PROFILING PYTHON CODE

IMPROVE PERFORMANCE FOR GREAT GOOD!

WHAT IS PROFILING?

Measure execution time and resource usage of programs to identify bottlenecks

Types of profiling:

- Line based
- Function based

Resources:

- CPU Time
- RAM Use
- GPU Time
- GPU Memory
- (I/O Time)
- (Networking)

Profiler	Slowdown	Lines or Functions?	Unmodified Code?	Threads?	Multiprocessing?	Python vs. C Time?	System Time?	Profiles Memory?	GPU?	Memory Trends?	Copy Volume?	Detects Leaks?
`pprofile (stat.)`	1×	lines	✓	✓	-	-	-	-	-	-	-	-
`py-spy`	1×	lines	✓	✓	-	-	-	-	-	-	-	-
`pyinstrument`	1.5×	functions	✓	-	-	-	-	-	-	-	-	-
`cProfile`	2×	functions	✓	-	-	-	-	-	-	-	-	-
`yappi wallclock`	3×	functions	✓	✓	-	-	-	-	-	-	-	-
`yappi CPU`	5×	functions	✓	✓	-	-	-	-	-	-	-	-
`line_profiler`	7×	lines	-	-	-	-	-	-	-	-	-	-
`Profile`	40×	functions	✓	-	-	-	-	-	-	-	-	-
`pprofile (det.)`	40×	lines	✓	✓	-	-	-	-	-	-	-	-
`memory_profiler`	270×	lines	-	-	-	-	-	✓	-	-	-	-
`Scalene`	**1.2×**	**both**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

LINE VS FUNCTION BASED

Function Based:

```
➜ python3 -m cProfile script.py

58 function calls in 9.419 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   1    0.000    0.000    9.419    9.419 part1.py:1(<module>)
   51   9.419    0.185    9.419    0.185 part1.py:1(computation)
   1    0.000    0.000    9.419    9.419 part1.py:10(function1)
   1    0.000    0.000    9.243    9.243 part1.py:15(function2)
   1    0.000    0.000    0.176    0.176 part1.py:20(function3)
   1    0.000    0.000    9.419    9.419 part1.py:24(main)
```

Line Based:

```
   11  |         |         |            | def calculate_z_serial_purepython(maxiter, zs, cs):
   12  |         |         |            |     """Calculate output list using Julia update rule"""
   13  |  0.08%  |  0.02%  |     0.06   |     output = [0] * len(zs)
   14  |  0.25%  |  0.01%  |     9.50   |     for i in range(len(zs)):
   15  |         |         |            |         n = 0
   16  |  1.34%  |  0.05%  |    -9.88   |         z = zs[i]
   17  |  0.50%  |  0.01%  |    -8.44   |         c = cs[i]
   18  |  1.25%  |  0.04%  |            |         while abs(z) < 2 and n < maxiter:
   19  | 68.67%  |  2.27%  |    42.50   |             z = z * z + c
   20  | 18.46%  |  0.74%  |   -33.62   |             n += 1
   21  |         |         |            |         output[i] = n
   22  |         |         |            |     return output
```

SCALENE - SIMPLE EXAMPLE

```
[zenon@ralfSDSC example1]$ scalene example.py
Scalene: internal error: unable to find Python allocator functions
Scalene: internal error: unable to find Python allocator functions
                                        Memory usage: ___________________ (max:   0.00MB, growth rate:   0%)
                                                  example.py: % of time = 100.00% out of   7.21s.

  Line |Time    |------ |------ |Memory  |------ |----------     |Copy    |
       |Python  |native |system |Python  |avg    |timeline/%     |(MB/s)  |example.py
-------------------------------------------------------------------------------------------------------------
     1 |        |       |       |        |       |               |        |from typing import List
     2 |        |       |       |        |       |               |        |
     3 |        |       |       |        |       |               |        |def top_level() -> List[int]:
     4 |        |       |       |        |       |               |        |    result = []
     5 |        |       |       |        |       |               |        |    for i in range(15000):
     6 |        |       |       |        |       |               |        |        l = lower_level(i)
     7 |    14% |       |       |        |       |               |    161 |        s = sum(l)
     8 |        |       |       |        |       |               |        |        result.append(s)
     9 |        |       |       |        |       |               |        |
    10 |        |       |       |        |       |               |        |    return result
    11 |        |       |       |        |       |               |        |
    12 |        |       |       |        |       |               |        |
    13 |        |       |       |        |       |               |        |
    14 |        |       |       |        |       |               |        |def lower_level(i: int) -> List[int]:
    15 |        |       |       |        |       |               |        |    result = []
    16 |        |       |       |        |       |               |        |
    17 |    23% |       |       |        |       |               |    215 |    for j in range(i):
    18 |    58% |    2% |       |        |       |               |    604 |        result.append(j)
    19 |        |       |       |        |       |               |        |
    20 |        |       |       |        |       |               |      1 |    return result
    21 |        |       |       |        |       |               |        |
    22 |        |       |       |        |       |               |        |
    23 |        |       |       |        |       |               |        |if __name__ == "__main__":
    24 |        |       |       |        |       |               |        |    result = top_level()
    25 |        |       |       |        |       |               |        |
-------------------------------------------------------------------------------------------------------------
       |        |       |       |        |       |               |        |function summary for example.py
     3 |    14% |       |       |        |       |               |    161 |top_level
    14 |    81% |    2% |    2% |        |       |               |    819 |lower_level

[zenon@ralfSDSC example1]$
```

00:00

DIFFERENT OUTPUTS

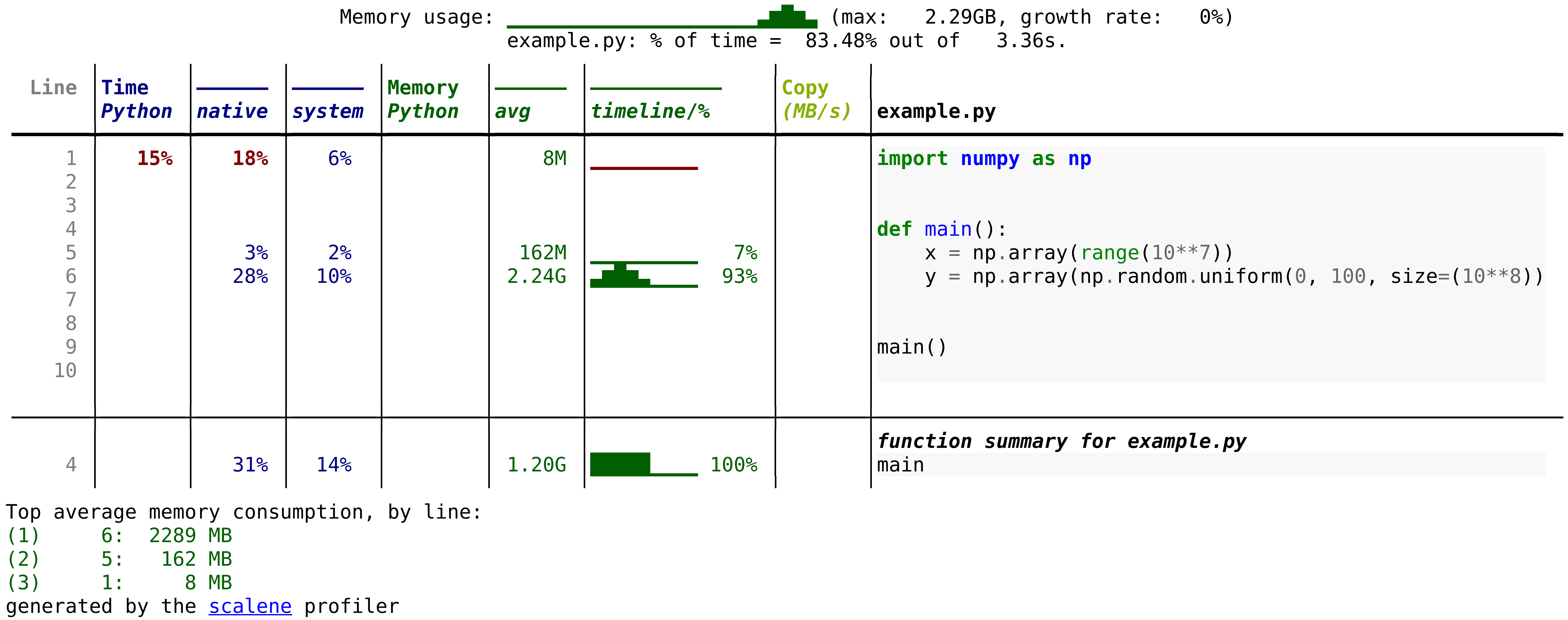
scalene --html --outfile profile.html

Memory usage: (max: 0.00MB, growth rate: 0%)
example.py: % of time = 99.87% out of 7.72s.

Line	Time Python	native	system	Memory Python	avg	timeline/%	Copy (MB/s)	example.py
1								`from typing import List`
2								
3								`def top_level() -> List[int]:`
4								`    result = []`
5								`    for i in range(15000):`
6								`        l = lower_level(i)`
7	13%						107	`        s = sum(l)`
8							2	`        result.append(s)`
9								
10								`    return result`
11								
12								
13								
14								`def lower_level(i: int) -> List[int]:`
15								`    result = []`
16								
17	20%						226	`    for j in range(i):`
18	63%	2%	1%				581	`        result.append(j)`
19								
20								`    return result`
21								
22								
23								`if __name__ == "__main__":`
24								`    result = top_level()`
25								

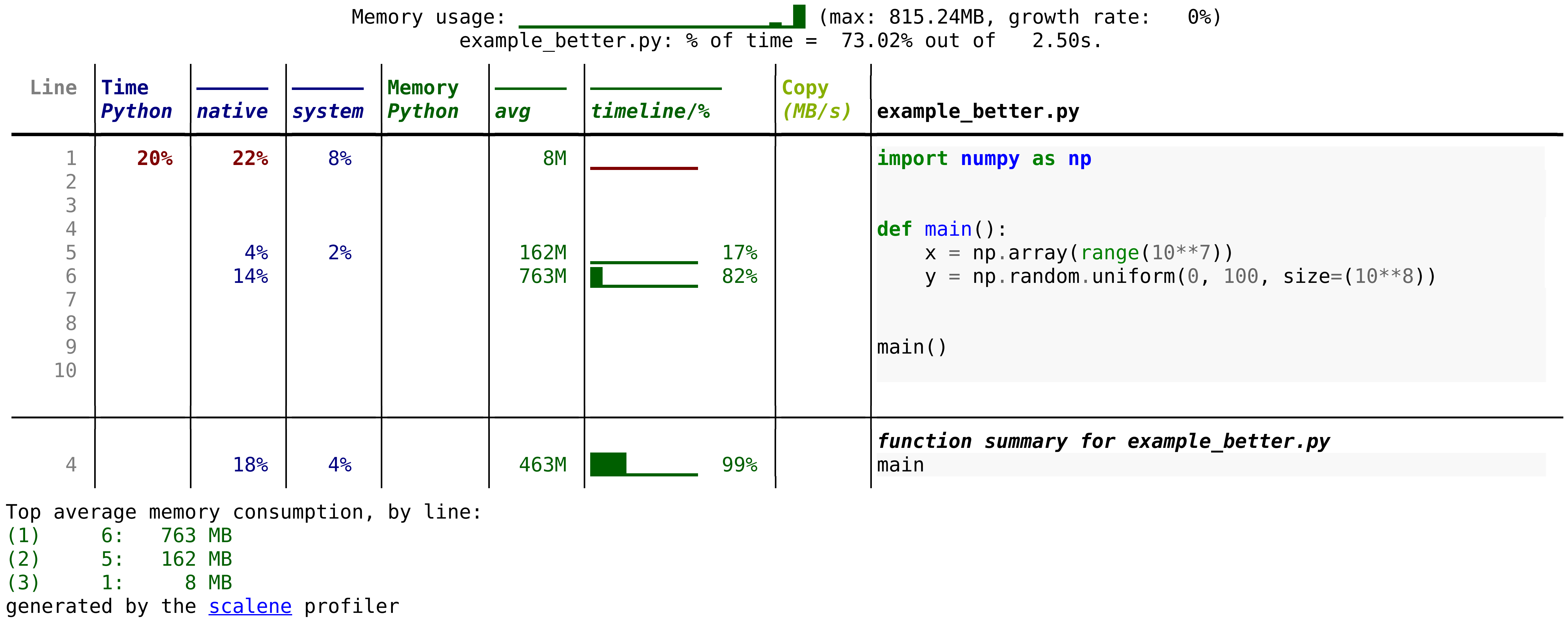
								function summary for example.py
3	13%						109	top_level
14	82%	3%	2%				807	lower_level

SCALENE - NATIVE CODE EXAMPLE



```
                    Memory usage:                      (max:   2.29GB, growth rate:   0%)
                                  example.py: % of time =  83.48% out of   3.36s.

 Line │Time    │        │        │Memory  │       │                │Copy   │
      │Python  │native  │system  │Python  │avg    │timeline/%      │(MB/s) │example.py
──────┼────────┼────────┼────────┼────────┼───────┼────────────────┼───────┼──────────────────────────────────────────────────────────
    1 │    15% │    18% │     6% │        │     8M│                │       │import numpy as np
    2 │        │        │        │        │       │                │       │
    3 │        │        │        │        │       │                │       │
    4 │        │        │        │        │       │                │       │def main():
    5 │        │     3% │     2% │        │   162M│              7%│       │    x = np.array(range(10**7))
    6 │        │    28% │    10% │        │  2.24G│             93%│       │    y = np.array(np.random.uniform(0, 100, size=(10**8)))
    7 │        │        │        │        │       │                │       │
    8 │        │        │        │        │       │                │       │
    9 │        │        │        │        │       │                │       │main()
   10 │        │        │        │        │       │                │       │
──────┼────────┼────────┼────────┼────────┼───────┼────────────────┼───────┼──────────────────────────────────────────────────────────
      │        │        │        │        │       │                │       │function summary for example.py
    4 │        │    31% │    14% │        │  1.20G│            100%│       │main
Top average memory consumption, by line:
(1)     6:  2289 MB
(2)     5:   162 MB
(3)     1:     8 MB
generated by the scalene profiler
```

SCALENE - NATIVE CODE EXAMPLE



```
                 Memory usage:                         (max: 815.24MB, growth rate:   0%)
                           example_better.py: % of time =  73.02% out of   2.50s.

 Line   Time                                 Memory                                    Copy
        Python   native   system   Python       avg   timeline/%                 (MB/s)   example_better.py
   1      20%      22%       8%                    8M                                      import numpy as np
   2
   3
   4                                                                                       def main():
   5                4%       2%                  162M                       17%                 x = np.array(range(10**7))
   6               14%                           763M                       82%                 y = np.random.uniform(0, 100, size=(10**8))
   7
   8
   9                                                                                       main()
  10

                                                                                           function summary for example_better.py
   4               18%       4%                  463M                       99%            main
Top average memory consumption, by line:
(1)     6:   763 MB
(2)     5:   162 MB
(3)     1:     8 MB
generated by the scalene profiler
```

SCALENE - GPU CODE EXAMPLE

Memory usage: (max: 921.85MB, growth rate: 0%)
example3.py: % of time = 100.00% out of 14.20s.

Line	Time Python	native	system	GPU	Memory Python	avg	timeline/%	Copy (MB/s)	example3.py
1	33%	10%	1%	14%		62M	…	2	import torch
2									import math
3									
4									
5									def torchtest():
6									dtype = torch.float
7									device = torch.device("cuda:0") # Uncomment this to run on GPU
8						1.31G	9…		q = torch.linspace(-math.pi, math.pi, 5000000, device=device, dtype=dtype)
9		7%	2%					132	x = torch.linspace(-math.pi, math.pi, 5000000, device=device, dtype=dtype)
10									y = torch.sin(x)
11									
12									a = torch.randn(()), device=device, dtype=dtype, requires_grad=True)
13									b = torch.randn(()), device=device, dtype=dtype, requires_grad=True)
14									c = torch.randn(()), device=device, dtype=dtype, requires_grad=True)
15									d = torch.randn(()), device=device, dtype=dtype, requires_grad=True)
16									
17									learning_rate = 1e-6
18									for t in range(2000):
19				1%					y_pred = a + b * x + c * x ** 2 + d * x ** 3
20									loss = (y_pred - y).sum()
21									if t % 100 == 99:
22	1%	17%		4%					print(t, loss.item())
23									loss.backward()
24									
25									with torch.no_grad():
26	12%		8%	59%				4	a -= learning_rate * a.grad
27				5%					b -= learning_rate * b.grad
28	1%			5%					c -= learning_rate * c.grad
29	2%			10%					d -= learning_rate * d.grad
30									
31									a.grad = None
32									b.grad = None
33									c.grad = None
34									d.grad = None
35									
36									print(f'Result: y = {a.item()} + {b.item()} x + {c.item()} x^2 + {d.item()} x^3')
37									
38									
39									torchtest()

SCALENE MEETS THE REALWORLD

Sometimes it doesn't work that well...

We could filter using **--profile-only**

```
$ scalene --profile-only renku/cli/__main__.py -- renku/cli/__init__.py --version
```

Or try showing the reduced profile with **--reduced-profile**

```
$ scalene --reduced-profile -- renku/cli/__init__.py --version
```

Or use the decorator

```
from scalene import profile

@profile
def my_slow_function():
    [...]
```

BRINGING OUT THE BIG GUNS - CPROFILE

$ python -m cProfile [-m module | file]

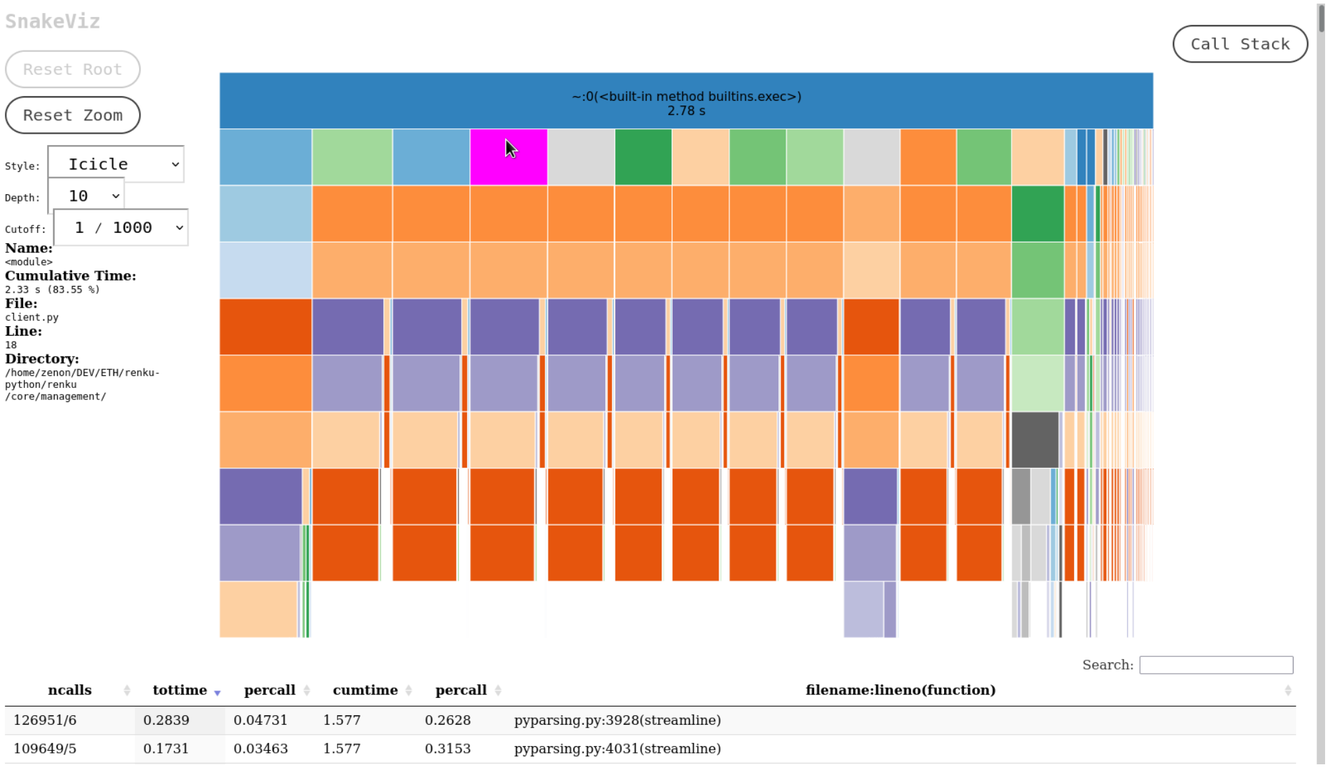
```
$ python -m cProfile -s cumulative -m renku.cli --version
0.16.1
    6786351 function calls (6157157 primitive calls) in 2.753 seconds

Ordered by: cumulative time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  21    0.000    0.000    6.535    0.311 __init__.py:18(<module>)
1370/1    0.008    0.000    2.756    2.756 {built-in method builtins.exec}
    1    0.000    0.000    2.756    2.756 <string>:1(<module>)
    1    0.000    0.000    2.756    2.756 runpy.py:195(run_module)
  2/1    0.000    0.000    2.755    2.755 runpy.py:102(_get_module_details)
199/4    0.000    0.000    2.747    0.687 {built-in method builtins.__import__}
1363/2    0.005    0.000    2.747    1.374 <frozen importlib._bootstrap>:978(_find_and_load)
1361/2    0.004    0.000    2.747    1.374 <frozen importlib._bootstrap>:948(_find_and_load_unlocked)
1313/2    0.004    0.000    2.747    1.374 <frozen importlib._bootstrap>:663(_load_unlocked)
1228/2    0.002    0.000    2.747    1.374 <frozen importlib._bootstrap_external>:722(exec_module)
1552/2    0.001    0.000    2.747    1.373 <frozen importlib._bootstrap>:211(_call_with_frames_removed)
    1    0.000    0.000    2.288    2.288 client.py:18(<module>)
[...]
```

ACTUALLY WORKING WITH CPROFILE

```
$ python -m cProfile -o outfile [-m module | file]
$ snakeviz outfile
```

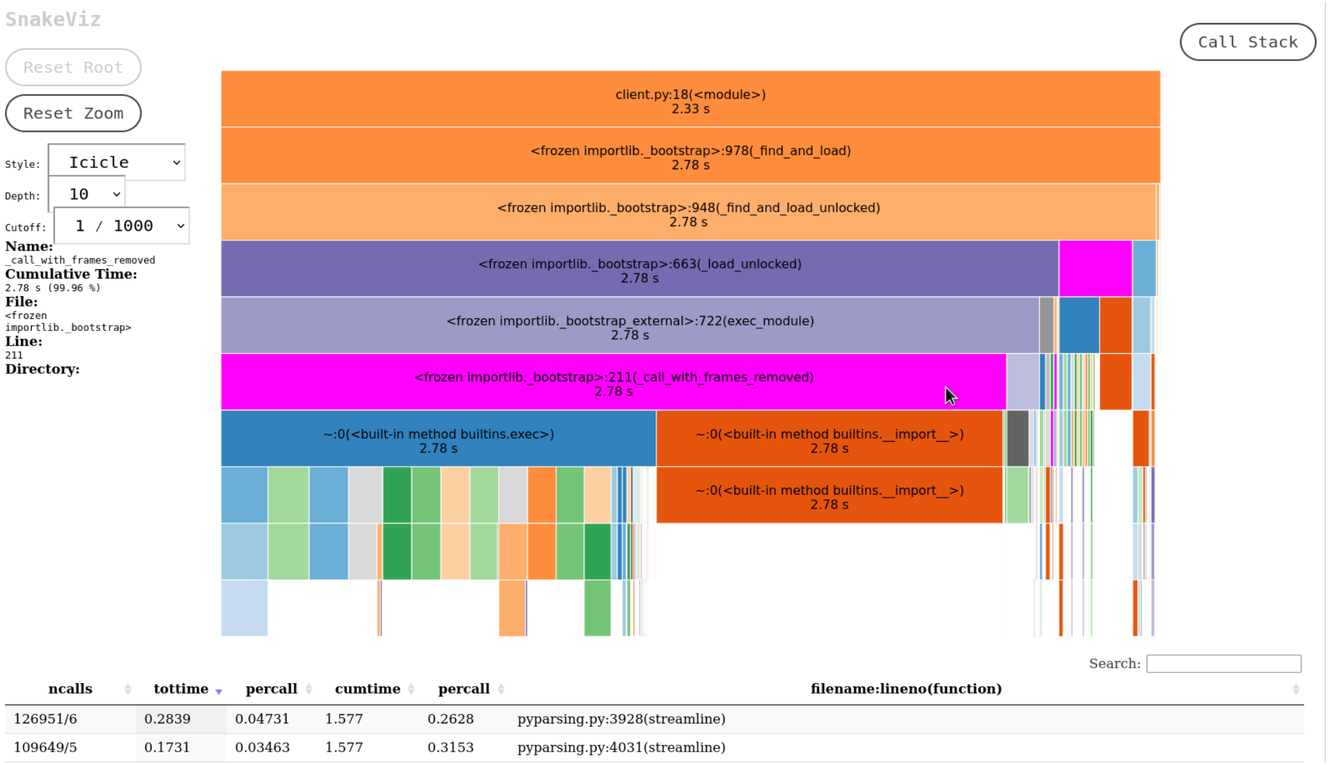
ACTUALLY WORKING WITH CPROFILE

```
$ python -m cProfile -o outfile [-m module | file]
$ snakeviz outfile
```



ACTUALLY WORKING WITH CPROFILE

```
$ python -m cProfile -o outfile [-m module | file]
$ snakeviz outfile
```



ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
126951/6	0.2839	0.04731	1.577	0.2628	pyparsing.py:3928(streamline)
109649/5	0.1731	0.03463	1.577	0.3153	pyparsing.py:4031(streamline)

TOO MANY IMPORTS!

$ python -X importtime myscript.py

```
import time: self [us] | cumulative | imported package
import time:       152 |        152 | zipimport
import time:       969 |        969 | _frozen_importlib_external
import time:       117 |        117 |     _codecs
import time:       851 |        967 |   codecs
import time:       717 |        717 |   encodings.aliases
[...]
import time:       198 |        198 |               cwlgen.workflowdeps
import time:       159 |        357 |             cwlgen.workflow
import time:       208 |       9497 |           cwlgen
import time:       323 |       9888 |         renku.core.management.workflow.converters.cwl
import time:       104 |       9992 |       renku.core.plugins.implementations
import time:       153 |        153 |       renku.core.plugins.run
import time:       137 |      10280 |     renku.core.plugins.pluginmanager
import time:     27119 |     101357 |   renku.cli.workflow
import time:       161 |        161 |       lockfile.linklockfile
import time:       314 |        474 |     lockfile
import time:       536 |       1010 |   renku.core.commands.version
import time:       731 |     347862 | renku.cli
```

THE PROBLEM

```
63  import os
64  import sys
65  import uuid
66  from pathlib import Path
67
68  import click
69  import click_completion
70  import yaml
71  from click_plugins import with_plugins
72  from pkg_resources import iter_entry_points
73
74  from renku.cli.clone import clone
75  from renku.cli.config import config
76  from renku.cli.dataset import dataset
77  from renku.cli.doctor import doctor
78  from renku.cli.exception_handler import IssueFromTraceback
79  from renku.cli.githooks import githooks as githooks_command
80  from renku.cli.graph import graph
81  from renku.cli.init import init as init_command
82  from renku.cli.log import log
83  from renku.cli.login import login, logout, token
84  from renku.cli.migrate import check_immutable_template_files, migrate, migrationscheck
85  from renku.cli.move import move
86  from renku.cli.project import project
87  from renku.cli.remove import remove
88  from renku.cli.rerun import rerun
89  from renku.cli.run import run
90  from renku.cli.save import save
91  from renku.cli.service import service
92  from renku.cli.status import status
93  from renku.cli.storage import storage
94  from renku.cli.update import update
95  from renku.cli.workflow import workflow
96  from renku.core.commands.echo import WARNING
97  from renku.core.commands.options import install_completion, option_external_storage_requested
98  from renku.core.commands.version import check_version, print_version
99  from renku.core.errors import UsageError
100 from renku.core.management.client import LocalClient
101 from renku.core.management.config import RENKU_HOME, ConfigManagerMixin
102 from renku.core.management.repository import default_path
```

HONORARY MENTIONS - TIME

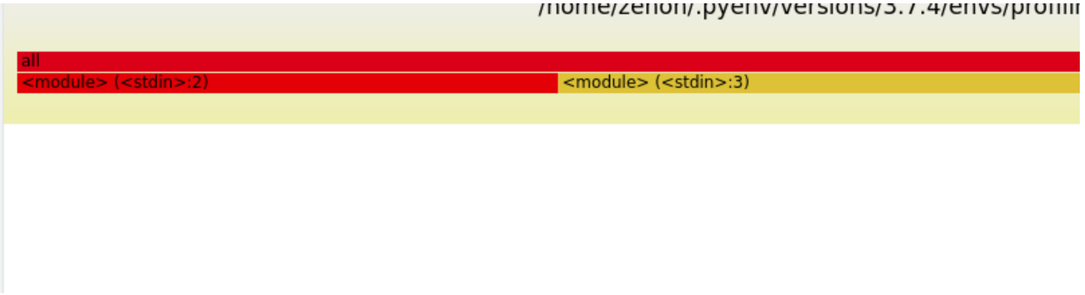
time

```
$ time python myfile.py
[...]

real    0m2.173s
user    0m2.025s
sys     0m0.143s
```

HONORARY MENTIONS - PY-SPY

Runtime profiling!

```
$ python -m myserver &
$ sudo su # needs root unfortunately
$ ps aux |grep python
user     817920 21.5  0.0  14948  9924 pts/2    R+   18:13   0:14 /usr/bin/python -m myserver
$ py-spy record --pid 817920
py-spy> Sampling process 100 times a second. Press Control-C to exit.
^C
py-spy> Stopped sampling because Control-C pressed
py-spy> Wrote flamegraph data to '817920-2021-10-06T18:20:03+02:00.svg'. Samples: 619 Errors: 0
```



PYTORCH PROFILER

```python
import torch
import torchvision.models as models
from torch.profiler import profile, record_function, ProfilerActivity

model = models.resnet18()
inputs = torch.randn(5, 3, 224, 224)

with profile(activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA],
  profile_memory=True, record_shapes=True
) as prof:
  with record_function("model_inference"):
    model(inputs)
print(prof.key_averages().table(sort_by="cuda_time_total", row_limit=10))
```

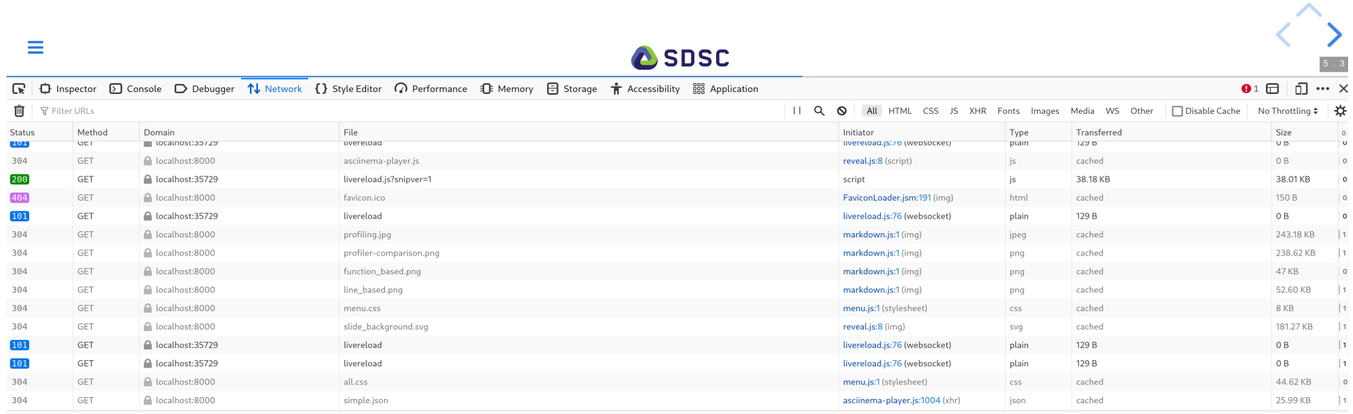
```
# ---------------------------------  ------------  ------------  ------------
#                              Name       CPU Mem  Self CPU Mem    # of Calls
# ---------------------------------  ------------  ------------  ------------
#                       aten::empty      94.79 Mb      94.79 Mb           121
#     aten::max_pool2d_with_indices      11.48 Mb      11.48 Mb             1
#                       aten::addmm      19.53 Kb      19.53 Kb             1
#               aten::empty_strided         572 b         572 b            25
#                     aten::resize_         240 b         240 b             6
#                         aten::abs         480 b         240 b             4
#                         aten::add         160 b         160 b            20
#               aten::masked_select         120 b         112 b             1
#                          aten::ne         122 b          53 b             6
#                          aten::eq          60 b          30 b             2
# ---------------------------------  ------------  ------------  ------------
# Self CPU time total: 53.064ms
```

HONORARY MENTIONS - BROWSER DEBUG CONSOLE (F12)

Shown by pressing F12 in your browser.
Also helpful for debugging other problems.

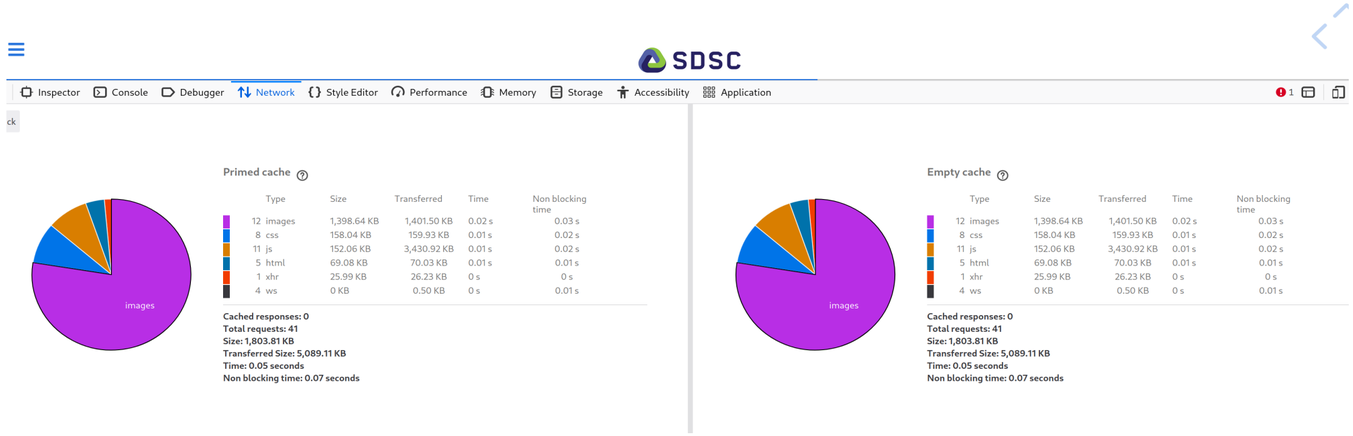
HONORARY MENTIONS - BROWSER DEBUG CONSOLE (F12)

Shown by pressing F12 in your browser.
Also helpful for debugging other problems.



HONORARY MENTIONS - BROWSER DEBUG CONSOLE (F12)

Shown by pressing F12 in your browser.
Also helpful for debugging other problems.



WHAT CAN WE DO ABOUT IT?

MEMOIZATION

```
def fib(n):
  if n < 2:
    return n
  return fib(n-1) + fib(n-2)

fib(50)
```

MEMOIZATION 2

```
from functools import lru_cache

@lru_cache(maxsize = 128)
def fib_with_cache(n):
  if n < 2:
    return n
  return fib_with_cache(n-1) + fib_with_cache(n-2)

fib_with_cache(50)
```

LAZY IMPORTS

```python
import my_huge_library

def do_something():
  my_huge_library.calculate()

def do_something_else():
  dont.use.huge.library()
```

```python
def do_something():
  import my_huge_library

  my_huge_library.calculate()

def do_something_else():
  dont.use.huge.library()
```

LEARN ITERTOOLS

itertools module has many built-in functions for common tasks that are fast

- **chain** Concatenate lists/iterators without having to actually concatenate them
- **product** Get the cartesian product of two or more lists/iterators
- **permutations** Get all possible orderings of a list of elements

PYTHON MULTIPROCESSING

Try the **multiprocessing** library.

Not **threading**! (GIL)

```
from multiprocessing import Process

def f(name):
  print('hello', name)

if __name__ == '__main__':
  p = Process(target=f, args=('bob',))
  p.start()
  p.join()
```

TRY PYPY

- Implements Python 3.7.10
- Behaves like regular Python
- Is way faster
- Might use less memory
- Should work with most Python code

This turns python-code into optimized byte-code that can be run directly:

```
>>> import dis
>>> def f(x):
...     return x + 1
>>> dis.dis(f)
2           0 LOAD_FAST                0 (x)
            3 LOAD_CONST               1 (1)
            6 BINARY_ADD
            7 RETURN_VALUE
```

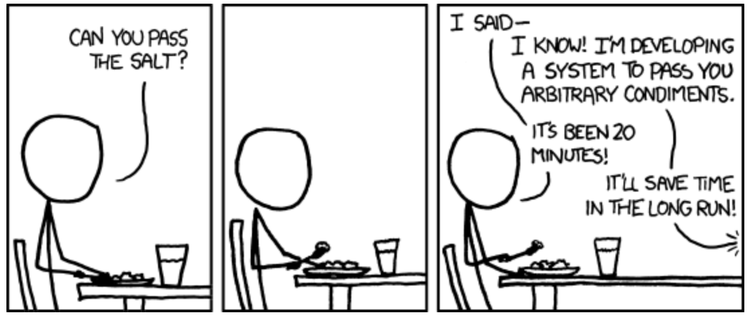
TRY C!

Maybe numpy or another C Python library does what you want.
It's not that difficult to extend Python with C code

```c
#include <Python.h>

static PyObject *method_fputs(PyObject *self, PyObject *args) {
  char *str, *filename = NULL;
  int bytes_copied = -1;

  /* Parse arguments */
  if(!PyArg_ParseTuple(args, "ss", &str, &filename)) {
    return NULL;
  }

  FILE *fp = fopen(filename, "w");
  bytes_copied = fputs(str, fp);
  fclose(fp);

  return PyLong_FromLong(bytes_copied);
}
```

ALWAYS COMPARE

- Keep metrics from before and after a change
- Make sure you actually improved something
- Focus on big gains
- Don't overoptimize. Is saving 0.01s worth losing readability?



QUESTIONS?

SDSC