Scientific Computing with Python and CUDA

Stefan Reiterer

High Performance Computing Seminar, January 17 2011

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

What is Python?

Python is

What is Python?

Python is

- a high level programming language

What is Python?

Python is

- a high level programming language
- interpreted,

What is Python?

Python is

- a high level programming language
- interpreted,
- object oriented,

What is Python?

Python is

- a high level programming language
- interpreted,
- object oriented,
- named after the "Monty Pythons".

What is Python?

Python is

- a high level programming language
- interpreted,
- object oriented,
- named after the "Monty Pythons".

It was invented by Guido van Rossum in the early 90s.

Why Python?

- Object Oriented programming

Why Python?

- Object Oriented programming $\rightarrow$ reusable code

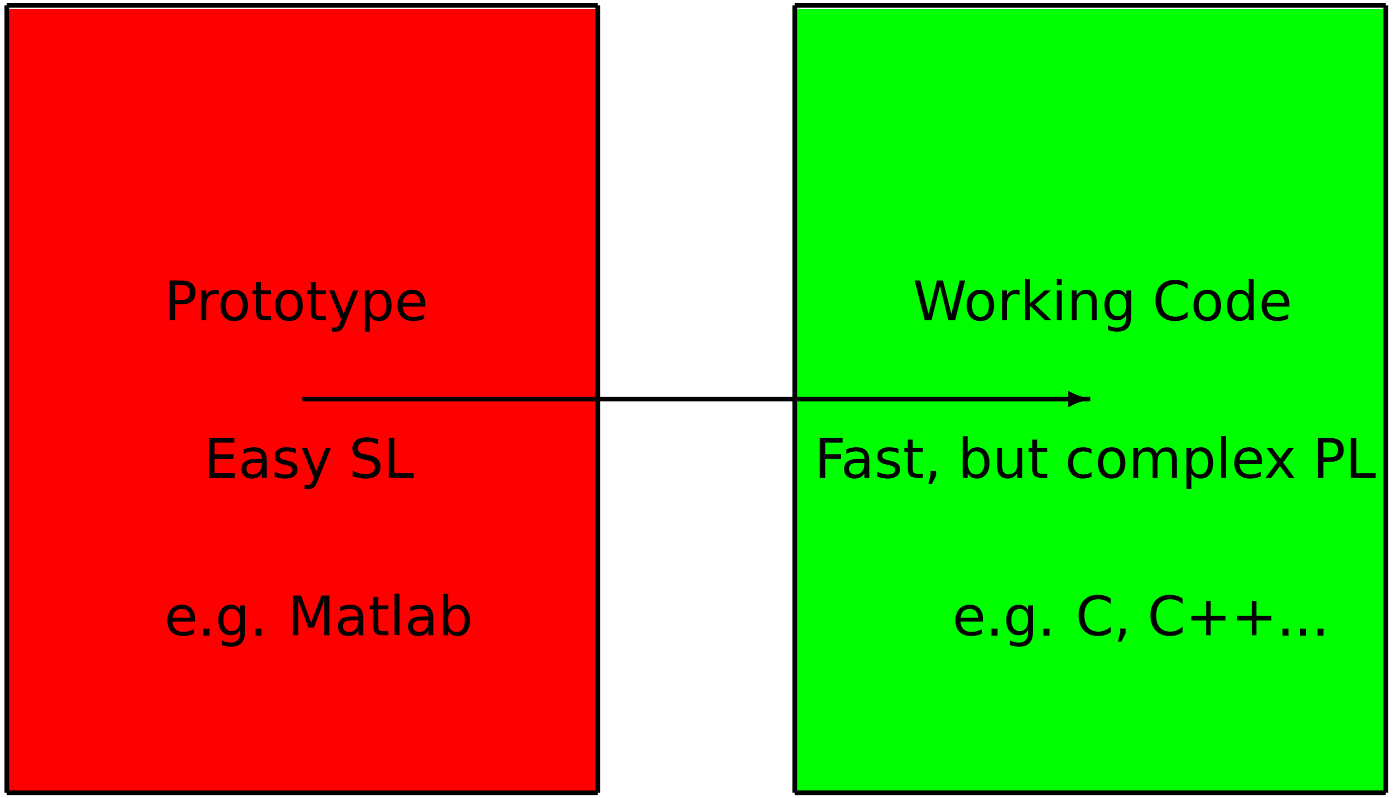
Why Python?

- Object Oriented programming $\rightarrow$ reusable code
- True strength of Python: Rapid prototyping

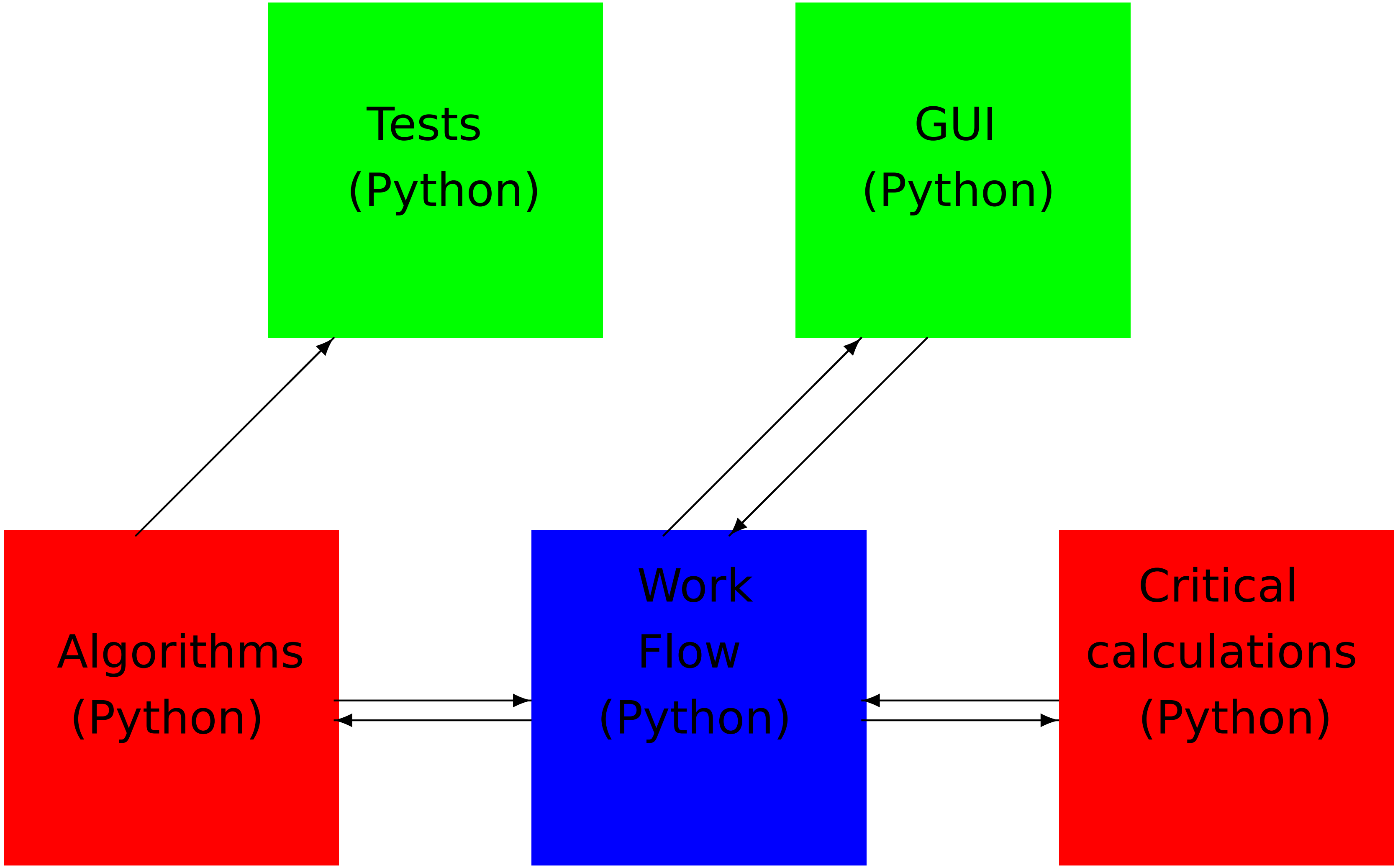
Why Python?

- Object Oriented programming $\rightarrow$ reusable code
- True strength of Python: Rapid prototyping $\rightarrow$ accelerate research process, saves time and nerves.

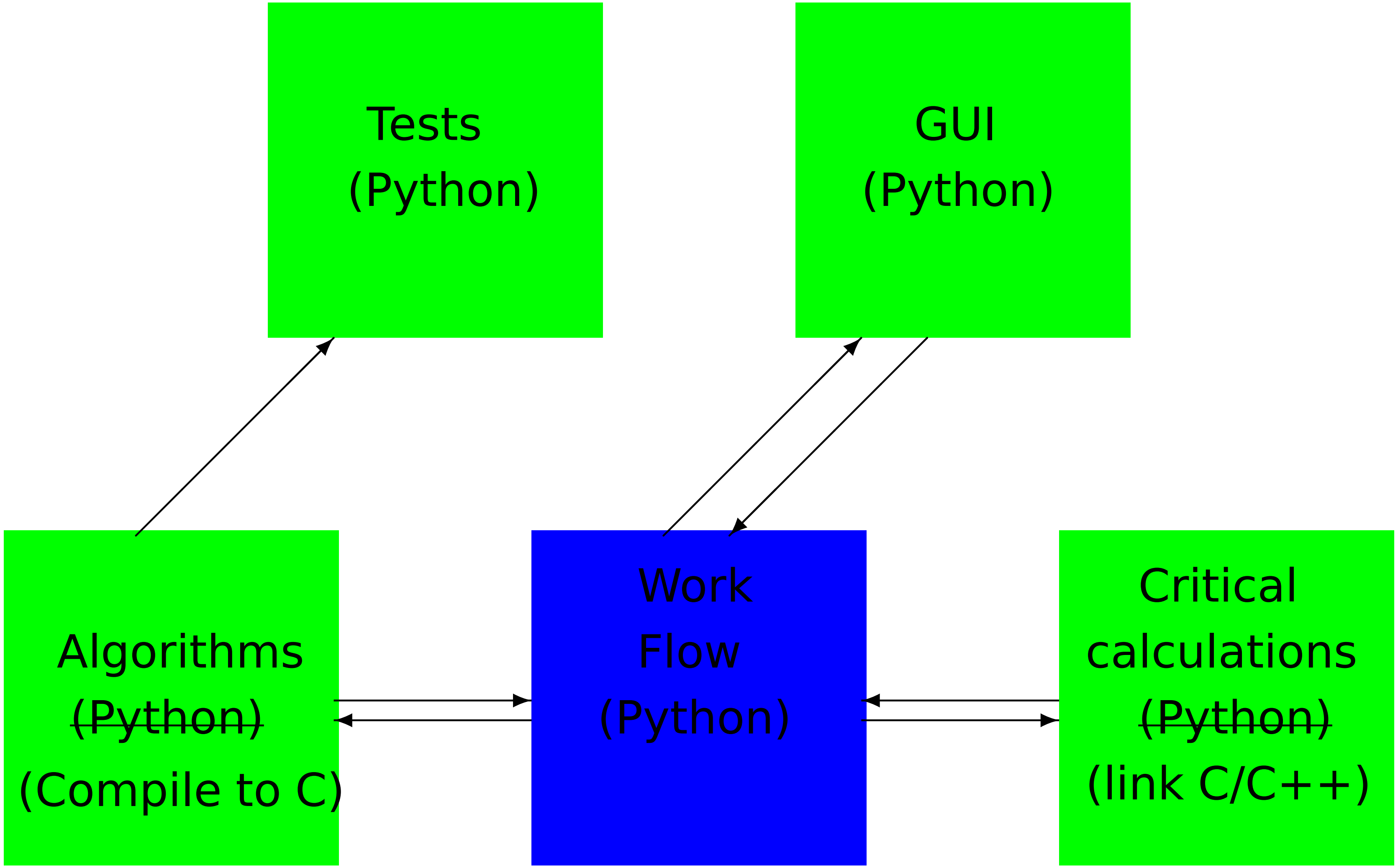
Classical prototyping process



Prototyping with Python



Prototyping with Python



Don't fear Python!

Don't fear Python!

- Intuitive Syntax
- No Allocation
- Garbage Collection
- Easy to Learn!

The Goodbye World Program

```
print("GoodbyeWorld!")
```

Definition of functions

```
def function_name(arg1,arg2,arg3 = default):
    result = arg1 + arg2*arg3 #do something
    return result
```

Definition of functions

```
def function_name(arg1,arg2,arg3 = default):
    result = arg1 + arg2*arg3 #do something
    return result
```

Python recognizes code blocks with intends!

Definition of functions

```
def function_name(arg1,arg2,arg3 = default):
    result = arg1 + arg2*arg3 #do something
    return  result
```

Python recognizes code blocks with intends!

Always remember: Don’t mix tabs with spaces!

If statments

```
if cond1 is True:  #normal if
    do this
elif cond2 is True: #else if
    do that
else:               #else
    do something_else
```

while loops

```
while condition is True:
    do something
```

while loops

```
while condition is True:
    do something
```

While loops know else too:

```
while condition is True:
    do something
else:
    do something_else
```

for loops

```
for x in range(n):
    do something with x
```

for loops

```
for x in range(n):
    do something with x
```

- Instead of **range** one can use any list!

for loops

```
for x in range(n):
    do something with x
```

- Instead of **range** one can use any list!
- For better performance use **xrange**.

for loops

```
for x in range(n):
    do something with x
```

- Instead of **range** one can use any list!
- For better performance use **xrange**.
- for loops also have an else statement.

An example...

```
for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print n, 'equals', x, '*', n/x
            break
    else:
        # loop fell through without finding a factor
        print n, 'is a prime number'
```

Output

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Classes

In Python everything is a class!

Classes

In Python everything is a class!

No elementary data types!

Definition of a class

Classes are defined in the same way as functions

```
class my_class:
    statement1
    statement2
    ...
```

Definition of a class

Classes are defined in the same way as functions

```
class my_class:
    statement1
    statement2
     ...
```

- **__init__** serves as constructor

Definition of a class

Classes are defined in the same way as functions

```
class my_class:
    statement1
    statement2
    ...
```

- **__init__** serves as constructor
- **__del__** serves as destructor

Definition of a class

Classes are defined in the same way as functions

```
class my_class:
    statement1
    statement2
    ...
```

- **__init__** serves as constructor
- **__del__** serves as destructor
- Class methods take the object itself as first argument (as convention one writes **self**)

Derivative of classes

Classes are derived simply with

```
class my_class(inheritence):
    statement1
    statement2
    ...
```

Derivative of classes

Classes are derived simply with

```
class my_class(inheritence):
    statement1
    statement2
    ...
```

- They hold the same methods and variables as their parents

Derivative of classes

Classes are derived simply with

```
class my_class(inheritence):
    statement1
    statement2
    ...
```

- They hold the same methods and variables as their parents
- To overload methods simply add the same method agein.

Operator overloading

To overload operators like +,-,*,/ etc simply add the methods **__add__**, **__sub__**, **__mul__**, **__div__** ...

Another example:

```
class band_matrix:
  def __init__(self,ab):
      self.shape = (ab.shape[1],ab.shape[1])
      self.data = ab
      self.band_width = ab.shape[0]
      self.dtype = ab.dtype

  def matvec(self,u):
      if self.shape[0] != u.shape[0]:
          raise ValueError("Dimension␣Missmatch!")
      return band_matvec(self.data,u)
  def __add__(self,other):
      return add_band_mat(self,other)
```

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

NumPy and SciPy

- NumPy is for Numerical Python
- SciPy is for Scientific Python

NumPy and SciPy

- NumPy is for Numerical Python
- SciPy is for Scientific Python
- Allow Numerical Linear Algebra like in Matlab

NumPy and SciPy

- NumPy is for Numerical Python
- SciPy is for Scientific Python
- Allow Numerical Linear Algebra like in Matlab
- Speed: NumPy + SciPy $\approx$ Matlab

Some examples...

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

Band Matrix vector Multiplication

```python
# -*- coding: utf-8 -*-
def band_matvec(A,u):

    result = zeros(u.shape[0],dtype=u.dtype)



    for i in xrange(A.shape[1]):
        result[i] = A[0,i]*u[i]

    for j in xrange(1,A.shape[0]):
        for i in xrange(A.shape[1]-j):
            result[i] += A[j,i]*u[i+j]
            result[i+j] += A[j,i]*u[i]

    return result
```

Write the C-Code inline in Python with Blitz

```
# -*- coding: utf-8 -*-
from numpy import array, zeros
from scipy.weave import converters
from scipy import weave

def band_matvec_inline(A,u):

    result = zeros(u.shape[0],dtype=u.dtype)

    N = A.shape[1]
    B = A.shape[0]
```

Write the C-Code inline in Python with Blitz

```
code = """
for(int i=0; i < N;i++) {
  result(i) = A(0,i)*u(i);
}
for(int j=1;j < B;j++) {
    for(int i=0; i < (N-j);i++) {
      if((i+j < N)) {
        result(i) += A(j,i)*u(j+i);
        result(i+j) += A(j,i)*u(i);
      }
    }
}
"""
```

Write the C-Code inline in Python with Blitz

```
weave.inline(code,['u', 'A', 'result', 'N', 'B'],
             type_converters=converters.blitz)
return result
```

Write the C-Code inline in Python with Blitz

```
weave.inline(code,['u', 'A', 'result', 'N', 'B'],
                    type_converters=converters.blitz)
return result
```

≈ 290x speedup to python

Cython

Python produces a lot of overhead with data types

Cython

Python produces a lot of overhead with data types

Solution: Declare which data types to use and compile it with Cython!

Python again

```python
# -*- coding: utf-8 -*-
def band_matvec(A,u):

    result = zeros(u.shape[0],dtype=u.dtype)



    for i in xrange(A.shape[1]):
        result[i] = A[0,i]*u[i]

    for j in xrange(1,A.shape[0]):
        for i in xrange(A.shape[1]-j):
            result[i] += A[j,i]*u[i+j]
            result[i+j] += A[j,i]*u[i]

    return result
```

Get more with Cython

```
import numpy, scipy
cimport numpy as cnumpy

ctypedef cnumpy.float64_t reals
def matvec_help(cnumpy.ndarray[reals,ndim=2] A,
                cnumpy.ndarray[reals,ndim=1] u):
    cdef Py_ssize_t i,j
    cdef cnumpy.ndarray[reals,ndim=1] result = \
            numpy.zeros(A.shape[1],dtype=A.dtype)
    for i in xrange(A.shape[1]):
        result[i] = A[0,i]*u[i]
    for j in xrange(1,A.shape[0]):
        for i in xrange(A.shape[1]-j):
            result[i] = result[i] + A[j,i]*u[i+j]
            result[i+j] = result[i+j]+A[j,i]*u[i]
```

...and even more

```
import numpy, scipy
cimport numpy as cnumpy, cython
@cython.boundscheck(False)
ctypedef cnumpy.float64_t reals
def matvec_help(cnumpy.ndarray[reals,ndim=2] A,
                cnumpy.ndarray[reals,ndim=1] u):
    cdef Py_ssize_t i,j
    cdef cnumpy.ndarray[reals,ndim=1] result = \
          numpy.zeros(A.shape[1],dtype=A.dtype)
    for i in xrange(A.shape[1]):
        result[i] = A[0,i]*u[i]
    for j in xrange(1,A.shape[0]):
        for i in xrange(A.shape[1]-j):
            result[i] = result[i] + A[j,i]*u[i+j]
            result[i+j] = result[i+j]+A[j,i]*u[i]
```

Speedup to python

Speedup to python

- Cython $\approx 220\times$ speedup

Speedup to python

- Cython $\approx 220\times$ speedup
- ...with boundscheck: $\approx 440\times$ speedup

Overview

Language	time	speedup
Python	2.96 s	$1\times$
Matlab	90 ms	$30\times$
Pure C	50 ms	$60\times$
Cython	12 ms	$220\times$
Inline C++	10 ms	$290\times$
Cython w. boundscheck	6 ms	$440\times$

Overview

Language	time	speedup
Python	2.96 s	1$\times$
Matlab	90 ms	30$\times$
Pure C	50 ms	60$\times$
Cython	12 ms	220$\times$
Inline C++	10 ms	290$\times$
Cython w. boundscheck	6 ms	440$\times$

With linking libraries like Lapack even more is possible!

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

What is PyCUDA

- PyCUDA is a Wrapper in Python to CUDA
- Inline CUDA
- Garbage Collection
- A NumPy like vector class.
- Developed by Andreas Klöckner

Step 1: Write your Code in CUDA

```
source = """
__global__ void doublify(float *a)
  {
    int idx = threadIdx.x + threadIdx.y*4;
    a[idx] *= 2;
  }
"""
```

Step 2: Get it into Python

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule

mod = SourceModule(source)
func = mod.get_function("doublify")
```

Step3: Call it

```
import numpy
#create vector
x = numpy.random.randn(4,4)
x = x.astype(numpy.float32)
#copy it on card
x_gpu = cuda.mem_alloc(x.nbytes)
cuda.memcpy_htod(x_gpu, x)
#call function
func(x_gpu, block=(4,4,1))
#get data back
x_doubled = numpy.empty_like(x)
cuda.memcpy_dtoh(x_doubled, x_gpu)
```

Benefits

- You can compile CUDA code on runtime

Benefits

- You can compile CUDA code on runtime
- You can call it in Python

Benefits

- You can compile CUDA code on runtime
- You can call it in Python

I was able to implement CG algorithm with the different variants without rewriting code!

Overview (Band-matrix vector multiplication)

Language	time	speedup
Python	2.87 s	$1\times$
Cython	9.87 ms	$270\times$
Cython w. boundscheck	5.56 ms	$500\times$
PyCUDA (on GTX 280)	585 μ s	$5000\times$

Overview (Band-matrix vector multiplication)

Language	time	speedup
Python	2.87 s	$1\times$
Cython	9.87 ms	$270\times$
Cython w. boundscheck	5.56 ms	$500\times$
PyCUDA (on GTX 280)	585 μ s	$5000\times$

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

Now something completly different... mpi4py

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
print("hello world")
print("my rank is: %d"%comm.rank)
```

Now something completly different... mpi4py

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
print("hello world")
print("my rank is: %d"%comm.rank)
```

Simply call Python with MPI:

mpirun -n <# processes> python filename.py

PyCUDA+mpi4py

```
from mpi4py import MPI
import pycuda.driver as cuda
from pycuda import gpuarray
from numpy import float32, array
from numpy.random import randn as rand
```

PyCUDA+mpi4py

```
comm = MPI.COMM_WORLD
rank = comm.rank
root = 0
nr_gpus = 4
sendbuf = []
N = 2**20*nr_gpus
K = 1000
if rank == 0:
  x = rand(N).astype(float32)*10**16
  print "x:", x
  cuda.init() #init pycudadriver
  sendbuf = x.reshape(nr_gpus,N/nr_gpus)
```

PyCUDA+mpi4py

```
if rank > nr_gpus-1:
  raise ValueError("To few gpus!")
current_dev = cuda.Device(rank) #device we are working on
ctx = current_dev.make_context() #make a working context
ctx.push() #let context make the lead
#recieve data and port it to gpu:
x_gpu_part = gpuarray.to_gpu(comm.scatter(sendbuf,root))
```

PyCUDA+mpi4py

```
#do something...
for k in xrange(K):
  x_gpu_part = 0.9*x_gpu_part
```

PyCUDA+mpi4py

```
#get data back:
x_part = (x_gpu_part).get()
ctx.pop() #deactivate again
ctx.detach() #delete it
```

PyCUDA+mpi4py

```
recvbuf=comm.gather(x_part,root) #recieve data
if rank == 0:
  x_altered= array(recvbuf).reshape(N)
  print "altered␣x:", x_altered
```

Overview

N	K	NumPy	MPI+Numpy	MPI+PyCUDA
2^{17}	500	125 ms	4.6 ms	3 s
2^{18}	1000	636 ms	212 ms	3.7 s
2^{19}	2000	2.68 s	781 ms	3.84 s
2^{20}	4000	10.9 s	6.11 s	7.6 s
2^{21}	8000	39.5 s	24.2 s	11.6 s
2^{22}	16000	30 min	93.0 s	19.3 s
2^{23}	32000	?	382 s	24.1 s
2^{24}	64000	?	25 min	48.0 s

Inhalt

1. A short Introduction to Python
2. Scientific Computing tools in Python
3. Easy ways to make Python faster
4. Python + CUDA = PyCUDA
5. Python + MPI = mpi4py
6. ...and the Rest

Where can I get Python?

Everything is Opensource

Links:

- http://www.python.org/
- http://www.scipy.org/
- http://www.sagemath.org/
- http://femhub.org/
- http://cython.org/

Where can I get Python?

Everything is Opensource

Links:

- http://www.python.org/
- http://www.scipy.org/
- http://www.sagemath.org/
- http://femhub.org/
- http://cython.org/

If you need the latest mpi4py and and PyCUDA packages for Sage/Femhub ask me!

Other interesting things for Python

- f2py $\rightarrow$ Fortran Code inline
- petsc4py $\rightarrow$ PETSC interface for Python
- PyOpenCL $\rightarrow$ like PyCUDA but for OpenCl
- matplotlib, Mayavi $\rightarrow$ powerful plotting libraries

... and many many more

Questions?