

Making Python code go *fast*

Identifying bottlenecks, and most common ways to make something faster

Preemptive optimisation is the root of all evil

Contents

- Profiling
- How to make code faster
- Workshop

Identifying bottlenecks, a.k.a. profiling

Quick

```
from timeit import timeit
```

```
total_time = timeit("fib(30)", number=iterations, globals=globals())
```

Serious

```
python -m cProfile -o output_file -s cumulative myscript.py
```

```
snakeviz output_file.prof
```

Fundamentals of going faster

- Make each step faster
 - Numba / JIT
 - Caching
 - Big brain coding
- Share running of steps across many different workers
 - Parallelism
 - Concurrency

Make each step go faster

Numba + JIT

- Pay more the very first time to pay *a lot* less every time after that

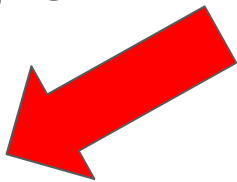
```
def go_fast(a):  
    # Using lots of numpy function  
    # Don't use other libraries
```

Numba + JIT

- Pay more the very first time to pay *a lot* less every time after that

```
from numba import jit
```

```
@jit(nopython=True)
```



```
def go_fast(a):
```

```
    # Using lots of numpy function
```

```
    # Don't use other libraries
```


Caching

- Save outputs of commonly called functions

```
def fib(n):  
    if n == 0:  
        return 0  
    if n == 1:  
        return 1  
    return fib(n-1) + fib(n-2)
```

Caching

- Save outputs of commonly called functions

```
from functools import lru_cache
```

```
@lru_cache(maxsize=None)
```



```
def fib(n):
```

```
    if n == 0:
```

```
        return 0
```

```
    if n == 1:
```

```
        return 1
```

```
    return fib(n-1) + fib(n-2)
```

Big brain coding

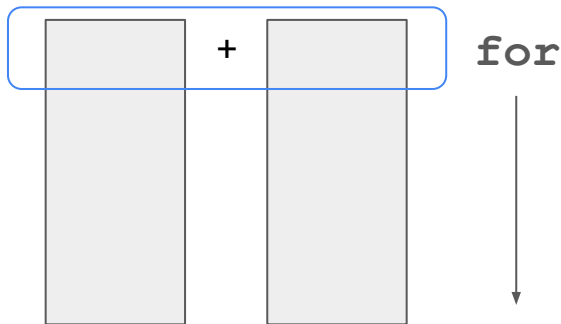
- Totally outside scope of this clinic
- Python wrappers on C++
- Random knowledge you accumulate as you code
 - Python 3.11 is faster than 3.10
 - Better to work in a manner that performs functions on memory already in CPU cache
 - Avoid sequential pandas concatenations
 - etc.

Distribute steps

Parallelism

- 'Free' parallelism
 - Numpy using SIMD instructions
 - Other C-backed libraries

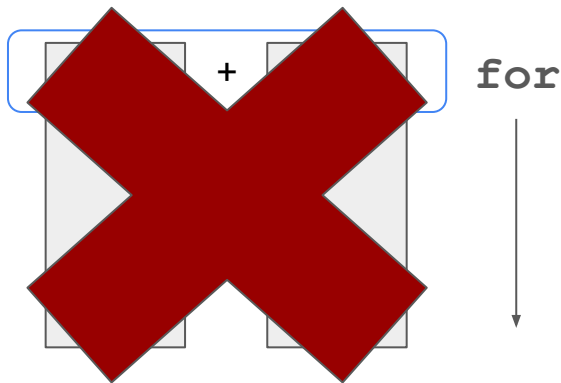
No fancy code - if using numpy aim to use library functions and think in element-wise operations where possible.



Parallelism

- 'Free' parallelism
 - Numpy using SIMD instructions
 - Other C-backed libraries

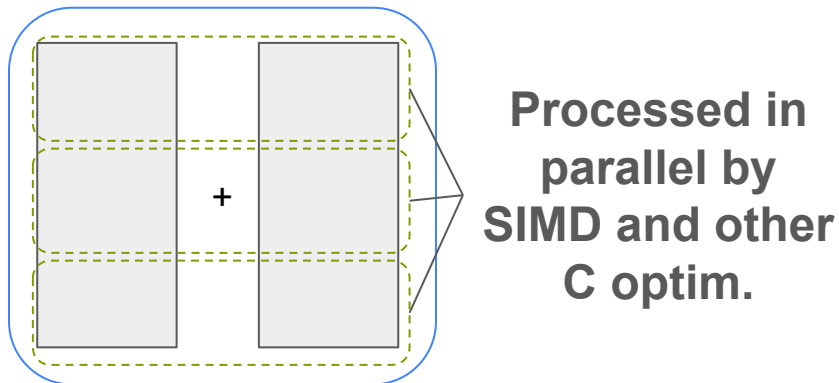
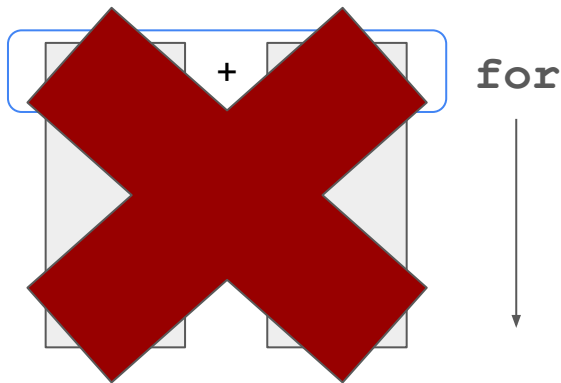
No fancy code - if using numpy aim to use library functions and think in element-wise operations where possible.



Parallelism

- 'Free' parallelism
 - Numpy using SIMD instructions
 - Other C-backed libraries

No fancy code - if using numpy aim to use library functions and think in element-wise operations where possible.



Parallelism

- Manual parallelism
 - `multiprocessing` library
 - In general can also do multi-threading but this doesn't work with python

```
results = []
```

```
for el in iterable:
```

```
    results.append(func(el))
```


Parallelism

- Manual parallelism
 - `multiprocessing` library
 - In general can also do multi-threading but this doesn't work with python

```
from multiprocessing import Pool
```

```
with Pool(threads) as p:
```

```
    results = p.map(func, iterable)
```

Concurrency

- Outside scope of today
 - In reality a bit easier than parallelism
- Do stuff while CPU is waiting for data to be read/received from disk or network