Where is the bottleneck

Manuel Miranda

Software Engineer

Contents

- Strategy: What/How to start
- Basic OS tools
- Resources tools
- Advanced

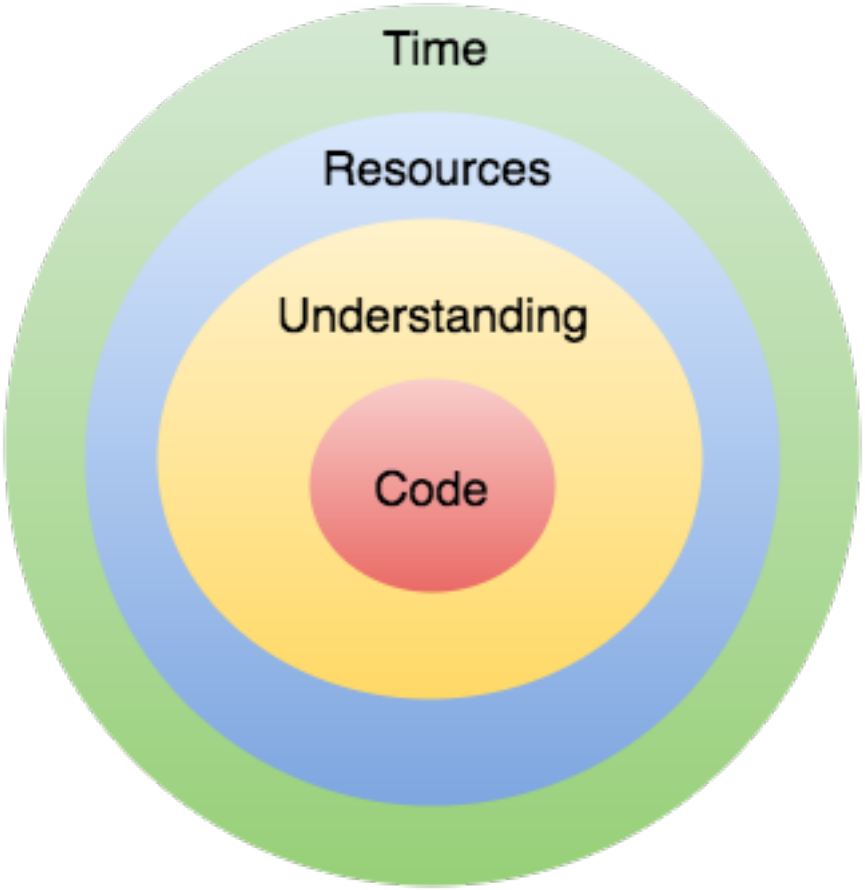
Strategy: Considerations

When you want to improve your program performance, ask yourself the following:

- **Focus:** What do you exactly want to accomplish?
- **Cost:** Does the time you will spend improving performance worth it?
- **Code knowledge:** Do you control all the code?
- **Context awareness:** Can external resources affect you?
- **Local context:** Are your tests reproducing exactly the production execution?

It's really important to think about this points before starting. They can save you lots of time!

Strategy: Start



OS tools

- time
- htop
- ntop
- lsof
- vmstat

MEMORY_PROFILER

https://github.com/fabianp/memory_profiler

376 Commits

Last commit: 29th Jun

792 Stars

Resources: memory_profiler

Memory profiler is pretty useful, it allows us to have a per function view of memory consumed. Easy to get a fast picture of how your program memory evolves.

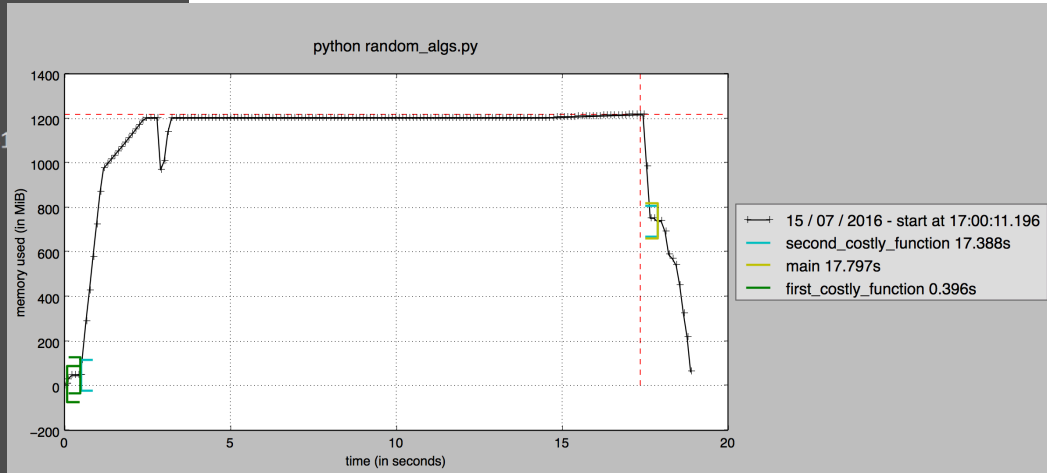
- Full program graph
- Per function graph
- Per line memory consumption
- Trigger for debugger when limit of memory reached

skyscanner

Resources: memory_profiler

```
17:04 $ time python -m memory_profiler random_algs.py
/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/memory_profiler.py:88:
  warnings.warn("psutil module not found. memory_profiler will be slow")
False

[2, 2, 2, 1014719, 11387281]
Filename: random_algs.py

Line #    Mem usage    Increment   Line Contents
================================================
     5    8.270 MiB    0.000 MiB   @profile
     6                             def first_costly_function():
     7   47.891 MiB   39.621 MiB       print is_prime(109230359867482937528940234872)


Filename: random_algs.py

Line #    Mem usage    Increment   Line Contents
================================================
    10   47.891 MiB    0.000 MiB   @profile
    11                             def second_costly_function():
    12  398.453 MiB  350.562 MiB       print trial_division(9243912311231


Filename: random_algs.py

Line #    Mem usage    Increment   Line Contents
================================================
    15    8.266 MiB    0.000 MiB   @profile
    16                             def main():
    17   47.891 MiB   39.625 MiB       first_costly_function()
    18  398.453 MiB  350.562 MiB       second_costly_function()
```

- `mprof run main.py`
- `mprof plot`
- `python -m memory_profiler main.py`
- `python -m memory_profiler --pdb-mmem=100 main.py`



LINE_PROFILER

https://github.com/rkern/line_profiler

78 Commits

Last commit: 21st Dec

872 Stars

Resources: line_profiler

Advanced version of cProfile. Shows hits, total time, per hit time and time percentage for each line of code. Easy for detecting hotspots in your program.

Also compatible with cProfile output.

Progress not lost when Ctrl+C. Displays current status ☺.

Resources: line_profiler

```
(ep) ✔ ~/Documents/europython/src [master L|…4]
17:31 $ kernprof -l -v random_algs.py
False
[2, 2, 2, 1014719, 11387281]
Wrote profile results to random_algs.py.lprof
Timer unit: 1e-06 s

Total time: 0.000248 s
File: random_algs.py
Function: inner_function at line 5

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     5                                           @profile
     6                                           def inner_function():
     7       100           46      0.5     18.5      for n in range(1, 100):
     8        99          202      2.0     81.5          n ** n

Total time: 22.7464 s
File: random_algs.py
Function: main at line 20

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    20                                           @profile
    21                                           def main():
    22         1      2037583 2037583.0      9.0      first_costly_function()
    23         1     20708800 20708800.0     91.0      second_costly_function()
```

Resources: Ipython magic

IP[y]: IPython
Interactive Computing

- Supported plugins (not by 5.0 in the case of line_profiler though)
- Interactive profiling for any function
- Easy as:
  - `%load_ext memory_profiler`
  - `%load_ext line_profiler`

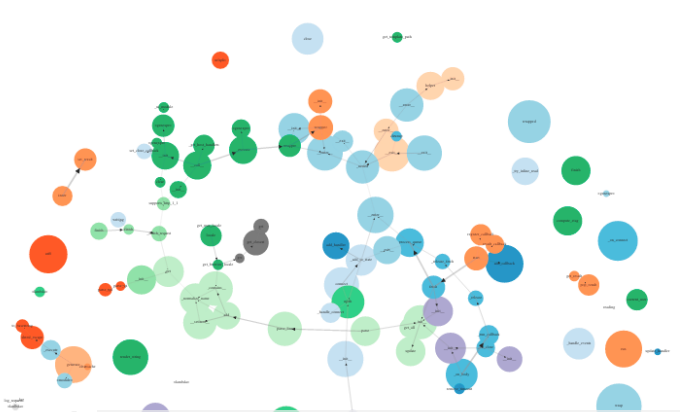
skyscanner

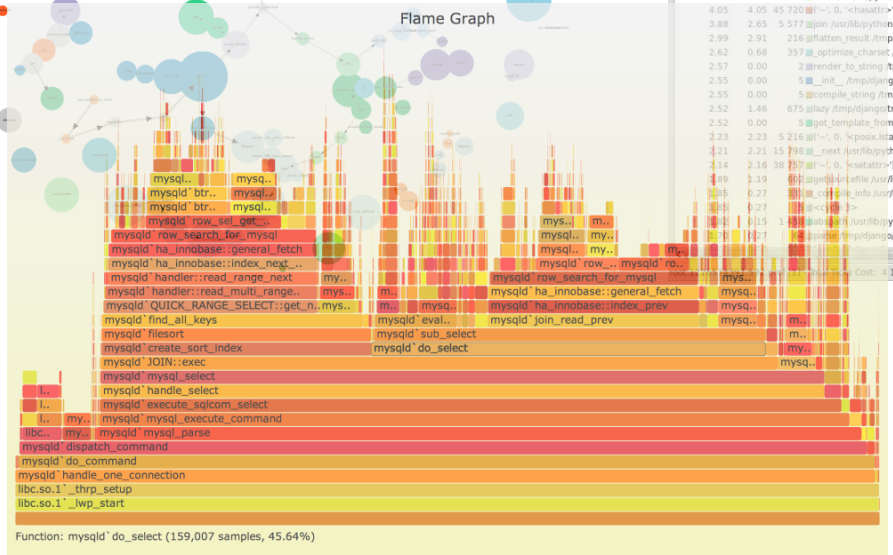
Resources: Ipython magic

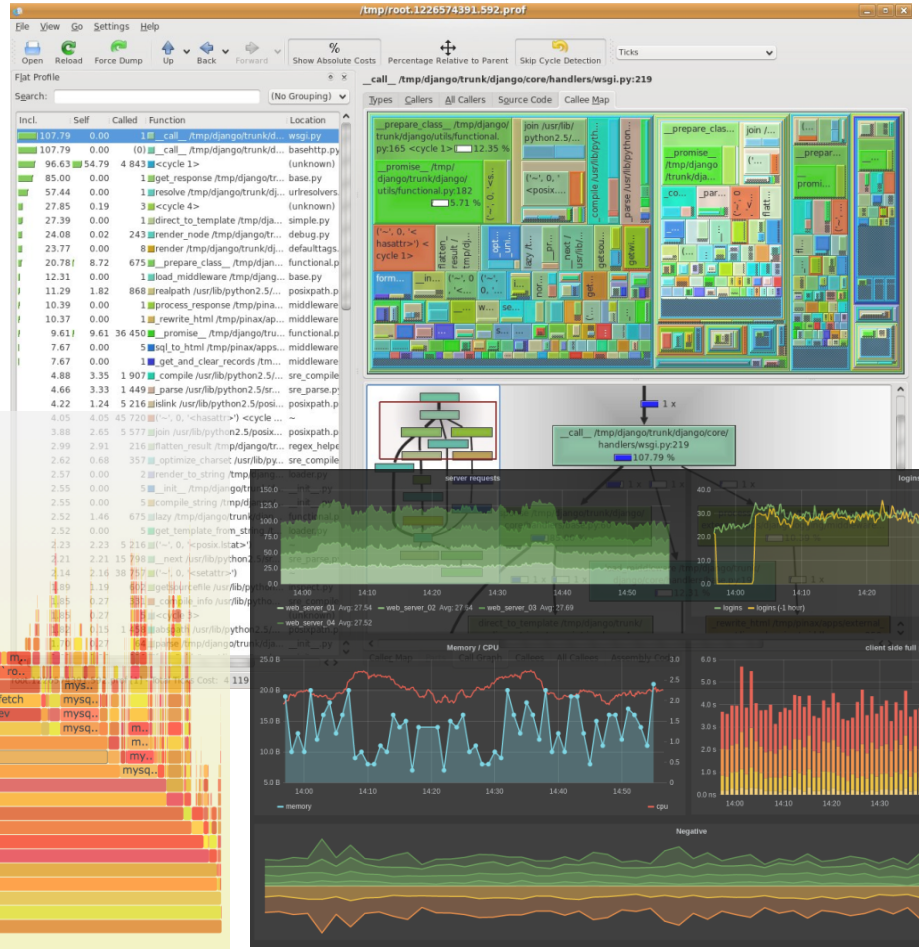
```
In [1]: %load_ext line_profiler

In [2]: import random_algs

In [3]: %lprun -f random_algs.trial_division ra
raise                random_algs          random_algs.py       random_algs.py.lprof  random_algs.pyc       rang

In [3]: %lprun -f random_algs.trial_division rand
random_algs          random_algs.py       random_algs.py.lprof  random_algs.pyc

In [3]: %lprun -f random_algs.trial_division random_algs.second_costly_function()
[2, 2, 2, 1014719, 11387281]
Timer unit: 1e-06 s

Total time: 21.9067 s
File: /Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/algorithms/factorization/trial_division.py
Function: trial_division at line 12

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    12                                           def trial_division(n):
    13                                               """
    14                                               Uses trial division to find prime factors of `n`.
    15
    16                                               :param n: An integer to factor.
    17                                               :rtype: The prime factors of `n`
    18                                               """
    19         1            2      2.0      0.0      prime_factors = []
    20         1            1      1.0      0.0      if n < 2:
    21                                                   return prime_factors
    22     79589     21813708    274.1     99.6      for p in eratosthenes(int(n**0.5) + 1):
    23     79589        36731      0.5      0.2          if p*p > n:
    24         1        12794  12794.0      0.1              break
    25     79592        43495      0.5      0.2          while n % p == 0:
    26         4            6      1.5      0.0              prime_factors.append(p)
    27         4            2      0.5      0.0              n //= p
    28         1            2      2.0      0.0      if n > 1:
    29         1            5      5.0      0.0          prime_factors.append(n)
    30         1            1      1.0      0.0      return prime_factors

In [4]:
```

skyscanner

skyscanner




Flame Graph
mysqld`row_search_for_mysql
mysqld`ha_innobase::general_fetch
mysqld`ha_innobase::index_next_..
mysqld`handler::read_range_next
mysqld`handler::read_multi_range..
mysqld`QUICK_RANGE_SELECT::get_n..
mysqld`find_all_keys
mysqld`filesort
mysqld`create_sort_index
mysqld`JOIN::exec
mysqld`mysql_select
mysqld`handle_select
mysqld`execute_sqlcom_select
mysqld`mysql_execute_command
mysqld`mysql_parse
mysqld`dispatch_command
mysqld`do_command
mysqld`handle_one_connection
libc.so.1`_thrp_setup
libc.so.1`_lwp_start
mysqld`ha_innobase::index_prev
mysqld`join_read_prev
mysqld`sub_select
mysqld`do_select
Function: mysqld`do_select (159,007 samples, 45.64%)


/tmp/root.1226574391.592.prof
File View Go Settings Help
Open Reload Force Dump Up Back Forward Show Absolute Costs Percentage Relative to Parent Skip Cycle Detection Ticks
Flat Profile
Search: (No Grouping)
Incl. Self Called Function Location
_call__/tmp/django/trunk/django/core/handlers/wsgi.py:219
Types Callers All Callers Source Code Callee Map
server requests
logins
Memory / CPU
client side full page load
Negative

PLOP

https://github.com/bdarnell/plop

95 Commits

Last commit: 14th Feb

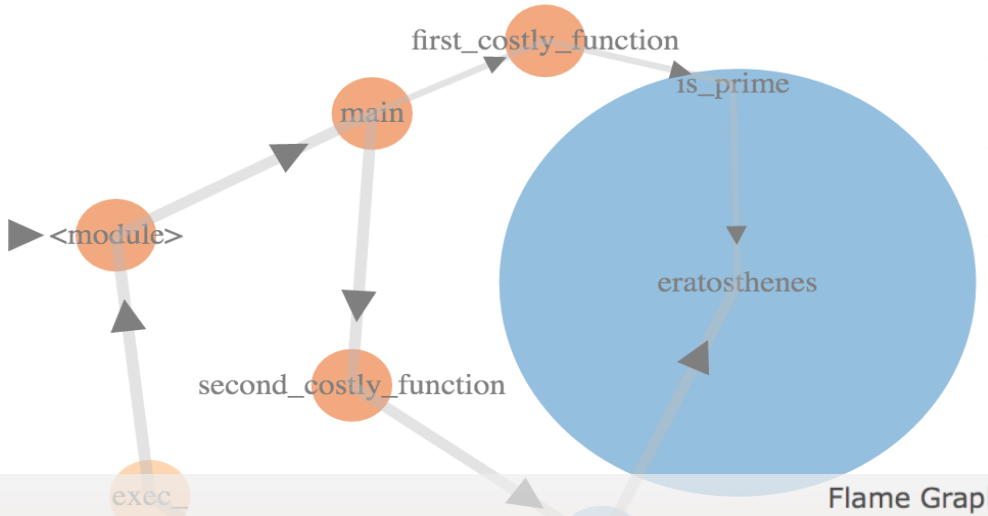
795 Stars

Advanced: Plop

Low overhead profiler. Some people is using it in production systems.

- Really low impact (use of strace and ltrace)
- It displays call graph with time spent on functions
- Flamegraph:
  - http://www.brendangregg.com/flamegraphs.html
  - https://github.com/brendangregg/FlameGraph
- Viewer running on Tornado
- With a decent setup, you can view the files while executing.

Advanced: Plop



- `python -m plop.collector random_algs.py`
- `python -m plop.viewer --datadir=profiles`
- `python -f flamegraph -m plop.collector myscript.py`
- `./flamegraph.pl profile.flame > profile.svg`

Flame Graph

Search

eratosthenes.. eratosthenes (/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/algorithms/math/sieve_eratosthenes.py:18)
is_prime (/Us.. trial_division (/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/algorithms/factorization/trial_division.py:12)
first_costly_.. second_costly_function ((string):15)
main ((string):19)
(module) ((string):1)
(module) ((string):1)
exec_ (/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/six.py:689)
main (/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/plop/collector.py:128)
(module) (/Users/manuelmiranda/.virtualenvs/ep/lib/python2.7/site-packages/plop/collector.py:1)
_run_code (/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/runpy.py:62)
_run_module_as_main (/System/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/runpy.py:136)

PYFORMANCE

https://github.com/omergertel/pyformance

123 Commits

Last commit: 20th Jun

68 Stars

Utilities for system and business metrics:

- Counting calls
- Checking average time of a function
- Grouping regex expressions for measuring time
- Measure rate of events over time
- Histograms
- Timers are shared variables. You can use timer(“name”) wherever

Advanced: Pyformance

```
# Send alarm when average higher
# than expected
def inner_function():
    for n in range(1, 100):
        with timer("test").time():
            sleep(random.uniform(0.1, 0.3))
            n ** n
    if timer("test").get_mean() > threshold:
        print "\n\nOMG SLOW EXECUTION"
        print timer("test").get_mean()
        print timer("test").get_max()
        print timer("test").get_var()
        print "mean rate", timer("test").get_mean_rate()
    print "1 min rate", timer("test").get_one_minute_rate()
```

KCACHEGRIND

https://kcachegrind.github.io/html/Home.html

809 Commits

Last commit: 6th Jul

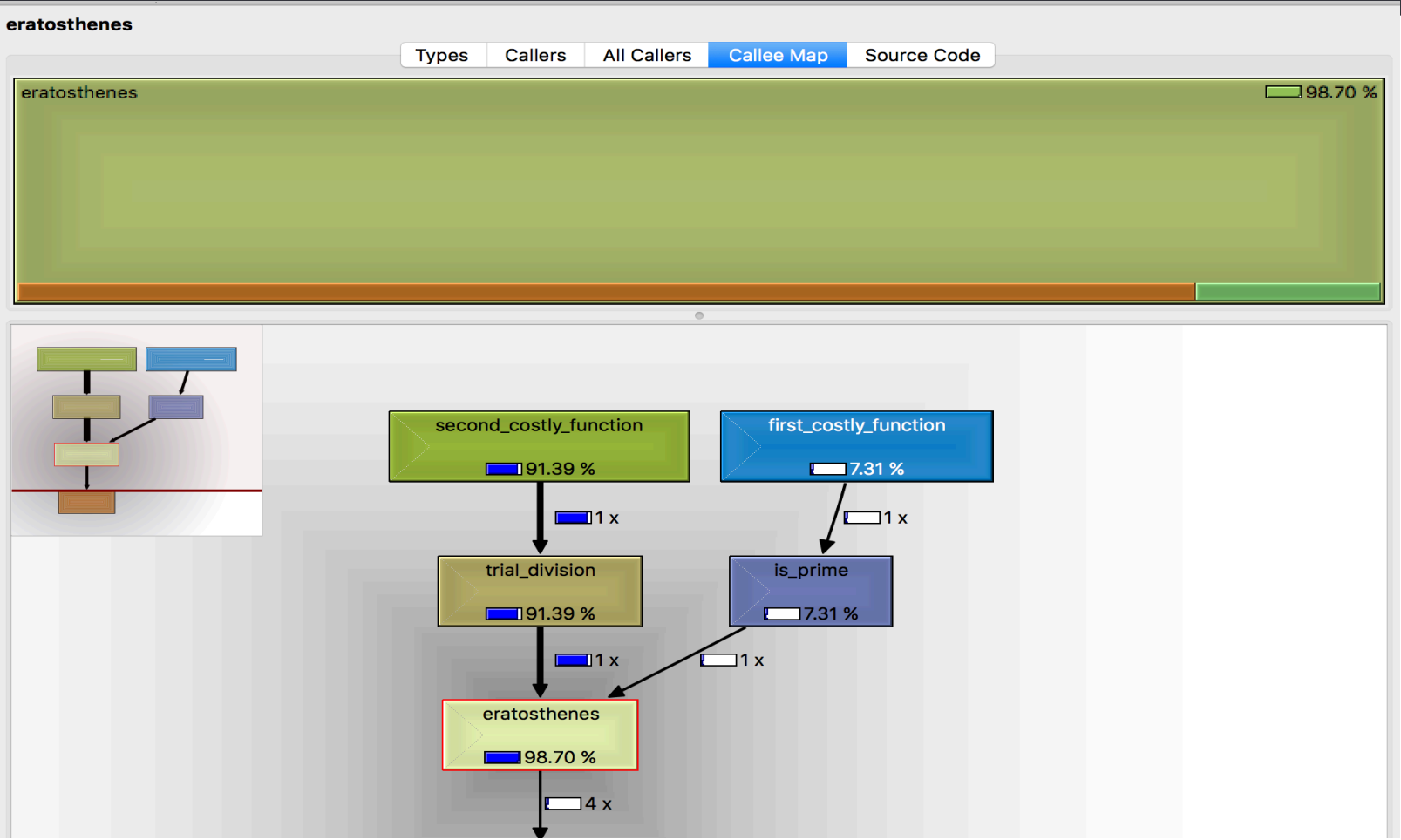
--

Advanced: Kcachegrind

Awesome (old) tool for giving global sight of your code:

- Call graph
- Execution time with percentage of time spent
- Block view of time spent for functions
- Time cost per line
- Even assembly code (which of course I use every day)
- Reads from cProfile output (pyprof2calltree)

Advanced: Kcachegrind



Summary

Use whatever tool fits your needs. Some others for other use cases:

- Aiohttp/Django/Flash debug toolbars
- vmprof
- Objgraph
- Snakeviz
- GreenletProfiler
- …

Questions

manuel.miranda@skyscanner.net
manu.mirandad@gmail.com