Profiling python code

Zbigniew Jędrzejewski-Szmek

zbyszek@in.waw.pl

Stuttgart, 2020.01.21

Lecture links

`https://github.com/IMS-workshop/profiling-lecture`

What is optimization?

"Program optimization or software optimization is the process of modifying a software system to make some aspect of it work more efficiently or use fewer resources."

Wikipedia

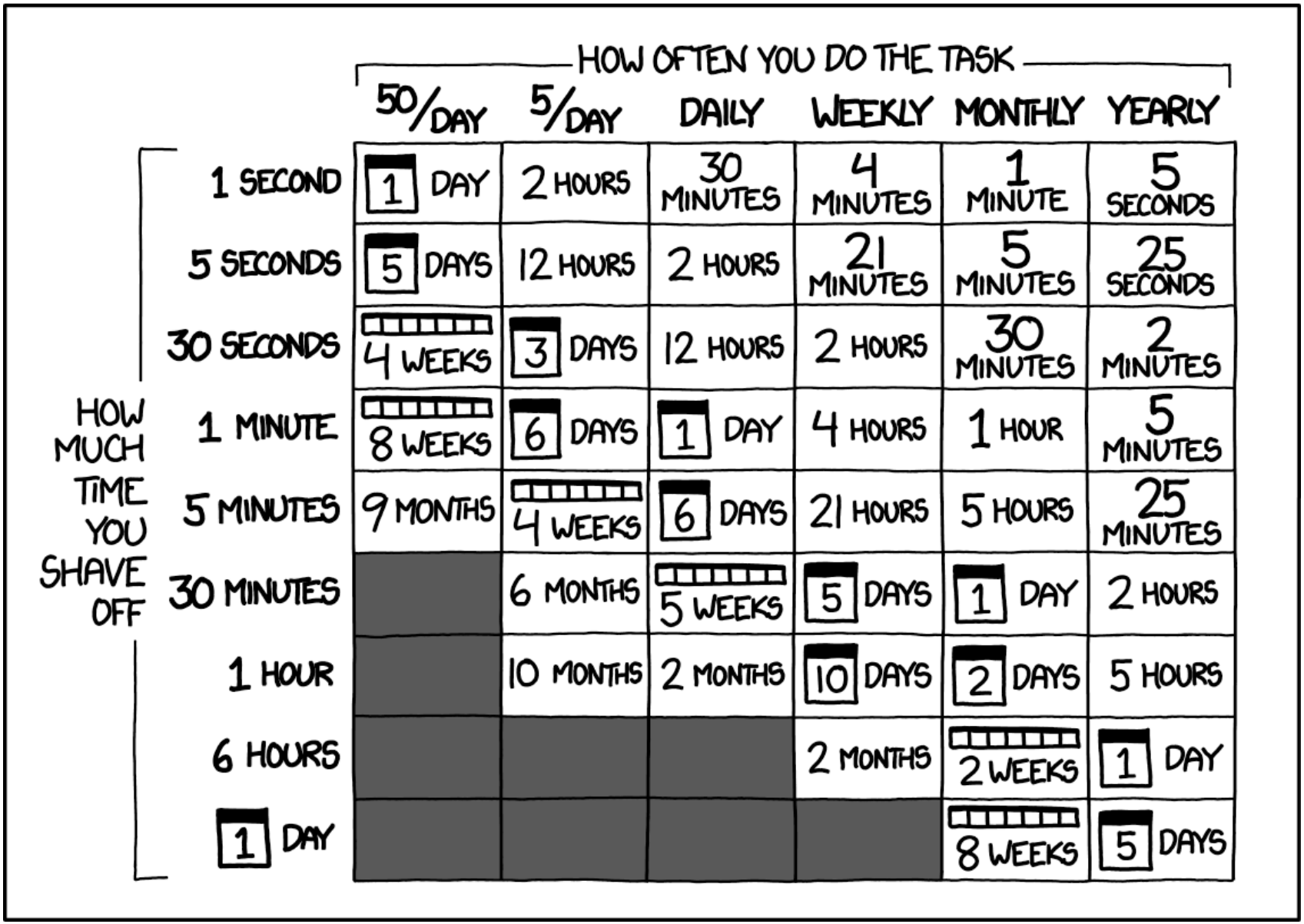
Premature optimization

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%."

Donald Knuth

HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?

(ACROSS FIVE YEARS)



xkcd.com/1205

"Optimization is the root of all evil."

Donald Knuth

Optimization workflow

1. **Make it work**: write the code in a simple legible way
2. **Make it work reliably**: write automated test cases, make really sure that your algorithm is right and that if you break it, the tests will capture the breakage.
3. Optimize the code by profiling simple use-cases to **find the bottlenecks** and speed up these bottlenecks, finding a better algorithm, or implementation. Keep in mind the tradeoff between simplicity and reliability and speed of exection of the code.

%timeit

demo

Exercise

Please see `even-odds/exercise.txt`.

Exercise!

Make factorial faster by using a cache (e.g. a dict).
What about `functools.lru_cache`?

`time` shell builtin

On *nix systems, the command time gives a quick way of measuring time:

```
$ time find /usr -name idontexist
1.91s user 4.32s system 36% cpu 17.279 total

$ time sleep 5
0.00s user 0.00s system 0% cpu 5.005 total
```

"total" or "real" is wall clock time
"user" is CPU time executing the script
"system" or "sys" is CPU time spent in system calls

Exercise

Please see `pyc-files/exercise.txt`.

cProfile

standard Python module to profile an entire application (profile is an old, slow profiling module)

Running the profiler from command line:

```
$ python -m cProfile -s cumtime myscript.py
```

Sorting options:

- tottime : time spent in function only
- cumtime : time spent in function and sub-calls
- calls : number of calls

cProfile

Save results to disk:

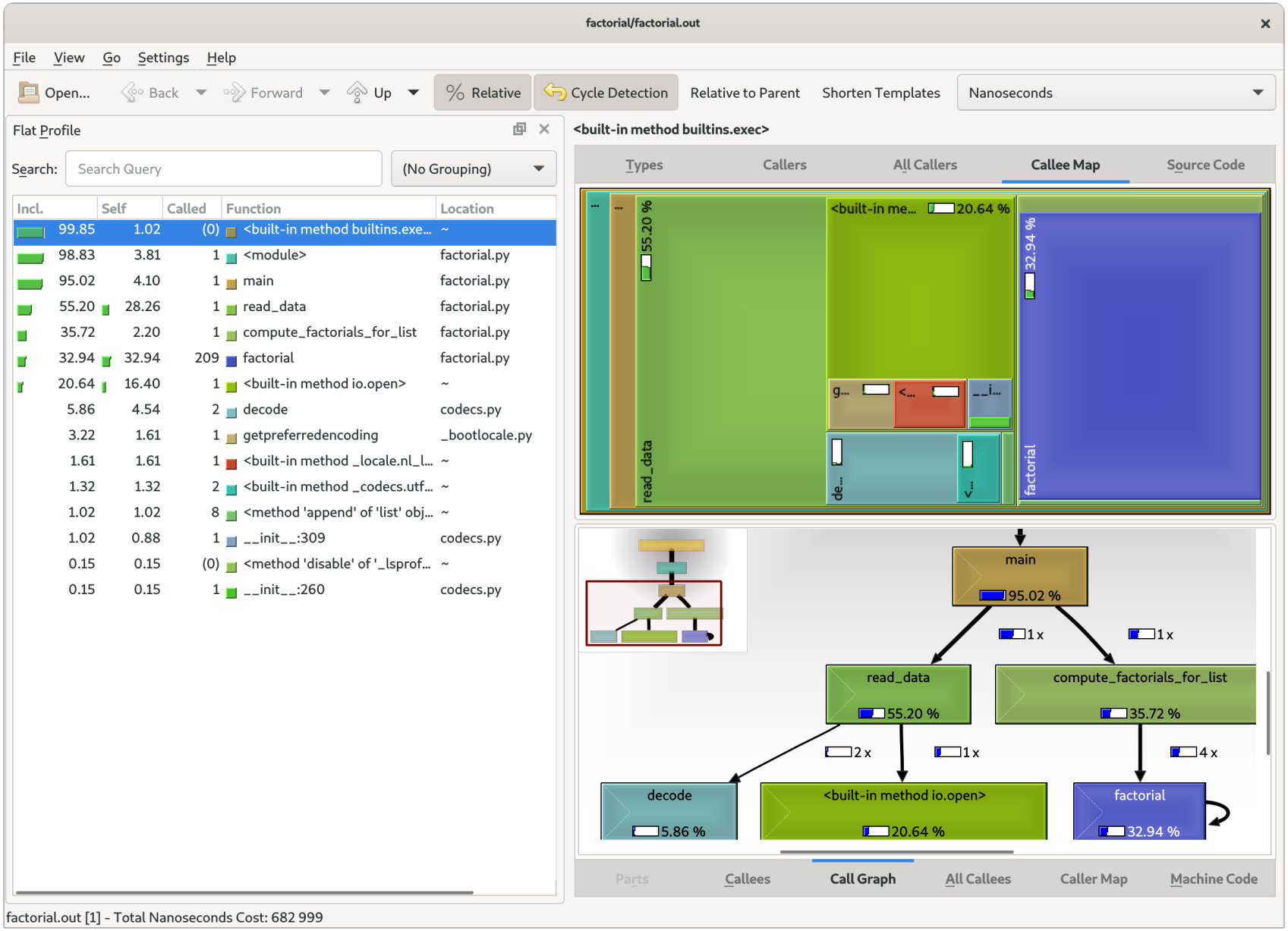
```
$ python -m cProfile -o filename.prof myscript.py
```

Explore:

```
$ python -m pstats filename.prof
```

```
filename.prof% help

stats [n | regexp]: print statistics
sort [cumulative, time, ...] : change sort order
callers [n | regexp]: show callers of functions
callees [n | regexp]: show callees of functions
...
```


factorial/factorial.out
File View Go Settings Help
Open... Back Forward Up Relative Cycle Detection Relative to Parent Shorten Templates Nanoseconds
Flat Profile
Search: Search Query (No Grouping)
Incl. Self Called Function Location
99.85 1.02 (0) <built-in method builtins.exe... ~
98.83 3.81 1 <module> factorial.py
95.02 4.10 1 main factorial.py
55.20 28.26 1 read_data factorial.py
35.72 2.20 1 compute_factorials_for_list factorial.py
32.94 32.94 209 factorial factorial.py
20.64 16.40 1 <built-in method io.open> ~
5.86 4.54 2 decode codecs.py
3.22 1.61 1 getpreferredencoding _bootlocale.py
1.61 1.61 1 <built-in method _locale.nl_l... ~
1.32 1.32 2 <built-in method _codecs.utf... ~
1.02 1.02 8 <method 'append' of 'list' obj... ~
1.02 0.88 1 __init__:309 codecs.py
0.15 0.15 (0) <method 'disable' of '_lsprof... ~
0.15 0.15 1 __init__:260 codecs.py
<built-in method builtins.exec>
Types Callers All Callers Callee Map Source Code
55.20 %
read_data
<built-in me...
20.64 %
32.94 %
factorial
main
95.02 %
1 x
1 x
read_data
55.20 %
compute_factorials_for_list
35.72 %
2 x
1 x
4 x
decode
5.86 %
<built-in method io.open>
20.64 %
factorial
32.94 %
Parts Callees Call Graph All Callees Caller Map Machine Code
factorial.out [1] - Total Nanoseconds Cost: 682 999

kernprof

```
$ sudo dnf install 'python3dist(line-profiler)'
(or)
$ pip3 install --user line_profiler
```

```
@profile
def slow_function(a, b, **kwargs):
    ...
```

```
$ kernprof -l script_to_profile.py
$ python3 -m line_profiler script_to_profile.py.lprof
```

pysnooper

```
$ sudo dnf install 'python3dist(pysnooper)'
(or)
$ pip3 install --user pysnooper
```

```
import pysnooper

@pysnooper.snoop()
def funcition_to_trace(n):
    ...
```

```
$ python3 script_to_trace.py |& less
```

Thanks to ...

- Pietro Berkes
  `https://github.com/ASPP/testing_debugging_profiling`
- Nelle Veroquaux
  `https://github.com/ASPP/`
  `2019-camerino-profiling-cython-numba`

Toolbox

https://docs.python.org/3/library/profile.html
https://kcachegrind.github.io/
https://github.com/rkern/line_profiler
https://github.com/cool-RR/PySnooper
https://pypi.python.org/pypi/memory_profiler
http://www.camillescott.org/2013/12/06/yep/