



Python en Supercomputación

Charla Técnica

Guillem Borrell i Nogueras

ETSIA, Octubre 2007

Charla técnica

- ▶ Python Vs. Matlab.
- ▶ Más sobre los wrappers.
- ▶ Arrays en C, Fortran y Python.
- ▶ ctypes.
- ▶ F2Py.
- ▶ Python en paralelo.
- ▶ GIL.

Python vs. Matlab

¿Ayuda Matlab a manejar la complejidad?

- ▶ Cada función es un archivo
- ▶ No existe la modularidad
- ▶ Orientación a objetos inútil
- ▶ No existen los namespaces
- ▶ wrapper *eq mex*

Más sobre los wrappers

- ▶ Un wrapper es un adaptador entre los tipos de C, Fortran y los del lenguaje interpretado.
- ▶ En un lenguaje interpretado los tipos siempre son structs.
- ▶ En python los tipos tienen ~ 30 elementos.
- ▶ El intérprete tiene su propio modelo de stack.

Arrays

El objetivo es encajar

```
double array[ ];
```

en:

```
typedef struct PyArrayObject {  
    PyObject HEAD  
    char * data;  
    int nd;  
    intp * dimensions;  
    intp * strides;  
    PyObject * base;  
    PyArray_Descr * descr;  
    int flags;  
    PyObject * weakreflist;  
} PyArrayObject ; };
```

Posibles problemas

- ▶ Hacerlo manualmente requiere C medio
- ▶ Conocimiento del intérprete
- ▶ Problemas de alineación (strides)
- ▶ ¿Como Fortran o como C?

¿No era tan fácil?

Es fácil gracias a...

- ▶ ctypes
- ▶ f2py

Permite enlazar en tiempo de ejecución una librería al intérprete

Un wrapper inútil.

```
from ctypes import c_int, POINTER #1
import numpy as np
from numpy.ctypeslib import load_library,ndpointer #1

def dgesv(N,A,B):
    A = np.asfortranarray(A.astype(np.float64)) #2
    B = np.asfortranarray(B.astype(np.float64))

    cN=c_int(N)
    NRHS=c_int(1)
    LDA=c_int(N)
    IPIV=(c_int * N)()
    LDB=c_int(N)
    INFO=c_int(1)

    lapack=load_library('liblapack.so', '/usr/lib/')#3
    ...
```

```
lapack.dgesv_.argtypes=[POINTER(c_int),
    POINTER(c_int),
    ndpointer(dtype=np.float64,
    ndim=2,
    flags='FORTRAN'),
    POINTER(c_int), POINTER(c_int),
    ndpointer(dtype=np.float64,
    ndim=2,
    flags='FORTRAN'),
    POINTER(c_int),POINTER(c_int)]#4
```

```
lapack.dgesv_(cN,NRHS,A,LDA,IPIV,B,LDB,INFO)#5  
return B
```

```
print dgesv(2,np.array([[1,2],[1,4]]),np.array([[1,3]]))
```

¡No hay que programar en C!

- ▶ FORTRAN (trailing underscore)
- ▶ Conversión de arrays
- ▶ Llamadas por referencia
- ▶ Toda subrutina puede ser una librería, sólo hay que compilarla de otra manera.
- ▶ Velocidad de ejecución \sim Fortran
- ▶ Reciclaje

Es una aplicación que es capaz de entender la mayoría del código en Fortran y lo convierte automáticamente en un módulo de Python

Aún más fácil con f2py

```
!t.f90
subroutine withCallback(a, b, ipar, rpar, callback)
  real a,b, rpar(*)
  integer ipar(*)
  external callback
  print*, 'The parameters are', a,b, ipar(:3), rpar(:3)
  call callback(rpar, ipar)
end subroutine withCallback

subroutine theCallback(rpar, ipar)
  real rpar(*)
  integer ipar(*)
  print*, 'Here the callback is called', ipar(:3), rpar(:3)
end subroutine theCallback
```

Y funciona...

```
$ f2py -c -m callback t.f90 --fcompiler=gnu95
```

```
>>> from numpy import *
```

```
>>> import callback
```

```
>>> ipar=array([4,5,6])
```

```
>>> rpar=array([1.,2.,3.])
```

```
>>> callback.withcallback(8,9,ipar,rpar,
```

```
...     callback.thecallback)
```

```
The parameters are  8.000000          9.000000    4    5
```

```
        6  1.000000          2.000000          3.000000
```

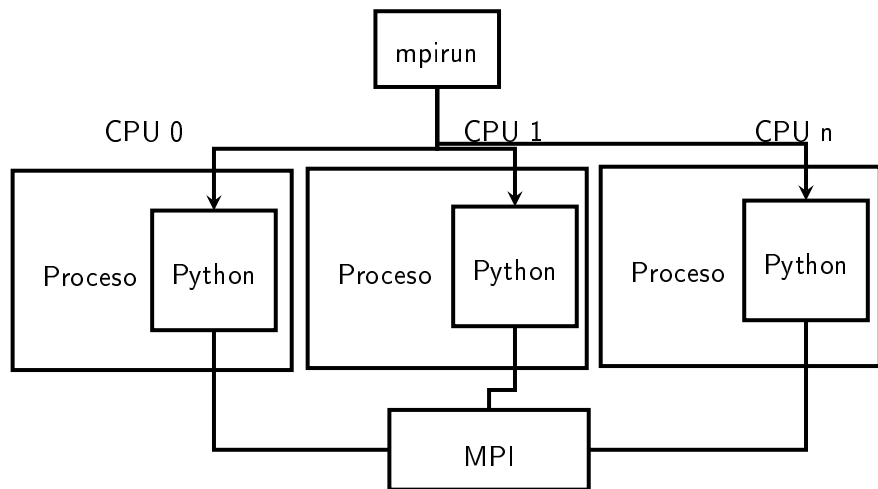
```
Here the callback is called          4    5    6
```

```
1.000000          2.000000          3.000000
```

```
>>>
```

Ahora sí parece más fácil.

Python en paralelo



Python en paralelo II

- ▶ Se lanza Python como proceso
- ▶ La comunicación entre los intérpretes puede hacerse mediante MPI
 - ▶ No hay wrappers para **blacs** pero pueden hacerse

Por ejemplo:

GIL

- ▶ Cpython no es thread safe
- ▶ No aprovecha los multiple core
- ▶ Programación concurrente (Threading)
 - ▶ No hay ganancia respecto a C
 - ▶ ¿Esperar a stackless o pypy?
- ▶ Lo más seguro sigue siendo usar procesos