How to run stable benchmarks
In 2014, int+int optimization proposed: 14 patches, many authors

Is it faster? Is it worth it?

*The Grand Unified Python Benchmark Suite*

Sometimes slower, sometimes faster

Unreliable and unstable benchmarks?
Goal

- Unstable benchmarks lead to bad decisions
- Patch makes Python faster, slower or... is not significant?
- Need **reproducible** benchmark results on the same computer
WTF meter

The only valid measurement of code quality: WTFs/minute
System & noisy apps

- CPU-bound microbenchmark:
  python3 -m timeit 'sum(range(10**7))'
- Idle system: **229 ms**
- Busy system: **372 ms** (1.6x slower, +62%)
  python3 -c 'while True: pass'
- **WTF?**
Isolated CPUs

- System and applications share same CPUs, memory and storage
- Linux kernel `isolcpus=3` don’t schedule processes on CPU 3
- Pin a process to a CPU: `taskset -c 3 python3 script.py`
- Idle system: **229 ms**
- Busy system, isolated CPU: **230 ms!**
Enter GRUB, modify Linux command line to add: **isolcpus=3**

**nohz_full=3**: if only 0 or 1 process running on CPU 3, disable all interruptions on this CPU *(WARNING: see later!)*

**rcu_nocbs=3**: don’t run kernel code on CPU 3
April 2016, experimental change to avoid temporary tuple to call functions

- Builtin functions **20-50% faster**!
- **But** some **slower** benchmarks
- 20,000 lines patch reduced to adding two unused functions... still slower. **WTF??**
Deadcode

- Reference:
  1201.0 ms +/- 0.2 ms

- Add 2 unused functions:
  1273.0 ms +/- 1.8 ms (slower!)

- Add 1 empty unused function:
  1169.6 ms +/- 0.2 ms (faster!)
Deadcode

I SEE DEAD PEOPLE

YOU MUST BE NEW HERE...
Code placement

- Root cause: *code placement*
- Memory layout and function addresses impact CPU cache usage
- It’s very hard to get the best placement and so reproducible benchmarks
70% slower!
Profiled Guided Optimizations (PGO):
.
configure --with-optimizations

(1) Compile with instrumentation
(2) Run the test suite to collect statistics on branches and code paths (hot code)
(3) Use statistics to recompile Python
Python hash function

Hash function randomized by default.

- PYTHONHASHSEED=1: 198 ms
- PYTHONHASHSEED=3: 207 ms (slower!)
- PYTHONHASHSEED=4: 187 ms (faster!)

**WTF???

Different number of hash collisions
Performance also impacted by:

- Unused **environment variables**
- Current working **directory**
- Unused **command line arguments**
- etc.
WTF????

WHAT THE FUCK

...IS THIS SHIT?!
First, I disabled Address Space Layout Randomization (ASLR), randomizing Python hash function, etc.

Lost cause: too many factors impact randomly performances

timeit uses minimum: wrong!

Solution to random noise: compute average of multiple samples
New Python module: perf

- Spawn multiple processes
- Compute average and standard deviation
New drama

Everything was fine for days, until... the new drama

Suddenly, a benchmark became 20% faster

WHAT-THE-FUCK ??????
Modern Intel CPUs

Since 2005, the frequency of Intel CPUs changes anytime for various reasons:

- Workload
- CPU temperature
- and... the number of active cores
Turbo Button?
Turbo Boost

- My laptop: 4 cores (HyperThreading)
  - 2-4 active cores: 3.4 GHz
  - 1 active core: **3.6 GHz (+5%)**

```
sudo cpupower frequency-info
```

Disable Turbo Boost in BIOS, or write 1 into:
```
/sys/devices/system/cpu/intel_pstate/no_turbo
```
I ran different benchmarks for days and even for **weeks**

Everything was **SUPER STABLE**
Stable benchmarks!
Nightmare never ends

But...

... one friday afternoon after I closed my GNOME session

... the benchmark became 2.0x faster

WTF?????? (sorry, this one should really be the last one... right?)
Nightmare never ends
Let me recall

- System and noisy apps: `isolcpus`
- Deadcode, code placement: `PGO`
- ASLR, Python hash function, env vars, cmdline, ...: `average + std dev`
- Turbo Boost: `disable TB`
CPU temperature?

NOPE
GODZILLA DELIVERS A NICE TALL
GLASS OF NOPE

NOPE
NOHZ_FULL and Pstate

- `nohz_full=3 (...)` disables all interruptions
- `intel_pstate` and `intel_idle` CPU drivers registers a scheduler callback
- No interruption means no scheduler interruption (LOC in `/proc/interrupts`)
- CPU 3 `Pstate` doesn’t depend on isolated CPUs workload, but other CPUs workload
NOHZ_FULL and Pstate

- intel_pstate and intel_idle drivers maintainer never tried NOHZ_FULL
- Linux real time (RT) developers: « it’s not a bug, it’s a feature! »
  - Use a fixed CPU frequency
  - or: don’t use NOHZ_FULL
Tune system to run benchmarks:
```
python3 -m perf system tune
```

Stop using `timeit`!
```
python3 -m timeit STMT
```
⇒
```
python3 -m perf timeit STMT
```

Use `perf` and its documentation!
```
http://perf.rtfd.io/
```
Before

![Graph showing performance over time for a specific commit](graph.png)

- **Date**: Oct 19
- **Result**: 0.0290
- **Std Dev**: 0.0008
- **Commit**: 1ce50f7027

The graph plots the performance of the `call_method` function over time, with the x-axis representing commit dates from February to December, and the y-axis showing the time in seconds (less is better). The graph shows a baseline performance from February to June, a spike in October, and a return to baseline by December.
After (with PGO)
Telco benchmark

![Graph showing time-series data for Telco benchmark](image-url)
Questions?

http://perf.rtfd.io/

https://github.com/python/performance/

https://speed.python.org/

Victor Stinner
victor.stinner@gmail.com
Perf features

- Collect **metadata**: CPU speed, uptime, Python version, kernel task#, ...
- Compare two results, check if **significant**
- **Stats**: min/max, mean/median, sample#, ...
- Dump all timings including **warmup**
- Check stability, render **histogram**, ...