Python & Profiling

par *serge-sans-paille*

- Ingénieur R&D en compil' à Quarkslab
- Chercheur associé à (feu) Télécom Bretagne
- Core dev Pythran

Premature optimization is the root…

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. -- Hoare

The Fallacy of Premature Optimization

By Randall Hyde full article (http://ubiquity.acm.org/article.cfm?id=1513451)

1. != Optimization is the root of all evil
2. engineers do not consider application performance during the design of the software
3. Pareto principle is OK, but only if performance bottleneck is local
4. Moore's law is dead
5. engineer don't profile (correctly)
6. engineer time << user time
7. optimization delay delivery but improve user experience
8. better algorithm don't solve all solutions
9. substituting algorithm is not always easy

Myth: Python is slow

from the computer language benchmark game (http://benchmarksgame.alioth.debian.org/)

```
binary-trees

source       secs
Python 3     152.06
C++ g++      6.98
```

Reality: CPython can be fast

from the computer language benchmark game (http://benchmarksgame.alioth.debian.org/), `pidigits` benchmark

×	source	secs
1.0	Pascal	1.73
1.0	C gcc	1.73
1.0	Rust	1.74
1.1	Fortran	1.92
1.3	Python3	2.20
1.3	C++ g++	2.29

Reality: numeric computations can be slow

spectral-norm benchmark, that uses **list** and **scalars**

×	source	secs
1.0	C gcc	1.98
2.1	Java	4.26
8.0	Node.js	15.77
126	Python3	250.12

But Python != CPython

```
$ gcc sn.c -o sn -O3
$ time ./sn 5500
./sn 5500  4.86s user 0.00s system 99% cpu 4.864 total
$ pythran sn.py
$ python -m timeit 'import sn' 'sn.main(5500)'
10 loops, best of 3: 4.79 sec per loop
```

Understanding CPython Performance

In [1]:

```python
import dis
code = lambda x, y : x + y
dis.dis(code)
```

```
  2           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                1 (y)
              6 BINARY_ADD
              7 RETURN_VALUE
```

```
foo:
    leal    (%rdi,%rsi), %eax
    ret
```

Glimpses of explanations #1

Indirections Everywhere

- `LOAD_FAST` ⇒ array lookup
- `BINARY_ADD` ⇒ dict lookup

Function Call Overhead

- Suspend current frame
- Create a new frame
- Push it on the stack

Glimpses of explanations #2

Almost no optimizations

```
$ gcc sn.c -o sn -O0 -fsanitize=address -fsanitize=null -fsanitize=signed-integ
er-overflow -fsanitize=integer-divide-by-zero
$ time ./sn 5500
./a.out 5500  11.04s user 0.02s system 99% cpu 11.053 total
```

Glimpses of explanations #3

Poor Parallelism Support

- Global Interpreter Lock ⇒ few parallism gain (except io/native calls)
- Do not speak about vectorization
- Generally no direct hardware access

Poor Memory Locality

How many allocations in

```
[x**y for x, y in enumerate(range(100, 200))]
```

Glimpses of Hope

- Efficient dictionnary
- Cached String hashing
- BigInt comparable to GMP
- **Many Native Library Wrappers**

Python was designed as a **wrapping** language

Profiling tools

Profiling = "sampling" or "instrument"

Official tools:

- cProfile
- profile
- -hotspot-

cProfile

In [2]:
```
!python -m cProfile -h
```

```
Usage: cProfile.py [-o output_file_path] [-s sort] scriptfile [arg] ...

Options:
  -h, --help            show this help message and exit
  -o OUTFILE, --outfile=OUTFILE
                        Save stats to <outfile>
  -s SORT, --sort=SORT  Sort order when printing to stdout, based on
                        pstats.Stats class
```

cProfile Example

cumulated time, saved to a file

```
python -m cProfile -o myscript.prof -s cumtime myscript.py arg0 arg1
```

total time, printed to stdout

```
python -m cProfile -s tottime myscript.py arg0 arg1
```

cProfile Limitations

no mutithreading support

- Only main thread is profiled
- Still possible to use the API to manually collect and merge stats

function level granularity

- no line information

text output

- difficult to sort relevant information for large applications

`pip install yappi`

Pro

- multithread support
- callgrind or pstat output

Contra

- only supports Python < 3.5

`pip install pprofile`

Pro

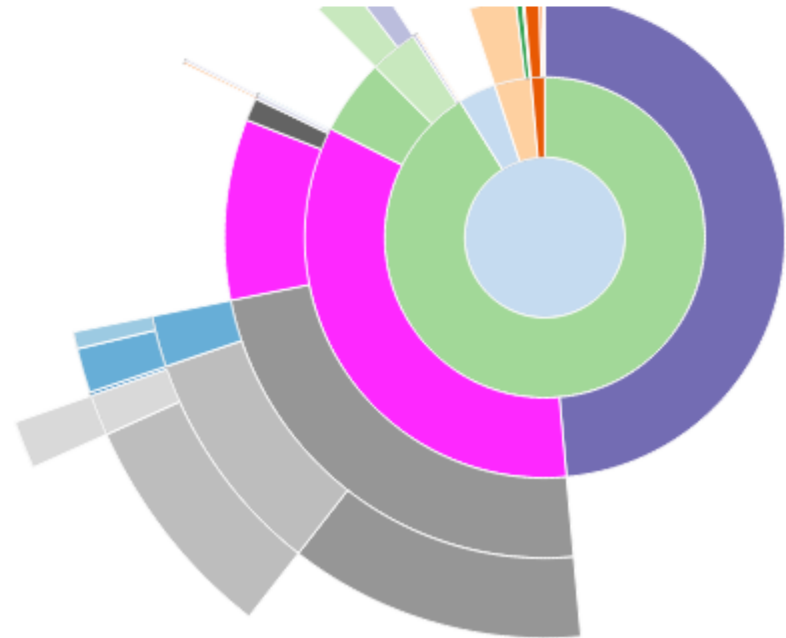
- multithread support
- callgrind or text output
- statistic profiling

Contra

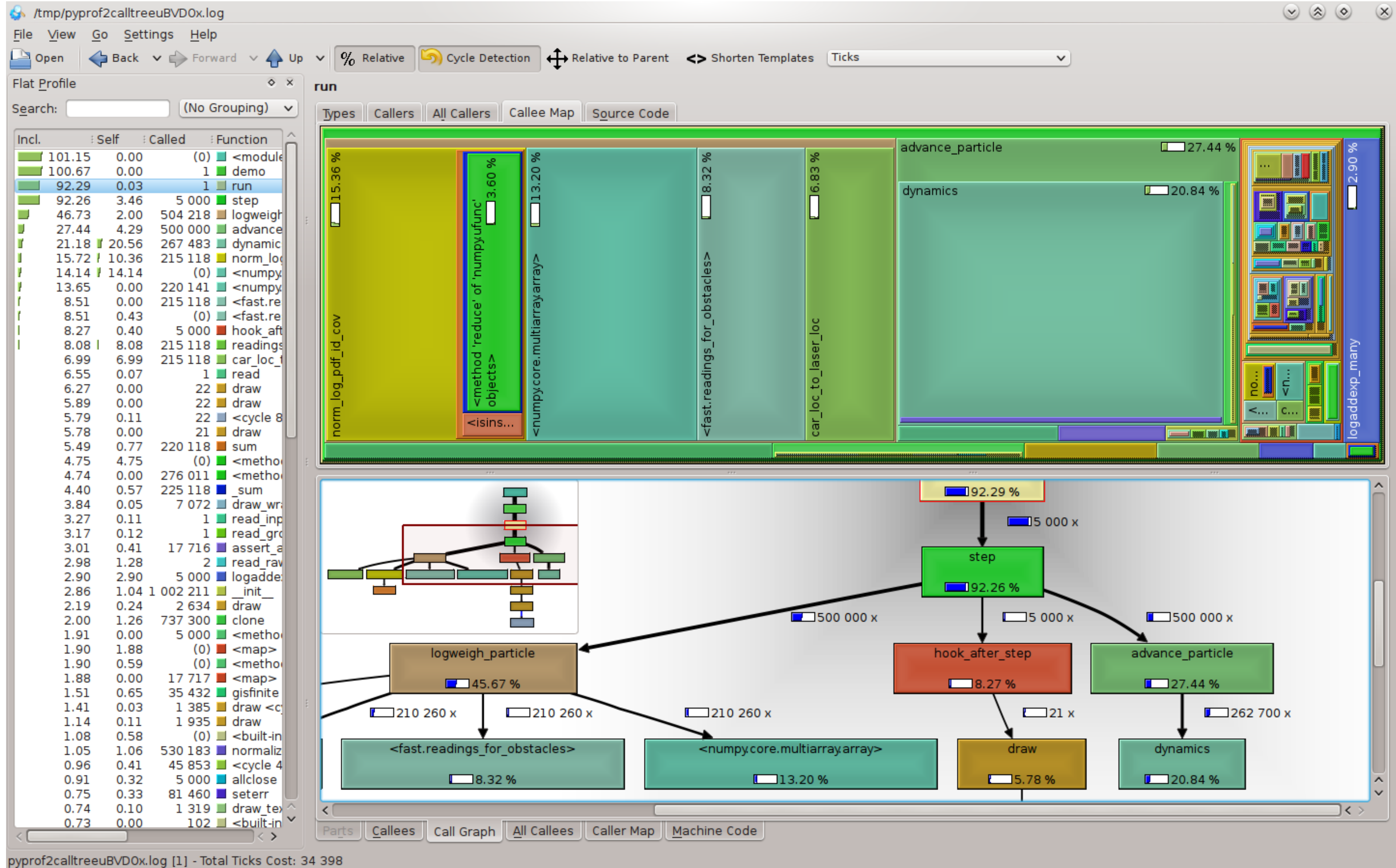
- relatively slow

```
pip install snakeviz
```

Name:
filter
Cumulative Time:
0.000294 s (31.78 %)
File:
fnmatch.py
Line:
48
Directory:
/Users/jiffyclub/miniconda3/envs/snakevizdev/lib/python3.4/



pip install pyprof2calltree + callgrind



Benchmarking

Standard tool

- timeit

Extension tool

- perf

timeit

In [3]:
```
%timeit [1 for _ in range(100)]
```

```
2.63 µs ± 61.5 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [4]:
```
%timeit -r 3 -n 100 [1 for _ in range(100)]
```

```
2.9 µs ± 74.9 ns per loop (mean ± std. dev. of 3 runs, 100 loops each)
```

In [5]:
```
!python -m timeit -s "x=list(range(100))" "[1 for _ in x]"
```

```
100000 loops, best of 3: 2.08 usec per loop
```

pip install perf

In [6]:
```
!python -m timeit -s "x=list(range(100))" "[1 for _ in x]"
```
```
100000 loops, best of 3: 2.12 usec per loop
```

In [7]:
```
!python -m perf
```
```
usage: -m perf [-h]
               {show,hist,compare_to,stats,metadata,check,collect_metadata,timei
t,system,convert,dump,slowest,command}
               ...
```

perf usage

In [8]:
```
!python -m perf timeit -o perf.json "[1 for _ in range(10)]"
```

```
ERROR: The JSON file 'perf.json' already exists
```

In [9]:
```
!python -m perf hist perf.json
```

```
568 ns:  3 ################
572 ns:  6 ###############################
576 ns:  2 ##########
579 ns:  3 ################
583 ns:  9 ###############################################
587 ns: 13 ####################################################################
591 ns:  5 ##########################
595 ns:  3 ################
598 ns:  5 ##########################
602 ns:  2 ##########
606 ns:  0 |
610 ns:  2 ##########
614 ns:  2 ##########
617 ns:  0 |
621 ns:  1 #####
625 ns:  1 #####
629 ns:  1 #####
633 ns:  0 |
636 ns:  0 |
640 ns:  0 |
644 ns:  1 #####
648 ns:  1 #####
```

In [10]:
```
!python -m perf command ls
```

```
.....................
command: Mean +- std dev: 980 us +- 29 us
```

perf again

```
$ perf record cmd arg0 arg1
```

(may require root privilege)

```
$ perf report
```

perf visu

```
Samples: 17K of event 'cycles:ppp', Event count (approx.): 234400145227499
Overhead  Command  Shared Object                      Symbol
  16.89%  python   [kernel.kallsyms]                  [.] system_call_fast_compare_end
   7.51%  python   libopenblasp-r0-39a31c03.2.18.so   [.] blas_thread_server
   5.16%  python   [kernel.kallsyms]                  [k] __schedule
   4.22%  python   [kernel.kallsyms]                  [k] kmem_cache_alloc_trace
   3.75%  python   [kernel.kallsyms]                  [k] update_curr
   3.75%  python   libc-2.24.so                       [.] _int_malloc
   3.75%  python   [kernel.kallsyms]                  [k] entry_SYSCALL_64_fastpath
   3.75%  python   [kernel.kallsyms]                  [k] get_empty_filp
   3.75%  python   python                             [.] 0x00000000000a7400
   3.75%  python   python                             [.] 0x00000000000e0a20
   3.28%  python   [kernel.kallsyms]                  [k] cpuacct_charge
   3.28%  python   [kernel.kallsyms]                  [.] native_irq_return_iret
   3.28%  python   libopenblasp-r0-39a31c03.2.18.so   [.] sched_yield@plt
   3.28%  python   python                             [.] 0x00000000000bce37
   3.28%  file     [kernel.kallsyms]                  [k] clear_page_c_e
   2.81%  python   [kernel.kallsyms]                  [k] pick_next_task_fair
   2.35%  python   python                             [.] 0x00000000000a73e0
   2.35%  python   python                             [.] 0x00000000000a64c0
   1.88%  python   [kernel.kallsyms]                  [k] native_sched_clock
   1.88%  python   python                             [.] PyObject_Malloc
   1.88%  python   python                             [.] 0x00000000000a6490
   1.41%  python   [kernel.kallsyms]                  [k] do_wp_page
   0.94%  python   [kernel.kallsyms]                  [k] _raw_spin_lock
   0.94%  python   [kernel.kallsyms]                  [k] __lru_cache_add
   0.94%  python   [kernel.kallsyms]                  [k] find_get_entry
   0.94%  python   [unknown]                          [k] 0x00007f2e263ccae5
   0.94%  file     [unknown]                          [k] 0x00007fb399690b11
   0.94%  python   [unknown]                          [k] 0x00000000004a30ac
   0.88%  python   libc-2.24.so                       [.] malloc_consolidate
   0.83%  python   gs.so                              [.] 0x00000000000179c6
   0.72%  python   gs.so                              [.] 0x0000000000017983
   0.72%  python   gs.so                              [.] 0x00000000000205ca
   0.47%  python   gs.so                              [.] 0x000000000001792e
   0.47%  python   [unknown]                          [k] 0x00007ff6b151fae5
   0.47%  file     libmagic.so.1.0.0                  [.] 0x000000000000ffa9
   0.41%  python   gs.so                              [.] 0x0000000000017927
   0.36%  python   gs.so                              [.] 0x0000000000017a1a
   0.36%  python   gs.so                              [.] 0x00000000000179a4
   0.36%  python   gs.so                              [.] 0x00000000000179bc
   0.36%  python   gs.so                              [.] 0x0000000000020153
   0.36%  python   gs.so                              [.] 0x00000000000205b0
   0.36%  python   gs.so                              [.] 0x000000000001cdf6
   0.00%  python   [kernel.kallsyms]                  [k] entry_SYSCALL_64_after_swapgs
   0.00%  python   gs.so                              [.] 0x0000000000017a07
   0.00%  python   gs.so                              [.] 0x00000000000179fb
   0.00%  python   gs.so                              [.] 0x0000000000017a16
   0.00%  python   libc-2.24.so                       [.] malloc
   0.00%  python   gs.so                              [.] 0x00000000000179c2
   0.00%  python   gs.so                              [.] 0x00000000000179e2
   0.00%  python   gs.so                              [.] 0x0000000000017901
   0.00%  python   gs.so                              [.] 0x0000000000017954
   0.00%  python   python                             [.] PyEval_EvalFrameEx
Tip: Use --symfs <dir> if your symbol files are in non-standard locations
```

perf visu++

```
std::copy<pythonic::types::numpy_expr_iterator<pythonic::types::numpy_expr<pythonic::operator_::functor::add, pythonic::types::numpy_expr<pythonic::operator_::functor::sub, pythonic::types::n
       │      _ZN8pythonic9operator_3subIdfEEDTmifp_fp0_ERKT_RKT0_():
       │        movapd %xmm0,%xmm2
       │      _ZN8pythonic9operator_3subIffEEDTmifp_fp0_ERKT_RKT0_():
       │        movaps %xmm6,%xmm0
       │      _ZN8pythonic9operator_3subIdfEEDTmifp_fp0_ERKT_RKT0_():
       │        subsd  %xmm1,%xmm2
       │      _ZN8pythonic9operator_3subIffEEDTmifp_fp0_ERKT_RKT0_():
 11.11 │        subss  (%r9),%xmm0
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3subEJNS0_9broadcastIffEENS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJNS0_16contiguo
       │        add    %rbp,%r9
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3addEJNS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJNS0_16contiguous_sliceEEEESE_EEES
       │        mov    %rdx,0xc68(%rsp)
       │        mov    %rcx,0xc60(%rsp)
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3mulEJNS0_9broadcastIffEENS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJNS0_16contiguo
 11.11 │        mov    %rsi,0xc38(%rsp)
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3addEJNS2_INS4_3subEJNS2_IS5_JNS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJNS0_16con
       │        mov    %rdi,0xc00(%rsp)
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3addEJNS2_IS5_JNS2_INS4_3subEJNS2_IS5_JNS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJ
 11.11 │        mov    %rax,0xbe0(%rsp)
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3subEJNS2_INS4_3mulEJNS0_9broadcastIfdEENS2_INS4_3addEJNS2_IS9_JNS2_IS5_JNS2_IS9_JNS0_11numpy_gexprI
       │        mov    %r8,0xba8(%rsp)
       │      _ZN8pythonic5types19numpy_expr_iteratorINS0_10numpy_exprINS_9operator_7functor3subEJNS0_9broadcastIffEENS0_11numpy_gexprINS0_11numpy_iexprIRKNS0_7ndarrayIfLm2EEEEEJNS0_16contiguo
 11.11 │        mov    %r9,0xb78(%rsp)
       │      _ZN8pythonic9operator_3mulIdfEEDTmlfp_fp0_ERKT_RKT0_():
       │        cvtss2 %xmm0,%xmm0
       │        movapd %xmm0,%xmm1
 11.11 │        mulsd  %xmm5,%xmm1
       │      _ZN8pythonic9operator_3addIddEEDTplfp_fp0_ERKT_RKT0_():
       │        addsd  %xmm1,%xmm2
       │      _ZNSt11__copy_moveILb0ELb0ESt26random_access_iterator_tagE8__copy_mIN8pythonic5types19numpy_expr_iteratorINS4_10numpy_exprINS3_9operator_7functor3addEJNS6_INS8_3subEJNS6_INS8_3mu
       │                    *__result = *__first;
       │        movsd  %xmm2,-0x8(%r11)
       │                for(_Distance __n = __last - __first; __n > 0; --__n)
       │        sub    $0x1,%r10
       │      ↑ jne    4f0
       │        mov    0x18(%rsp),%rax
       │        mov    0x10(%rsp),%rcx
       │        lea    (%rax,%rcx,8),%rax
       │      _ZSt4copyIN8pythonic5types19numpy_expr_iteratorINS1_10numpy_exprINS0_9operator_7functor3addEJNS3_INS5_3subEJNS3_INS5_3mulEJNS1_9broadcastIfdEENS3_IS6_JNS3_IS6_JNS3_IS7_JNS3_IS6_J
       │            __glibcxx_requires_valid_range(__first, __last);
       │
       │            return (std::__copy_move_a2<__is_move_iterator<_II>::__value>
       │                    (std::__miter_base(__first), std::__miter_base(__last),
       │                     __result));
       │          }
       │ 5ae:   add    $0xdd8,%rsp
       │        pop    %rbx
       │        pop    %rbp
       │        pop    %r12
       │        pop    %r13
       │        pop    %r14
       │        pop    %r15
       │      ← retq
Press 'h' for help on key bindings
```

Conclusion

- Profile before you optimize
- Think before you profile
- Plenty of supporting tools