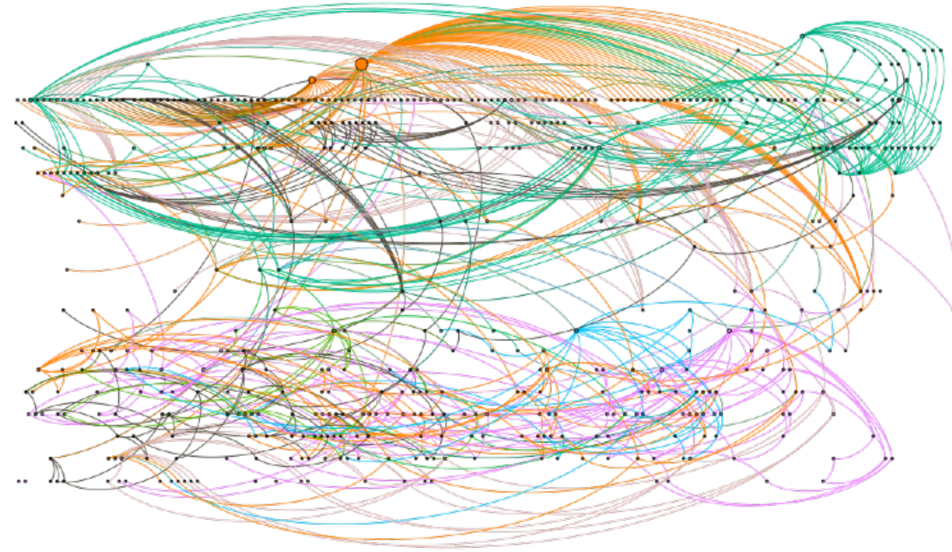
or:

How I learned to stop worrying and love distributed Python



Australia's National Science Agency

Why do *we* like/dislike Python?

- Go to: www.menti.com
- Use code 1999 0644

Go to **www.menti.com** and use the code **1999 0644**

What do you *like* about Python?

Press **S** to show image

Mentimeter

Go to **www.menti.com** and use the code **1999 0644**

What do you *dislike* about Python?

Press **S** to show image

Mentimeter

13

What do I like about Python?

- Syntactically elegant
- Interpreted and interactive
- Huge number of amazing libraries!

What do I dislike about Python?

- One line solutions
- ***Native*** performance
    - Iteration and built-in maths is slow (compared to compiled to compiled languages)
    - Parallelisation…

```
(lambda
pygame=__import__('pygame'),random=__import__('random'),WIDTH=800,HEIGHT=600,BOARD_SIZE=40,SNAK
E_SIZE=3:(pygame.init(),(lambda win=pygame.display.set_mode((WIDTH,HEIGHT)),draw_square=(lambda
window,color,x,y,k=WIDTH/BOARD_SIZE:pygame.draw.rect(window,color,(int(k*x),HEIGHT-
int(k*y),int(k),int(k)))):(lambda Snake=type("Snake",(),{"__init__":lambda
self,x,y:self.__dict__.update({'x':x,'y':y,'direction':0,'body':[],'add_tail':0,'color':
(0,255,0)}),"set_direction":lambda
self,direction:self.__dict__.update({'direction':direction}),"move":lambda self:None if
self.direction==0 else(self.body.insert(0,(self.x,self.y)),self.body.pop()if self.add_tail==0
else self.__dict__.update({'add_tail':self.add_tail-1}),self.__dict__.update({'y':self.y+1}if
self.direction==1 else{'x':self.x+1}if self.direction==2 else{'y':self.y-1}if self.direction==3
else{'x':self.x-1}if self.direction==4 else{}))[0],"draw":lambda self:
(draw_square(win,self.color,self.x,self.y),[draw_square(win,self.color,b[0],b[1])for b in
self.body])[0]}),Fruit=type("Fruit",(),{"color":(255,0,0),"__init__":lambda
self,x,y:self.__dict__.update({'x':x,'y':y}),"draw":lambda
self:draw_square(win,self.color,self.x,self.y)}):(lambda board=type("Board",(),
{"width":BOARD_SIZE,"height":int(BOARD_SIZE*HEIGHT/
WIDTH),"score":0,"gameover":False,"__init__":lambda
self:self.__dict__.update({'snake':Snake(int(self.width/2),int(self.height/
2)),'fruit':Fruit(*self.get_fruit_position(has_snake=False))}),"update":lambda self:
(self.end_game()if not(0<=self.snake.x<self.width and
0<self.snake.y<=self.height)or(self.snake.x,self.snake.y)in self.snake.body else
((self.__dict__.update({'fruit':Fruit(*self.get_fruit_position()),'score':self.score+1}),self.s
nake.__dict__.update({'add_tail':self.snake.add_tail+SNAKE_SIZE}))[0]if
self.snake.x==self.fruit.x and self.snake.y==self.fruit.y else None,self.snake.move()if not
self.gameover else None)[0])if not self.gameover else None,"draw":lambda self:
(self.snake.draw(),self.fruit.draw())[0],"end_game":lambda self:
(self.snake.__dict__.update({'direction':0}),self.__dict__.update({"gameover":True}),print(f"sc
ore: {self.score}"))[0],"get_fruit_position":(lambda f:lambda x,**k:f(x,f,**k))(lambda
self,f,has_snake=True,x=random.randint(0,BOARD_SIZE-1),y=random.randint(1,int(BOARD_SIZE*HEIGHT
/WIDTH)-1):f(self,f,x=random.randint(0,BOARD_SIZE-1),y=random.randint(1,int(BOARD_SIZE*HEIGHT/
WIDTH)-1))if has_snake and ((x,y)in self.snake.body or (self.snake.x==x and
self.snake.y==y))else (x,y))})(),clock=pygame.time.Clock():(lambda update=(lambda:
(win.fill((0,0,0)),board.update(),board.draw(),pygame.display.update(),
[(pygame.quit(),__import__('sys').exit())if event.type==pygame.QUIT
or(event.type==pygame.KEYDOWN and event.key==pygame.K_SPACE)else(board.snake.set_direction(1)if
event.key==pygame.K_UP and board.snake.direction!=3 else board.snake.set_direction(2)if
event.key==pygame.K_RIGHT and board.snake.direction!=4 else board.snake.set_direction(3)if
event.key==pygame.K_DOWN and board.snake.direction!=1 else board.snake.set_direction(4)if
event.key==pygame.K_LEFT and board.snake.direction!=2 else None)if event.type==pygame.KEYDOWN
else None for event in pygame.event.get()],clock.tick(10))[0]):[_ for _ in iter(update,0)])())
())())()))()
```

https://github.com/tjf801/oneliners/blob/master/snake.py

Parallelisation

Why do we want parallelisation?

- Many tasks can be run independently!
- Can get a huge speed up in time to complete tasks

Embarrassingly parallel:

“Doing the same thing over and over but expecting different results”

Parallelisation

Why you probably don't want it

- Can be slower!
- Can introduce new and hard to diagnose bugs (race conditions)
- Your code is poorly optimised

```
a = 2
thread_one: a = a + 2
thread_two: a = a * 3
```

If thread_one runs first:

```
a = 2 + 2, a is now 4.
a = 4 * 3, a is now 12.
```

If thread_two runs first:

```
a = 2 * 3, a is now 6
a = 6 + 2, a is now 8
```

https://python.land/python-concurrency/the-python-gil

How to parallelise Python

Threads vs Processes

- A ‘process’ is a single program
- You can run multiple copies of a program doing different things (multiprocessing)
- ‘Threads’ are run *inside* a single processes
- Threads are understood by your OS
- Threads share memory, processes do not

The GIL

CPython has a ‘Global Interpreter Lock’

- Only a ***single*** Python thread can run at any time
- No race conditions!
- Some low-level parts of the language rely on the GIL

Some interpreters don’t have a GIL (PyPy, IronPython)

Scaling your code

Always try to scale ‘vertically’ first!

- Use Numpy, Numba, Cython etc. to run expensive computation on compiled code
- Numpy arrays are stored in memory. If you want to access bigger data, try to get a machine with more memory!

Scaling your code

If you're *sure* you can benefit from parallelisation

- Scale 'horizontally' (i.e. parallelise)
- Adding more cores - or more computers!
- 'Big data' is a great example of when horizontal scaling will help
- Try make your 'bite size' version first
- Make it fast!

A simple example

- Runs in about 0.5ms on my laptop

```python
def inc(x):
    return x + 1

def dec(x):
    return x - 1

def add(x, y):
    return x + y

zs = []
for i in range(1000):
    x = inc(i)
    y = dec(x)
    z = add(x, y)
    zs.append(z)
```

Borrowed from https://examples.dask.org/

A simple example - multiprocessing

- Using 16 processes on my laptop:
  - 200ms!

```
import multiprocessing as mp

def inc(x):
    return x + 1

def dec(x):
    return x - 1

def add(x, y):
    return x + y

def worker(i):
    x = inc(i)
    y = dec(x)
    z = add(x, y)
    return z

with mp.Pool(processes=mp.cpu_count()) as pool:
   zs = list(pool.map(worker, range(1000)))
```

A simple example - Numpy

- Runs in about 30µs

```python
import numpy as np

def inc(x):
    return x + 1

def dec(x):
    return x - 1

def add(x, y):
    return x + y

i = np.arange(1000)
x = inc(i)
y = dec(x)
z = add(x, y)
```

A (longer) simple example

- Only 10 iterations!
- Runs in about 15s on my laptop

```python
import time
import random

def inc(x):
    time.sleep(random.random())
    return x + 1

def dec(x):
    time.sleep(random.random())
    return x - 1

def add(x, y):
    time.sleep(random.random())
    return x + y

zs = []
for i in range(10):
    x = inc(i)
    y = dec(x)
    z = add(x, y)
    zs.append(z)
```

Multiprocessing?

- Only 10 iterations
- Runs in about 2s on my laptop
- `Pool` interface doesn't allow for inter-process communication
  - Need `Queue`/`Process` interface
- Difficult to distribute across multiple machines

```
import time
import random

def inc(x):
    time.sleep(random.random())
    return x + 1

def dec(x):
    time.sleep(random.random())
    return x - 1

def add(x, y):
    time.sleep(random.random())
    return x + y

def worker(i):
     x = inc(i)
     y = dec(x)
     z = add(x, y)
     return z

with mp.Pool(processes=mp.cpu_count()) as pool:
   zs = list(pool.map(worker, range(10)))
```

MPI4py?

- Message Passing Interface (MPI)
- Probably installed on your favourite HPC/Supercomputer
- Only 10 iterations
- Runs in about 2s on my laptop
- Need to invoke:
  - `mpirun –n 16 python example_mpi.py`

```python
import time
import random
from mpi4py import MPI

def inc(x):
    time.sleep(random.random())
    return x + 1

def dec(x):
    time.sleep(random.random())
    return x - 1

def add(x, y):
    time.sleep(random.random())
    return x + y

comm = MPI.COMM_WORLD
nPE = comm.Get_size()
myPE = comm.Get_rank()
dims = 10
local_zs = []

if nPE > dims:

    my_start = myPE
    my_end = myPE

else:
    count = dims // nPE
    rem = dims % nPE

    if myPE < rem:
        # The first 'remainder' ranks get 'count + 1' tasks each
        my_start = myPE * (count + 1)
        my_end = my_start + count
    else:
        # The remaining 'size - remainder' ranks get 'count' task each
        my_start = myPE * count + rem
        my_end = my_start + (count - 1)

for i in range(my_start, my_end + 1):
    x = inc(i)
    y = dec(x)
    z = add(x, y)
    local_zs.append(z)

zs = comm.gather(local_zs, root=0)
if myPE == 0:
    zs = [item for sublist in zs for item in sublist]
```

Yuck

- We have to re-write our core algorithms
- Lose the Pythonic syntax we love
- Spending time/effort on parallelisation - Not the *actual* task
  - Scattering
  - Gathering
  - Resource management

CSIRO

Yuck

- We have to re-write our core algorithms
- Lose the Pythonic syntax we love
- Spending time/effort on parallelisation - Not the *actual* task
    - Scattering
    - Gathering
    - Resource management

There has to be a better way…

Dask

- Open source library
- Designed to scale Python the way you like to write it
- dask.org

Dask

Collections

(create task graphs)

Schedulers

(execute task graphs)

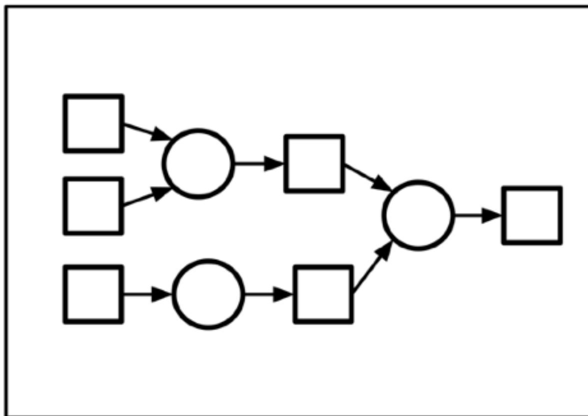
Dask Array

Dask DataFrame

Dask Bag

Dask Delayed

Futures



Single-machine

(threads, processes, synchronous)

Distributed

Dask and astronomy

- Astropy has growing support for Dask - Development underway!
- FITS Cubes:
  - Spectral-Cube (high-level): https://github.com/radio-astro-tools/spectral-cube
  - xarray-fits (low-level): https://github.com/ska-sa/xarray-fits
  - DA-FITS (low-level): https://github.com/AlecThomson/da-fits
- Measurement Sets:
  - Dask-MS https://github.com/ska-sa/dask-ms
  - ngCASA (in early development) - Currently written in ***pure*** python!
- Great talks from Dask Summit: https://youtu.be/xElpzGvr5UQ

Schedulers

Scale from you laptop to supercomputers

- LocalCluster - Single machine (laptop, server, HPC node)
- SSHCluster - Create your own supercomputer via ssh!
- Dask-mpi - Runs Dask via MPI
- Jobqueue - Submits job scripts for you via Slurm, PBS, etc
- Many more! https://blog.dask.org/2020/07/23/current-state-of-distributed-dask-clusters

- Only thing that changes is the `cluster` object definition
    - The rest of your code remains the same

Collections - Live examples

Questions?

- Give Dask a go in your browser! https://examples.dask.org/

CSIRO