door code:
2-3-5

Introduction

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University

Waitlist: see me
after class

Course Website

engr315.github.io

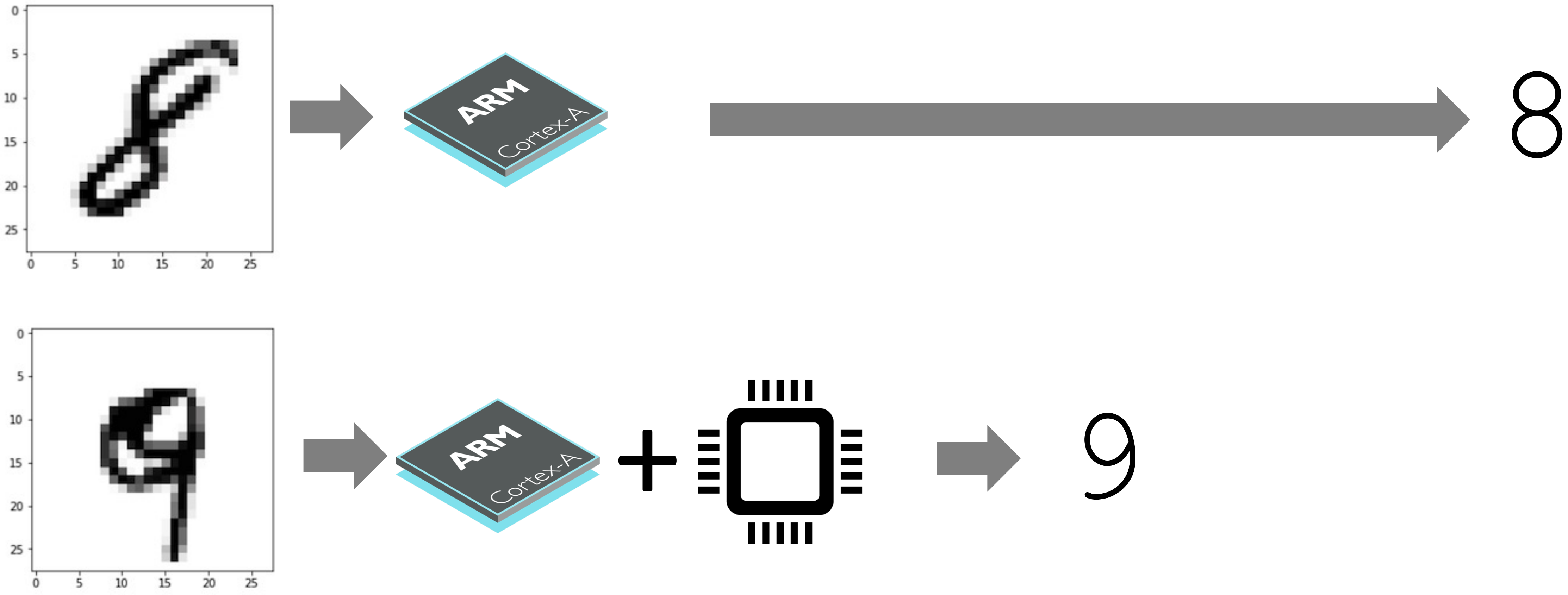
Write that down!

WARNING: Know Thy Foo()

- Python – First few weeks
- C – 2nd ish week onward
- Verilog – 3nd week onward

If you are not familiar with these, see me after class!

The goal



This class is ***NOT*** about computing.

This class is ***NOT*** about computing.

It's about computing ***FAST.***

How can we make our computation FAST(er)?

Better HW
- parallel
- clock speed

- optimize code (PI)
- reduce work

- cache
- pipelining

How can we make our computation FAST?

- Do less work.
- Do work faster. → don't work anymore
- Do work in parallel.

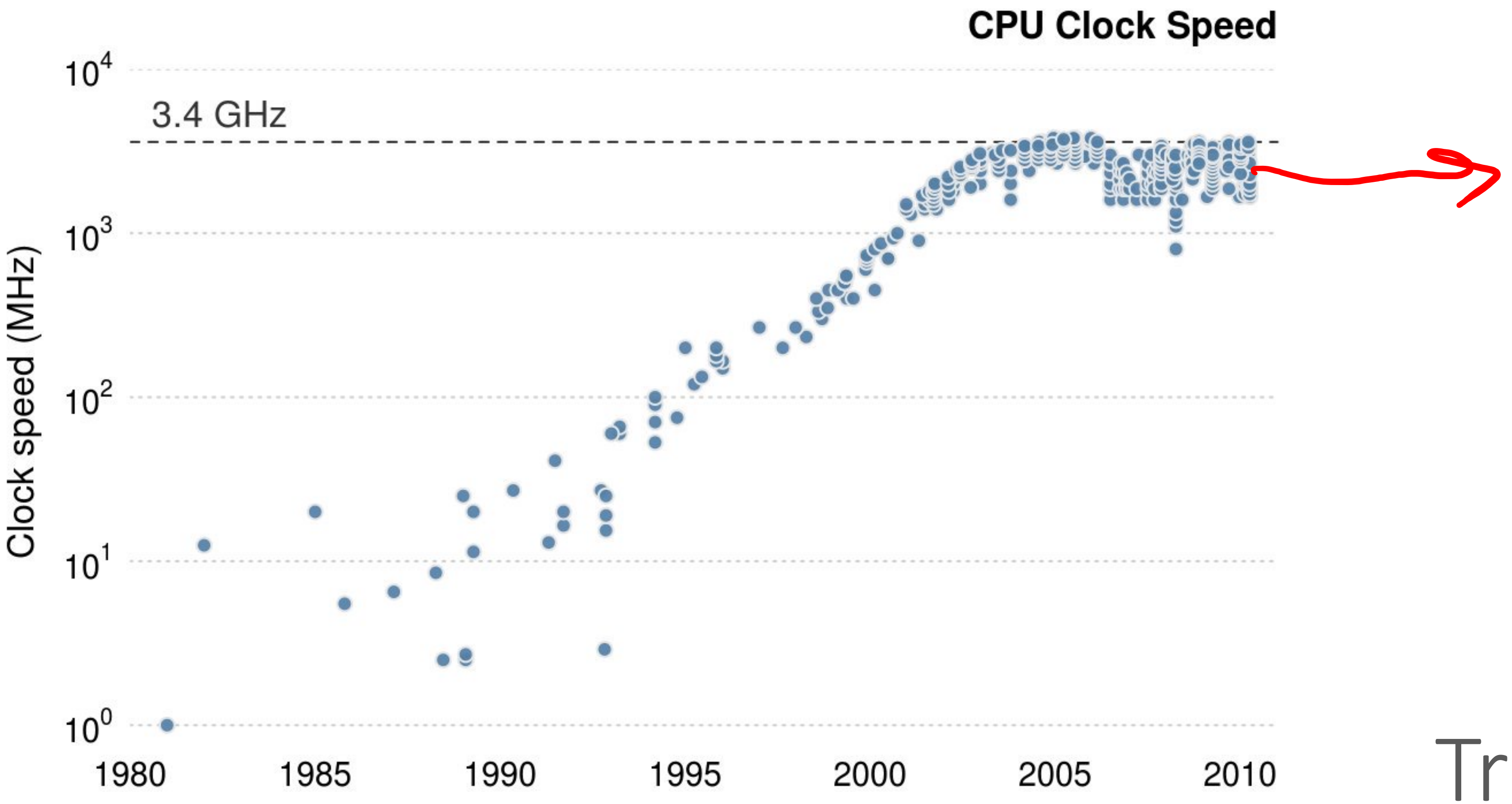
P1

Doing less work?

- Algorithmic complexity
- Languages:
  - Python vs. C++ vs. C/ASM
- Optimizing compiler
  - `gcc -O3`

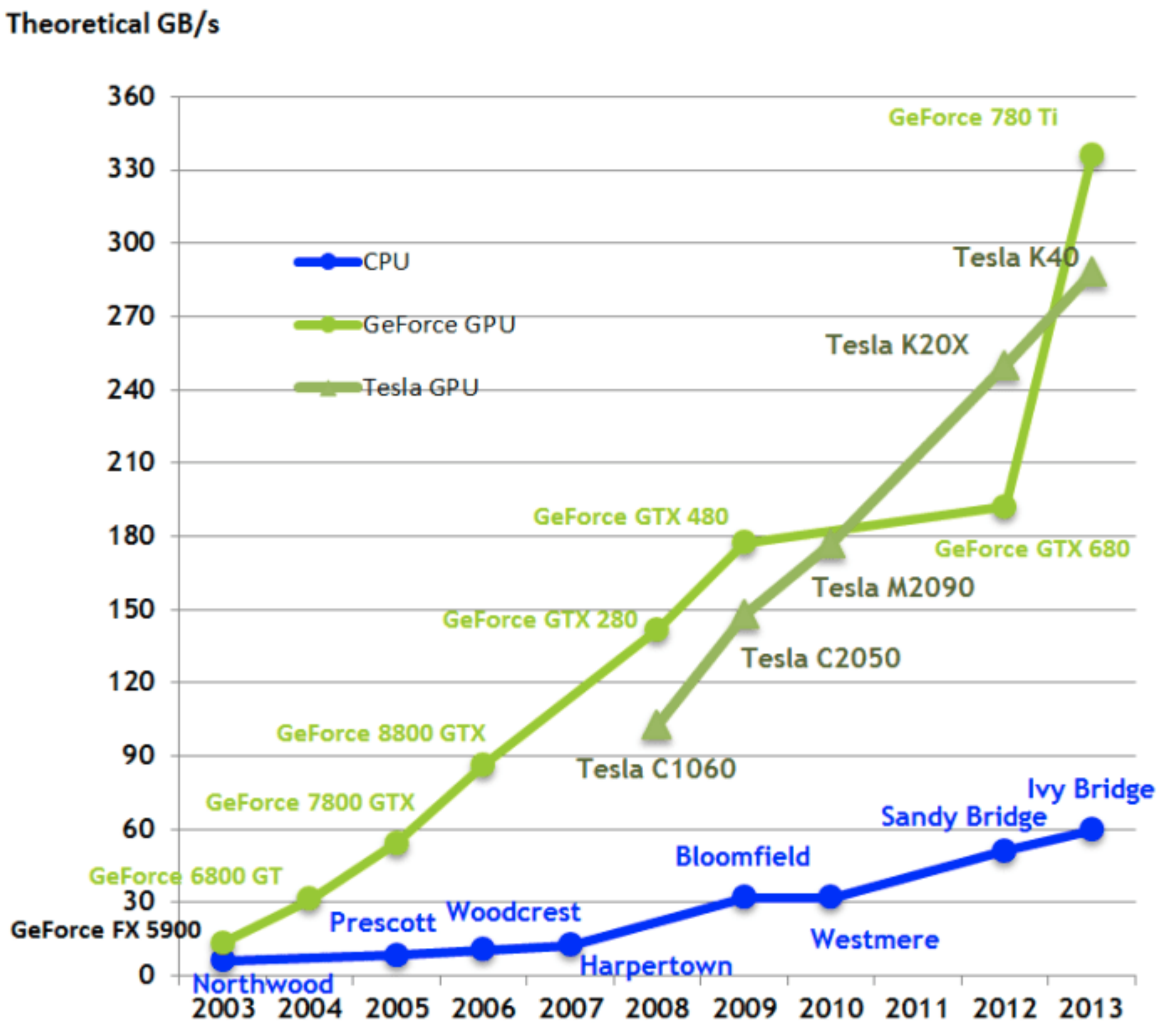
Yep. What else?

Do work faster?



Tried it. Next?

Do work in parallel?



When it works,
it really works!

How to do work in parallel?

The primary goal of this class is:

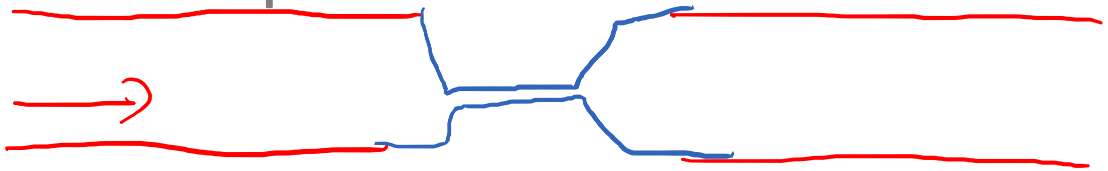
Learn methods to accelerate applications

Especially using hardware!

The secondary goals of this class are:

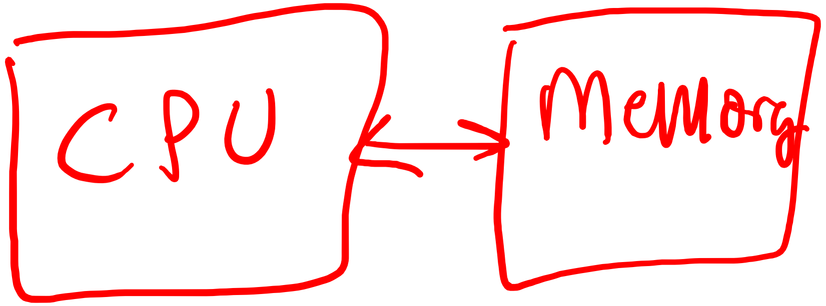
- Find performance bottlenecks in applications

pipe

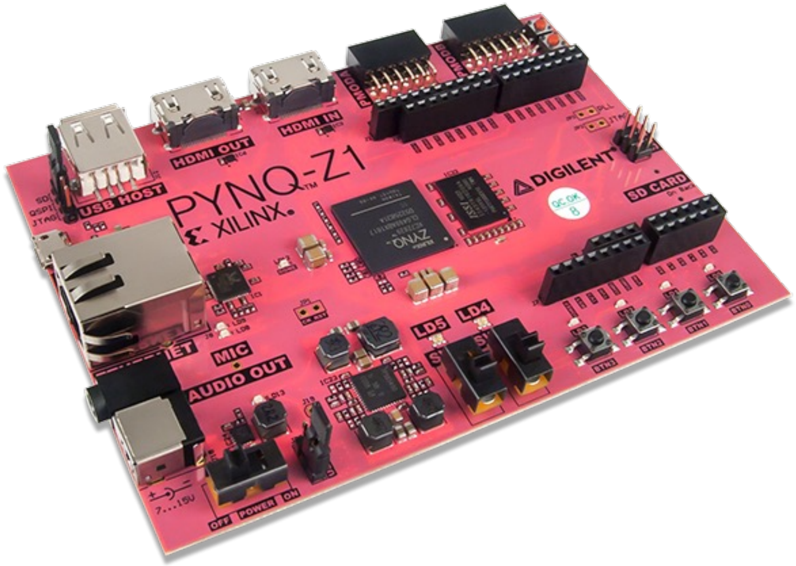


- Accelerate applications using parallelization

- Learn computer systems architectures!



We’ll be using the Pynq-Z1



- System-on-Chip
  - SoC - “S-O-C” or “Sock”
- Contains both FPGA and CPU
- Runs Linux
- http://www.pynq.io/

E315 assignments are all “optimizations”

- I give you a working software version.
- You need to:
  a) Make it go faster
  b) Make it run on hardware
  c) (usually) both

About Me

Andrew Lukefahr, Assistant Professor

Office: 2032 Luddy Hall

Email: lukefahr@Indiana.edu

Office Hours: See Syllabus

Research work on security for FPGA-based systems.

Email

- I treat email as “e”-mail, not instant massaging
- I bulk respond ~1 time / day. Sometimes ~1 time / 2 days.

Slack

- Can someone set this up? And add me?

TA's

Joe

WARNING: Know Thy Foo()

- Python – First few weeks
- C – 2^{nd} week onward.
- Verilog – 3^{nd} week onward.

If you are not familiar with these, see me after class!

Course Website

engr315.github.io

post lecture slides after class

Write that down!

Performance Profiling

How long does your code take to run?

Squared Values

```
def squares(n):
    if n <= 1:
        return [1]
    else:
        seq = squares(n-1)
        seq.append(n*n)
        return seq
```

```
for i in range(1,10):
    print (squares(i))
```

```
[1]
[1, 4]
[1, 4, 9]
[1, 4, 9, 16]
[1, 4, 9, 16, 25]
[1, 4, 9, 16, 25, 36]
[1, 4, 9, 16, 25, 36, 49]
[1, 4, 9, 16, 25, 36, 49, 64]
[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Measuring Execution Time

XXX. YYY ZZZ AAA

10^{-3} 10^{-6} 10^{-9}

milli micro Nano

ms μs ns

ms us ns

```python
import time

start_time = time.time()
squares(10)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.000107 seconds
```

107 μSec

Measuring Execution Time

```
import time
import sys
sys.setrecursionlimit(21000)

start_time = time.time()
squares(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.009825 seconds
```

~ 10 mSec

How do we ***reduce*** that time?

No recursion

less work

```
def squares(n):
    if n <= 1:
        return [1]
    else:
        seq = squares(n-1)
        seq.append(n*n)
        return seq
```

dont append, pre-size

How would we know *what* to optimize?

Code Profiling

- In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization. [Wiki]

Profilers give us call-stack information about where the program is spending its time.

```python
import cProfile
cProfile.run('squares(20000)')
```

```
         40002 function calls (20003 primitive calls) in 0.021 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  20000/1    0.019    0.000    0.021    0.021 <ipython-input-8-50d13c5dd8df>:1(squares)
        1    0.000    0.000    0.021    0.021 <string>:1(<module>)
        1    0.000    0.000    0.021    0.021 {built-in method builtins.exec}
    19999    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
```

ncalls: the total number of calls made to a function

tottime: the total time taken by all calls to a function

percall: time per function call (tottime / ncalls)

cumtime: total time spend in this and sub-functions

percall: total cumulative time / total time

filename:lineno (function): The name of the python function

What does this tell us?

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  20000/1    0.019    0.000    0.021    0.021 <ipython-input-8-50d13c5dd8df>:1(squares)
        1    0.000    0.000    0.021    0.021 <string>:1(<module>)
        1    0.000    0.000    0.021    0.021 {built-in method builtins.exec}
    19999    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Now, how do we *reduce* that time?

```python
def squares(n):
    if n <= 1:
        return [1]
    else:
        seq = squares(n-1)
        seq.append(n*n)
        return seq
```

for i in range(n):
    seq.append(i·i)

```
        40002 function calls (20003 primitive calls) in 0.021 seconds

  Ordered by: standard name

  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 20000/1    0.019    0.000    0.021    0.021 <ipython-input-8-50d13c5dd8df>:1(squares)
       1    0.000    0.000    0.021    0.021 <string>:1(<module>)
       1    0.000    0.000    0.021    0.021 {built-in method builtins.exec}
   19999    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
       1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Can we cut the recursion?

```
def squares(n):
    if n <= 1:
        return [1]
    else:
        seq = squares(n-1)
        seq.append(n*n)
        return seq
```

```
def squares2(n):
    if n <= 1:
        return [1]
    else:
        seq = []
        for i in range(1,n):
            seq.append(i*i)
        return seq
```

+1

```python
import time
import sys
sys.setrecursionlimit(21000)

start_time = time.time()
squares(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.009825 seconds
```

```python
import time

start_time = time.time()
squares2(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.004209 seconds
```

0.009825/0.004209 = 2.33

2.33x Faster!

Why was it faster?

```
import cProfile
cProfile.run('squares2(20000)')
```

```
         20003 function calls in 0.007 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.005    0.005    0.006    0.006 <ipython-input-21-5c6731cb3b0c>:1(squares2)
        1    0.000    0.000    0.007    0.007 <string>:1(<module>)
        1    0.000    0.000    0.007    0.007 {built-in method builtins.exec}
    19999    0.002    0.000    0.002    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

What’s missing?

Conclusion #1: Overheads to function calls!

Can we make it go even faster?

```
def squares2(n):
    if n <= 1:
        return [1]
    else:
        seq = []
        for i in range(1,n):
            seq.append(i*i)
        return seq
```

pre-size array

```
import time

start_time = time.time()
squares2(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.004209 seconds
```

Is there a way to remove list.append()?

Can we make it go even faster?

list comprehensions?

```
def squares2(n):
    if n <= 1:
        return [1]
    else:
        seq = []
        for i in range(1,n):
            seq.append(i*i)
        return seq
```

```
import time

start_time = time.time()
squares2(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.004209 seconds
```

```
import numpy as np
def squares3(n):

    seq = np.zeros(n, dtype=np.int)
    for i in range(1, n+1):
        seq[i-1] = i * i
    return seq
```

```
import time

start_time = time.time()
squares3(20000)
end_time = time.time()

# at the end of the program:
print("%f seconds" % (end_time - start_time))
```

```
0.003960 seconds
```

```python
import cProfile
cProfile.run('squares3(20000)')
```

```
         5 function calls in 0.005 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.005    0.005    0.005    0.005 <ipython-input-68-7272dceb0678>:2(squares3)
        1    0.000    0.000    0.005    0.005 <string>:1(<module>)
        1    0.000    0.000    0.005    0.005 {built-in method builtins.exec}
        1    0.000    0.000    0.000    0.000 {built-in method numpy.zeros}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Next Time

- More on Profiling!

Introduction

Engr 315: Hardware / Software Codesign

Andrew Lukefahr
Indiana University