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 a high-level, interpreted, object-oriented programming language. It was created in the 1990s by Guido van Rossum, named after Monty Python’s Flying Circus. It is known for its simple and clear syntax, making it accessible to beginners, while also powerful enough for complex tasks. Python is widely used in scientific computing, data analysis, machine learning, and web development.
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 → reusable code
Why Python?

- Object Oriented programming → **reusable code**
- True strength of Python: **Rapid prototyping**
Why Python?

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

Prototype

Easy SL

e.g. Matlab

Working Code

Fast, but complex PL

e.g. C, C++...
Prototyping with Python

- Tests (Python)
- GUI (Python)
- Algorithms (Python)
- Work Flow (Python)
- Critical calculations (Python)
Prototyping with Python

- Tests (Python)
- GUI (Python)
- Algorithms (Python) (Compile to C)
- Work Flow (Python)
- Critical calculations (Python) (link C/C++)
Don’t fear Python!
Don’t fear Python!

- Intuitive Syntax
- No Allocation
- Garbage Collection
- Easy to Learn!
The Goodbye World Program

```python
print("GoodbyeWorld!")
```
Definition of functions

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

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

Python recognizes code blocks with **intends**!
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 statements

```python
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 know else too:
while condition is True:
    do something
    else:
        do something_else
while loops

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

While loops know else too:

```python
while condition is True:
    do something
else:
    do something_else
```
for loops

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

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

- Instead of `range` one can use any list!
for loops

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

- Instead of `range` one can use any list!
- For better performance use `xrange`.
for loops

```python
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.
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
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

```python
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

```python
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`)

Stefan Reiterer
Scientific Computing with Python and CUDA
Derivative of classes

Classes are derived simply with

class my_class(inheritance):
    statement1
    statement2
    ...

Derivative of classes

Classes are derived simply with

class my_class(inheritance):
    statement1
    statement2
    ...

- They hold the same methods and variables as their parents
Networks, connections, and their properties.
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
```

Stefan Reiterer ()
Scientific Computing with Python and CUDA
HPC Seminar 26 / 55
Write the C-Code inline in Python with Blitz

```python
# 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]
```

Stefan Reiterer
Scientific Computing with Python and CUDA
HPC Seminar 27 / 55
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

```python
weave.inline(code, ['u', 'A', 'result', 'N', 'B'], type_converters=converters.blitz)
return result
```
Easy ways to make Python faster

Write the C-Code inline in Python with Blitz

```python
def 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!
Easy ways to make Python faster

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
```

Stefan Reiterer
Scientific Computing with Python and CUDA
HPC Seminar 31 / 55
Get more with Cython

```python
import numpy, scipy
cimport numpy as cnumpy
ctypedef cnumpy.float64_t reals
    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]
    return result
```

Stefan Reiterer
Scientific Computing with Python and CUDA
HPC Seminar
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]
Easy ways to make Python faster

Speedup to python

≈ 220 × speedup

...with boundscheck:

≈ 440 × speedup
Easy ways to make Python faster

Speedup to python

- **Cython** $\approx 220 \times$ speedup
Easy ways to make Python faster

Speedup to python

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

<table>
<thead>
<tr>
<th>Language</th>
<th>time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>2.96 s</td>
<td>1×</td>
</tr>
<tr>
<td>Matlab</td>
<td>90 ms</td>
<td>30×</td>
</tr>
<tr>
<td>Pure C</td>
<td>50 ms</td>
<td>60×</td>
</tr>
<tr>
<td>Cython</td>
<td>12 ms</td>
<td>220×</td>
</tr>
<tr>
<td>Inline C++</td>
<td>10 ms</td>
<td>290×</td>
</tr>
<tr>
<td>Cython w. boundscheck</td>
<td>6 ms</td>
<td>440×</td>
</tr>
</tbody>
</table>

With linking libraries like Lapack even more is possible!
Easy ways to make Python faster

Overview

<table>
<thead>
<tr>
<th>Language</th>
<th>time (s)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>2.96</td>
<td>1×</td>
</tr>
<tr>
<td>Matlab</td>
<td>0.09</td>
<td>30×</td>
</tr>
<tr>
<td>Pure C</td>
<td>0.05</td>
<td>60×</td>
</tr>
<tr>
<td>Cython</td>
<td>0.012</td>
<td>220×</td>
</tr>
<tr>
<td>Inline C++</td>
<td>0.01</td>
<td>290×</td>
</tr>
<tr>
<td>Cython w. boundscheck</td>
<td>0.006</td>
<td>440×</td>
</tr>
</tbody>
</table>

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;
}
"
```

Stefan Reiterer
Scientific Computing with Python and CUDA
Step 2: Get it into Python

```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

```python
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)

<table>
<thead>
<tr>
<th>Language</th>
<th>time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>2.87 s</td>
<td>1×</td>
</tr>
<tr>
<td>Cython</td>
<td>9.87 ms</td>
<td>270×</td>
</tr>
<tr>
<td>Cython w. boundscheck</td>
<td>5.56 ms</td>
<td>500×</td>
</tr>
<tr>
<td>PyCUDA (on GTX 280)</td>
<td>585 µs</td>
<td>5000×</td>
</tr>
</tbody>
</table>
Overview (Band-matrix vector multiplication)

<table>
<thead>
<tr>
<th>Language</th>
<th>time</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>2.87 s</td>
<td>1×</td>
</tr>
<tr>
<td>Cython</td>
<td>9.87 ms</td>
<td>270×</td>
</tr>
<tr>
<td>Cython w. boundscheck</td>
<td>5.56 ms</td>
<td>500×</td>
</tr>
<tr>
<td>PyCUDA (on GTX 280)</td>
<td>585 μs</td>
<td>5000×</td>
</tr>
</tbody>
</table>
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 completely different... mpi4py

```python
from mpi4py import MPI
comm = MPI.COMM_WORLD
print("hello_world")
print("my_rank_is:%d"%comm.rank)
```
from mpi4py import MPI
comm = MPI.COMM_WORLD
print("hello_world")
print("my_rank_is:%d\n\ncomm.rank")

Simply call Python with MPI:
mpirun -n <# processes> python filename.py
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
```python
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)
```

Stefan Reiterer ()
Scientific Computing with Python and CUDA
HPC Seminar 46 / 55
if rank > nr_gpus - 1:
    raise ValueError("Too 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
# receive data and port it to gpu:
x_gpu_part = gpuarray.to_gpu(comm.scatter(sendbuf, root))
PyCUDA+mpi4py

#do something . . .

```python
for k in xrange(K):
    x_gpu_part = 0.9*x_gpu_part
```

Stefan Reiterer

Scientific Computing with Python and CUDA

HPC Seminar 48 / 55
#get data back:
x_part = (x_gpu_part).get()
ctx.pop() #deactivate again
ctx.detach() #delete it
recvbuf=comm.gather(x_part, root)  # receive data
if rank == 0:
    x_altered = array(recvbuf).reshape(N)
print "altered x: ", x_altered
## Overview

<table>
<thead>
<tr>
<th>N</th>
<th>K</th>
<th>NumPy</th>
<th>MPI+Numpy</th>
<th>MPI+PyCUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{17}$</td>
<td>500</td>
<td>125 ms</td>
<td>4.6 ms</td>
<td>3 s</td>
</tr>
<tr>
<td>$2^{18}$</td>
<td>1000</td>
<td>636 ms</td>
<td>212 ms</td>
<td>3.7 s</td>
</tr>
<tr>
<td>$2^{19}$</td>
<td>2000</td>
<td>2.68 s</td>
<td>781 ms</td>
<td>3.84 s</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>4000</td>
<td>10.9 s</td>
<td>6.11 s</td>
<td>7.6 s</td>
</tr>
<tr>
<td>$2^{21}$</td>
<td>8000</td>
<td>39.5 s</td>
<td>24.2 s</td>
<td>11.6 s</td>
</tr>
<tr>
<td>$2^{22}$</td>
<td>16000</td>
<td>30 min</td>
<td>93.0 s</td>
<td>19.3 s</td>
</tr>
<tr>
<td>$2^{23}$</td>
<td>32000</td>
<td>?</td>
<td>382 s</td>
<td>24.1 s</td>
</tr>
<tr>
<td>$2^{24}$</td>
<td>64000</td>
<td>?</td>
<td>25 min</td>
<td>48.0 s</td>
</tr>
</tbody>
</table>
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.python.org/)
- [http://www.sagemath.org/](http://www.sagemath.org/)
- [http://femhub.org/](http://femhub.org/)
- [http://cython.org/](http://cython.org/)

If you need the latest mpi4py and PyCUDA packages for Sage/Femhub ask me!
Where can I get Python?

Everything is **Opensource**

Links:

- [http://www.python.org/](http://www.python.org/)
- [http://www.sagemath.org/](http://www.sagemath.org/)
- [http://femhub.org/](http://femhub.org/)
- [http://cython.org/](http://cython.org/)

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

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

... and many many more
Questions?