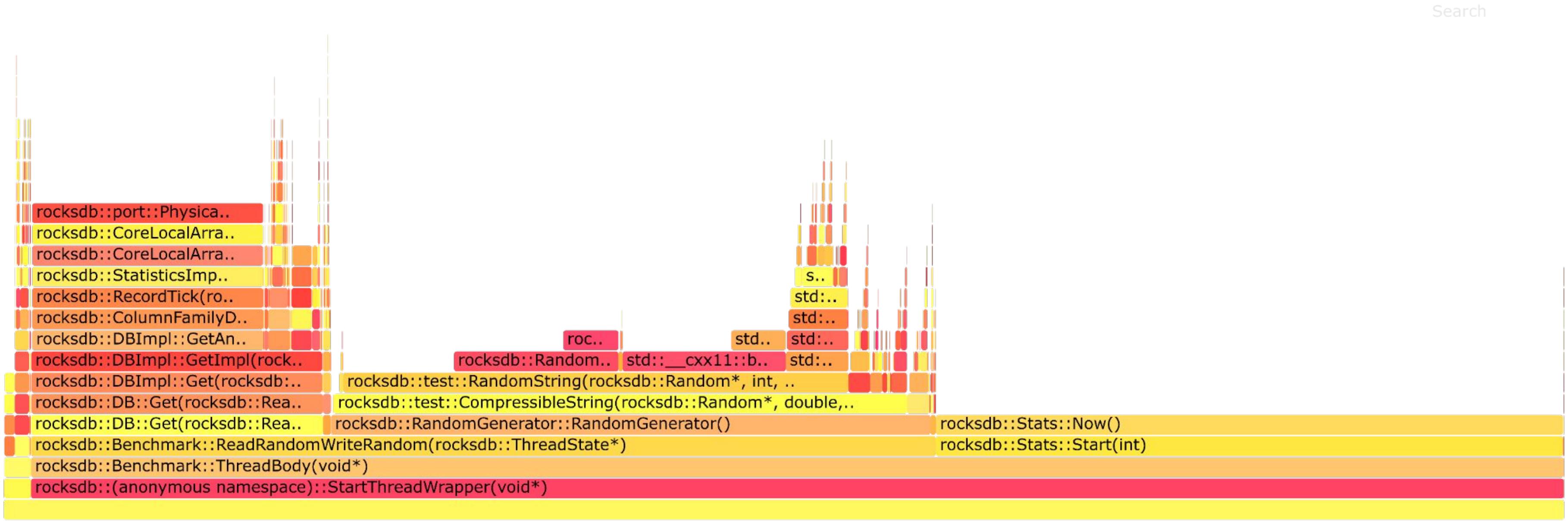
Scope: all CPUs

Perf Report

- Summarizes a report
- Can still be very verbose

Flamegraph

```
perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```



Common pitfalls

- Making wrong predictions/Random changes
  - See previous slide
- Not knowing your data
  - AoS vs SoA
- Overoptimizing/Premature optimization
- Bad benchmarks