Debunking Myths and Rumors of High Performance Python

Guillermo Cornejo-Suárez

HPC School Costa Rica
Centro Nacional de Alta Tecnología

February 2nd, 2018

About me

Research assistant @ CNCA-CeNAT since August 2016.

- Academics

  2016 Bachelor in Electrical Engineering from UCR.

  2020 Master degree in Computer Science from TEC.

- HPC *skills*
  - High Performance Python.
- Research interests
  - Computational Seismology.
  - Earth Sciences.

Introduction

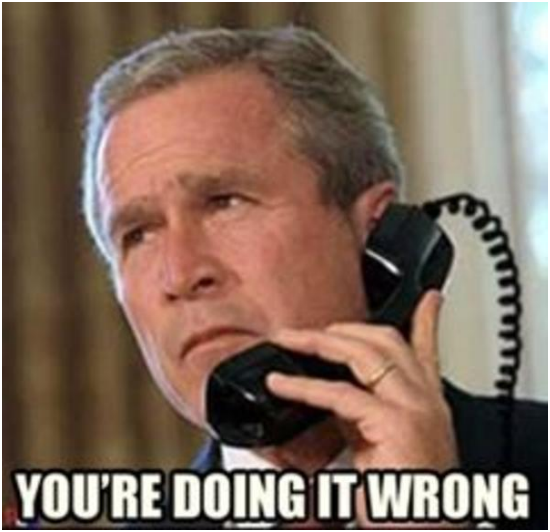
An oxymoron

- Lonely Together (Avicii feat. Rita Ora)
- Acompáñame a estar solo (Ricardo Arjona)
- Virtual reality
- ”A joke is actually an extremely really serious issue.” - Winston Churchill

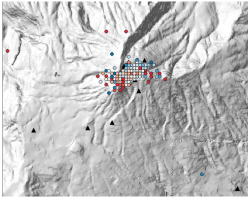
An oxymoron

- Lonely Together (Avicii feat. Rita Ora)
- Acompáñame a estar solo (Ricardo Arjona)
- Virtual reality
- ”A joke is actually an extremely really serious issue.” - Winston Churchill
- High Performance Python

High Performance Python



Why trying it out anyway?



Why trying it out anyway?

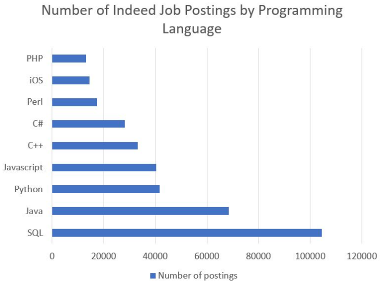
Solution	Estimated time
Teach C programming to Leo	A few months
Translating Leo's program to C	A few weeks
Make Leo's program parallel with mpi4py	less than a week

Python is pretty popular

Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C		99.7
3. Java		99.5
4. C++		97.1
5. C#		87.7
6. R		87.7
7. JavaScript		85.6
8. PHP		81.2
9. Go		75.1
10. Swift		73.7

Worldwide, Jan 2018 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.76 %	-1.3 %
2		Python	20.76 %	+5.4 %
3		PHP	8.7 %	-1.8 %
4	↑	Javascript	8.49 %	+0.3 %
5	↓	C#	7.99 %	-0.8 %



1 JavaScript
2 Java
3 Python
4 PHP
5 C#

Jan 2018	Jan 2017	Change	Programming Language	Ratings	Change
1	1		Java	14.215%	-3.06%
2	2		C	11.037%	+1.69%
3	3		C++	5.603%	-0.70%
4	5	^	Python	4.678%	+1.21%


Basic

Myths

It cannot run as fast as C

Myth 1: It cannot run as fast as C

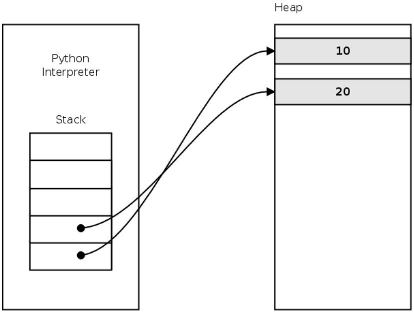
```
### test.py
a = 10
b = 20
c = a + b
### end test.py
```

```
$ python -m dis test.py
```

```
1    0 LOAD_CONST     0 (10)
     3 STORE_NAME     0 (a)

2    6 LOAD_CONST     1 (20)
     9 STORE_NAME     1 (b)

3   12 LOAD_NAME      0 (a)
    15 LOAD_NAME      1 (b)
    18 BINARY_ADD
    19 STORE_NAME     2 (c)
    22 LOAD_CONST     2 (None)
    25 RETURN_VALUE
```

Myth 1: It cannot run as fast as C

```
1    0 LOAD_CONST     0 (10)
     3 STORE_NAME     0 (a)

2    6 LOAD_CONST     1 (20)
     9 STORE_NAME     1 (b)

3   12 LOAD_NAME      0 (a)
    15 LOAD_NAME      1 (b)
    18 BINARY_ADD
    19 STORE_NAME     2 (c)
    22 LOAD_CONST     2 (None)
    25 RETURN_VALUE
```



Myth 1: It cannot run as fast as C

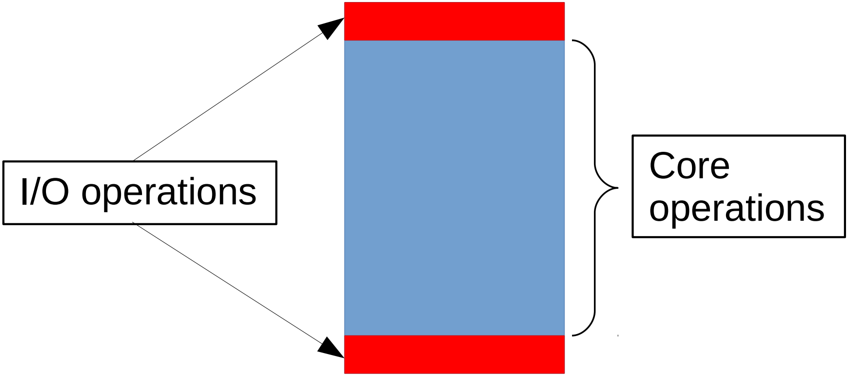
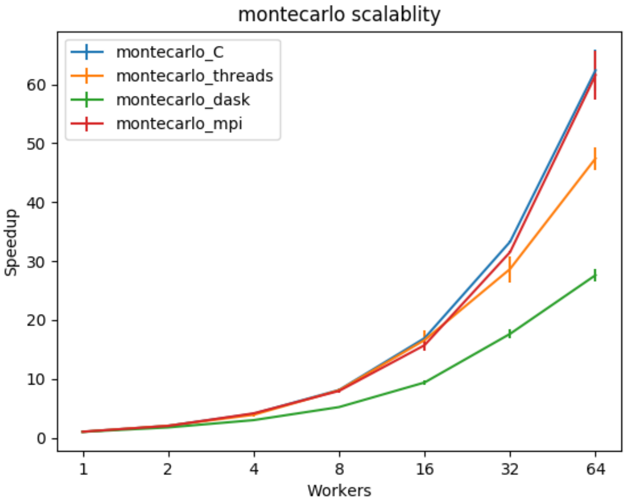


Figure: Common workload pattern in HPC

Myth 1:It cannot run as fast as C



Myth 1: It cannot run as fast as C

- Numpy
- Numba
- Wrappers

It doesn't support threads

Myth 2: It doesn't support threads

GIL

Global Interpreter Lock

Myth 2: It doesn't support threads

A mental experiment

```
def gauss(n):
    count = 0
    for i in range(n+1):
        count += 1

def matmul(n):
    m1 = np.empty((n,n))
    m2 = np.empty((n,n))
    m3 = m1.dot(m2)
```

Myth 2: It doesn't support threads

Secure | https://docs.scipy.org/doc/numpy/reference/c-api.array.html#threading-support

Threading support

These macros are only meaningful if `NPY_ALLOW_THREADS` evaluates True during compilation of the extension module. Otherwise, these macros are equivalent to whitespace. Python uses a single Global Interpreter Lock (GIL) for each Python process so that only a single thread may execute at a time (even on multi-cpu machines). When calling out to a compiled function that may take time to compute (and does not have side-effects for other threads like updated global variables), the GIL should be released so that other Python threads can run while the time-consuming calculations are performed. This can be accomplished using two groups of macros. Typically, if one macro in a group is used in a code block, all of them must be used in the same code block. Currently, `NPY_ALLOW_THREADS` is defined to the python-defined `WITH_THREADS` constant unless the environment variable `NPY_NOSMP` is set in which case `NPY_ALLOW_THREADS` is defined to be 0.

Group 1

This group is used to call code that may take some time but does not use any Python C-API calls. Thus, the GIL should be released during its calculation.

`NPY_BEGIN_ALLOW_THREADS`

Equivalent to `Py_BEGIN_ALLOW_THREADS` except it uses `NPY_ALLOW_THREADS` to determine if the macro if replaced with white-space or not.

`NPY_END_ALLOW_THREADS`

Equivalent to `Py_END_ALLOW_THREADS` except it uses `NPY_ALLOW_THREADS` to determine if the macro if replaced with white-space or not.

`NPY_BEGIN_THREADS_DEF`

Place in the variable declaration area. This macro sets up the variable needed for storing the Python state.

`NPY_BEGIN_THREADS`

Place right before code that does not need the Python interpreter (no Python C-API calls). This macro saves the Python state and releases the GIL.

`NPY_END_THREADS`

Place right after code that does not need the Python interpreter. This macro acquires the GIL and restores the Python state from the saved variable.

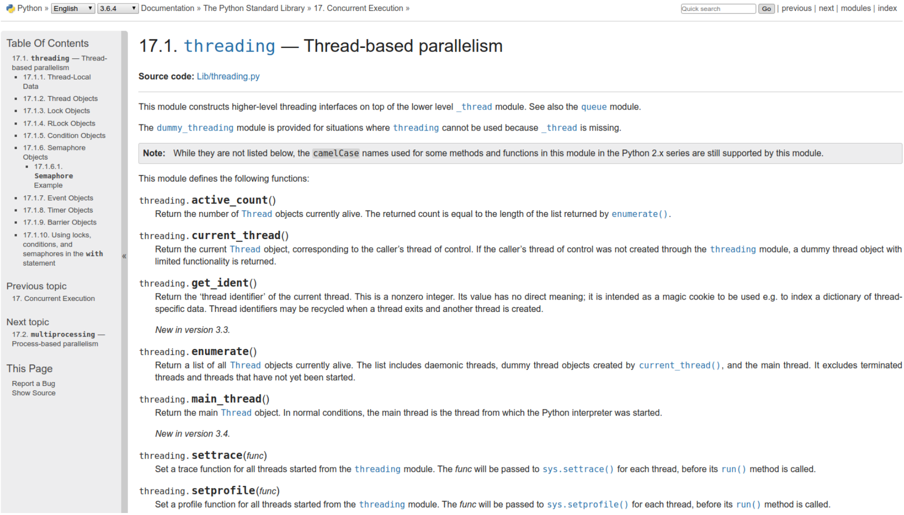
`NPY_BEGIN_THREADS_DESCR`(PyArray_Descr **dtype*)

Useful to release the GIL only if *dtype* does not contain arbitrary Python objects which may need the Python interpreter during execution of the loop. Equivalent to

`NPY_END_THREADS_DESCR`(PyArray_Descr **dtype*)

Useful to regain the GIL in situations where it was released using the BEGIN form of this macro.

Myth 2: It doesn't support threads



Myth 2: It doesn’t support threads

- Numpy and similar well-behaved modules.
- threading module

No one else is doing it

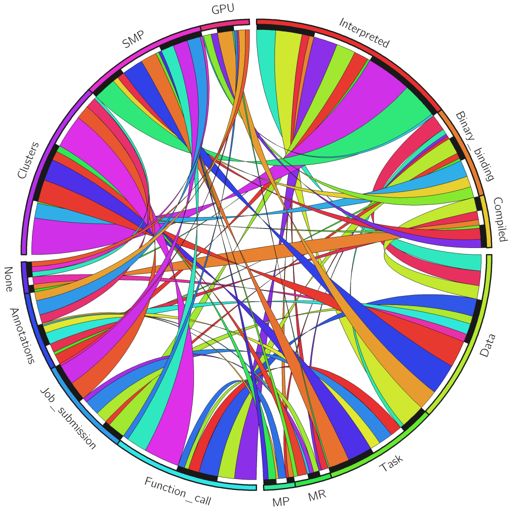
Myth 3: No one else is doing it

Myth 3: No one else is doing it

TABLE I
CLASSIFICATION OF TOOLS THAT PROVIDE PARALLELISM IN THE PYTHON PROGRAMMING LANGUAGE.

Project	Execution strategy	Parallel paradigm	Vector data oriented	Language support	Code modifications	Parallel platform	Latest release
Bohrium [14]	Interpreted	Data	Yes	Full, Python 2	None	SMP, GPU, Clusters	0.3, Apr-2016
PyStream [15]	Compiled	Data	Yes	Subset, Python 2	None	GPU	0.1, Jul-2011
Dask.array [16]	Interpreted	Data	Yes	Full, Python 3	FunCall	SMP, Clusters	0.13.0, Jan-2017
PupyMPI [17]	Interpreted	MsgPsg	No	Full, Python 2	FunCall	SMP, Clusters	0.9.5, May-2011
Papy [18]	Interpreted	Task	No	Full, Python 2	JobSub	SMP, Clusters	1.0.8, Nov-2014
GAiN [19]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	1.0, 2009
Global Arrays [20]	Binary binding	Data	Yes	Full, Python 2	FunCall	Clusters	5.5, Aug-2016.
mpi4Py [21]	Binary binding	MsgPsg	No	Full, Python 2-3	FunCall	SMP, Clusters	2.0.0, Oct-2015
Pythran [22]	Compiled	Data	Yes	Subset, Python 3	Annotations	SMP	0.7.6.1, Jul-2016
ASP [23]	Binary binding	Data, Task	No	Full, Python 2	JobSub	SMP, GPU	0.1.3.1, Oct-2013
Dispel4py [24]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.2, Jun-2015
PMI [25]	Interpreted	Data	No	Full, Python 2-3	FunCall	SMP, Clusters	1.0, Dec-2009
Jit4OpenCL [26]	Compiled	Data	Yes	Full, Python 2	Annotations	SMP, GPU	1.0, 2010
MRS [27]	Interpreted	MapRed	No	Full, Python 2-3	FunCall	Clusters	0.9, Nov-2012
Pydron [28]	Interpreted	Task	No	Subset	Annotations	Clusters	–
CoArray [29]	Interpreted	Data	Yes	Full, Python 2	FunCall	Clusters	2004
PyCuda, PyOpenCL [30]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU	2016.2, Oct-2016
SCOOP [31]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	0.7.1.1, Ago-2015
DistArray [32]	Interpreted	Data	Yes	Full, Python 2-3	JobSub	SMP, Clusters	0.6, Oct-2015
Dispy [33]	Interpreted	Data, MapRed	No	Full, Python 2-3	JobSub	SMP, Clusters	4.6.17, Sep-2016
IpyParallel [34]	Interpreted	Data, Task	No	Full, Python 2-3	JobSub	SMP, Clusters	5.3.0, Oct-2016
PyRo [35]	Interpreted	MsgPsg	No	Full, Python 2-3	Annotations, FunCall	Clusters	4.50, Nov-2016
Parallel python [36]	Interpreted	Task	No	Full, Python 2-3	JobSub	SMP, Clusters	1.6.5, Jul-2016
JUG [37]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	1.3.0, Nov-2016
Multiprocessing [38]	Interpreted	Task, Data	No	Full, Python 2-3	FunCall	SMP, Clusters	3.6, Jul-2016
Copperhead [39]	Binary binding	Data	Yes	Subset, Python 2	Annotations	GPU	2013
Celery [40]	Interpreted	Task	No	Full, Python 2-3	Annotations, FunCall	SMP, Clusters	4.0.0, Nov-2016
Disco [41]	Interpreted	MapRed	No	Full, Python 2	Annotations, FunCall	SMP, Clusters	0.5.4, Oct-2014
Spark [42]	Binary binding	Task	No	Full, Python 2-3	FunCall	Clusters	2.0.2, Nov-2016
Theano [43]	Binary binding	Data	Yes	Full, Python 2-3	FunCall	SMP, GPU, Clusters	0.8.2, Apr-2016
Numba [44]	Compiled	Data	Yes	Full, Python 2-3	Annotations	SMP, GPU	0.29.0, Oct-2016
Joblib [45]	Interpreted	Task	No	Full, Python 2-3	JobSub, Annotations	SMP	0.10.3, Oct-2016
Hadoopy [46]	Binary binding	MapRed	No	Full, Python 2	JobSub	Clusters	0.5.0, Jun-2012
PyMW [47]	Interpreted	Task	No	Full, Python 2	FunCall	Clusters	0.4, Jun-2010
Pyfora [48]	Compiled	Data	No	Subset, Python 2	None	Clusters, SMP	0.5.8, Set-2016

Myth 3: No one else is doing it



Myth 3: No one else is doing it

- (At least) 34 projects that provide parallelism in the Python programming language.

The Real Problem

Hiding complexity that doesn't go anywhere

Rumors

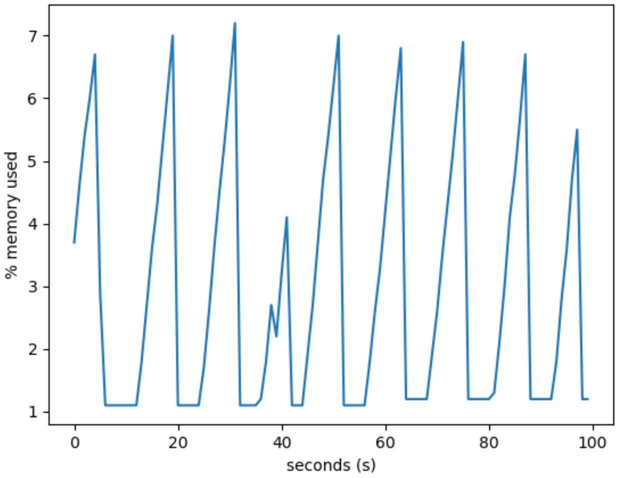
Dummy objects that screw up memory

Dummy objects that screw up memory

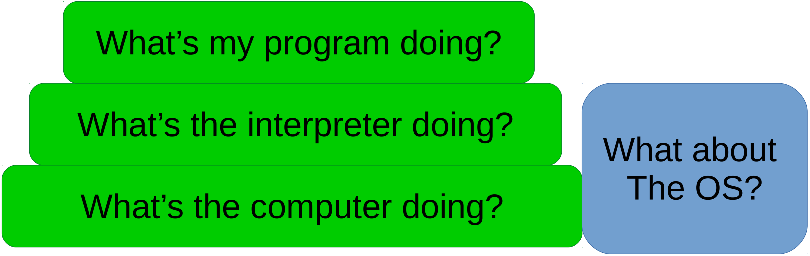
```
import numpy as np

n = 100000
rand = np.random.random(n*n)

count = 0
for i in range(n):
    count +=
      np.sum(rand[i*n:i*n+n])
```

```
import numpy as np

n = 100000

count = 0
for i in range(n):
    count +=
      np.sum(np.random.random(n))
```

Dummy objects that screw up memory

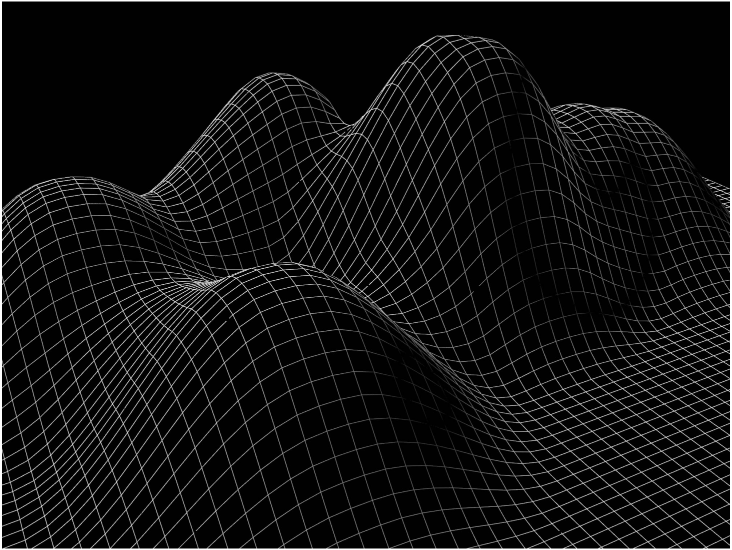


Dummy objects that screw up memory



Hey, use Numpy!

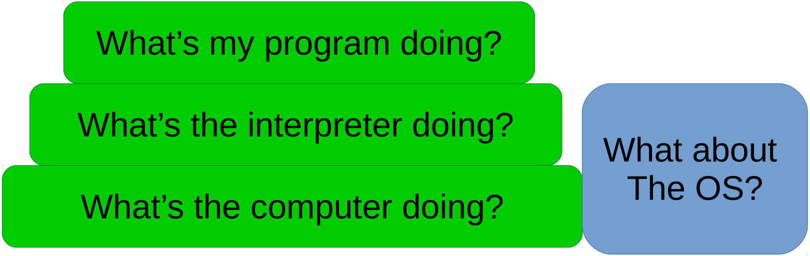
Hey, use Numpy!



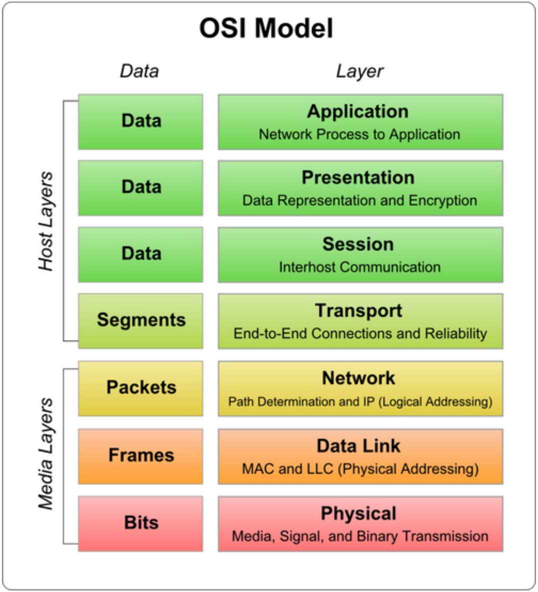
Hey, use Numpy!

```
def locate_events(events, stations):
    locations = []
    for event in events:
        min_err = math.inf
        for x in range(x_i, x_f+dx, dx):
            for y in range(y_i, y_f+dy, dy):
                for z in range(z_i, z_f+dz, dz):
                    for A in np.arange(A_i, A_f+dA, dA):
                        err_accum = 0
                        for s_k, s_v in stations.items():
                            r = math.sqrt(math.pow(x-s_v[0], 2) + math.pow
                            A_calc = A * math.exp(-B*r) / r
                            err_accum += math.pow(A_calc - event[s_k], 2)
                        if err_accum < min_err:
                            min_err = err_accum
                            loc = [event['event'], x, y, z, A, err_accum]
        A_obs = sum([math.pow(event[s], 2) for s in stations.keys()])
        loc[-1] = 100.0 * math.sqrt(loc[-1] / A_obs)
        locations.append(loc)
    return locations
```

Hey, use Numpy!

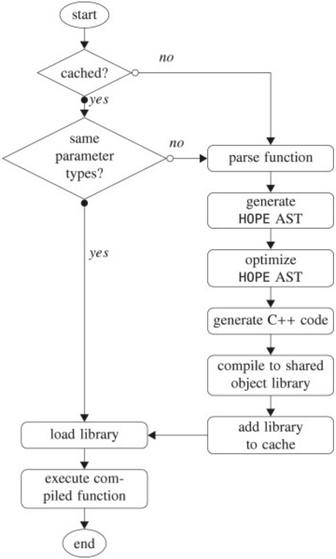


But... there's hope!


OSI Model
Data
Layer
Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Physical Addressing)
Bits
Physical
Media, Signal, and Binary Transmission

Emerging Technologies

Just In Time Compilers



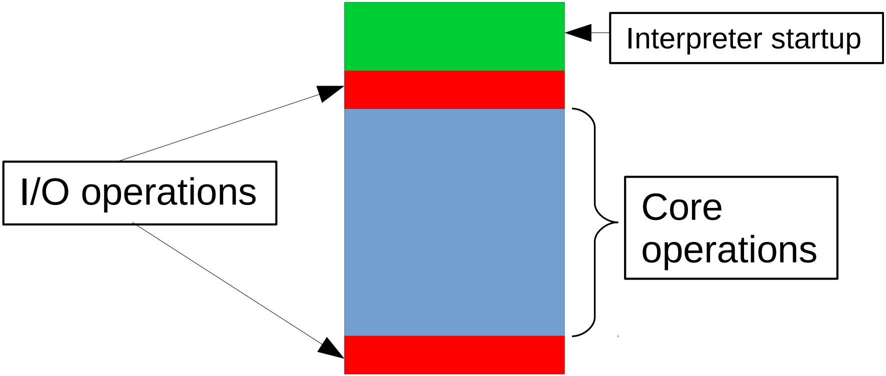
- HOPE
- Numba
- PyPy

Domain Specific Languages

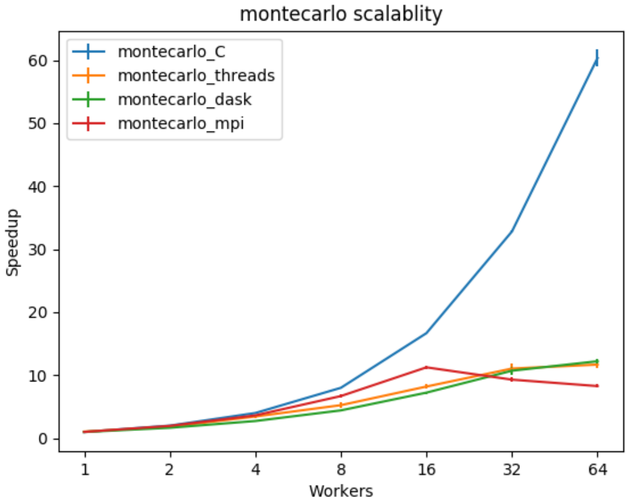
- Allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain.
- Not Turing-complete necessarily.

A dose of technical realism

A dose of technical realism



A dose of technical realism



Conclusions

1. KISS: our scientific colleges program in Python, so we do.
2. Python is a well-suited language for HPC environments, but programming requires a lot of effort.
3. HPC Python programmers must study the technology to handle the different abstraction levels at which problems arise.
4. This is an exciting research field, any volunteers?