Introduction

HPC Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

April 17, 2015

Why Python

- Easy!
- Fast development
- Nice, readable code
- Great for prototyping
- Many third party libraries
- Data analysis
- Many users!

Data Types

Dynamic language, but also a strongly typed language

- Objects have a type, which is determined at runtime
- A variable is a value bound to a name: the value has a type, but the variable doesn't
- The interpreter keeps track of all variable types
- You can't do anything that's incompatible with the type of data you're working with:
  - You can do 'string+string' and it will concatenate the strings
  - You can do 'integer+integer'
  - You can't do 'string+integer'

Data Structures

Python List

- Dynamic arrays
- Indexed structure
- Items: Python objects
- Items of different types
- Insertion and deletion at random positions

Data Structures

Dictionary

- Associative arrays (key - value pairs)
- Indexed by key (string or number)
- Key: unique
- Value: any Python object
- Main operation: store a value with some key and extract the value given the key

Python in HPC

- You'll hear that Python is slow
  - Is it Python or your code?
  - Remember: Python is easy $\rightarrow$ bad programming
- If it's slow, why should you use it?
- If you already have a Python code, what should you do?

Python in Stampede

- python/2.7.3-epd-7.3.2 (deprecated)
- python/2.7.6
- python/2.7.9

You can install your own modules

1. `python setup.py install --user`
2. `python setup.py install --home=<dir>`
3. `pip install --user module_name`

- You can use virtualenv

Before We Begin

Connect to Stampede

```
ssh -Y trainXXX@stampede.tacc.utexas.edu
```

Python Exercises

```
cp ~train00/python-hpc.tar.gz .
tar -xzf python-hpc.tar.gz
module load intel/14.0.1.106
module load python/2.7.6
idev -t 2:00:00
```

Profiling

`python -m cProfile [-o output_file] [-s sort_order] script.py`

examples/1_intro/test_prof1.py

```
1  import math
2  def function(arg):
3      res = []
4      for i in range(-10000000, 10000000):
5        res.append(math.sqrt(abs(i+1)*arg**5))
6      return res
7
8  print sum(function(10.0))
```

```
1.33333333332e+13

60000004 function calls in 17.351 seconds

Ordered by: standard name

ncalls  tottime  percall  cumtime  percall
     1    0.141    0.141   17.351   17.351 test_prof1.py:1
     1   12.835   12.835   17.039   17.039 test_prof1.py:2
20000000   1.312    0.000    1.312    0.000 {abs}
20000000   1.429    0.000    1.429    0.000 {math.sqrt}
20000000   1.463    0.000    1.463    0.000 {'append'}
     1    0.000    0.000    0.000    0.000 {'disable'}
     1    0.171    0.171    0.171    0.171 {sum}
```

Profiling

line profiler

- Installation

```
pip install line_profiler --user
export PATH=$PATH:$HOME/.local/bin
```

- Add `@profile` decorators to the functions you want to profile

examples/1_intro/test_prof1b.py

```
1  from builtins import profile
2  import math
3  @profile
4  def function(arg):
5      res = []
6      for i in range(-10000000, 10000000):
7        res.append(math.sqrt(abs(i+1)*arg**5))
8      return res
9
10 print sum(function(10.0))
```

Run it

```
kernprof -l -v test_prof1b.py
```

Profiling

examples/1_intro/test_prof2.py

```
1  from builtins import profile
2  import math
3  @profile
4  def function(arg):
5      res = []
6      factor = arg**5
7      for i in xrange(-10000000, 10000000):
8        if i<0:
9          res.append(math.sqrt((-i+1)*factor))
10       else:
11         res.append(math.sqrt((i+1)*factor))
12     return res
13
14 print sum(function(10.0))
```

Run it

```
kernprof -l -v test_prof2.py
```

Profiling

examples/1_intro/test_prof3.py

```
1  from builtins import profile
2  import math
3  @profile
4  def function(arg):
5    factor = arg**5
6    res = range(-10000000, 10000000)
7    res = map(abs, res)
8    res = [(x+1)*factor for x in res]
9    res = map(math.sqrt, res)
10   return res
11
12 print sum(function(10.0))
```

Run it

```
kernprof -l -v test_prof3.py
```

License