Python for Grid- Cloud-, and High-Performance-Computing

ISGC 2012, Taipei, Taiwan

Andreas Schreiber
German Aerospace Center (DLR)

Abstract

- Python is an accepted high-level scripting language with a growing community in academia and industry. It is used in many scientific applications in many different scientific fields and in more and more industries. In all fields, the use of Python for high-performance and parallel computing is increasing. Several organizations and companies are providing tools or support for Python development. This includes libraries for scientific computing, parallel computing, and MPI. Python is also used on many core architectures and GPUs, for which specific Python interpreters are being developed. The talk describes, why Python is used and specific advantages and current drawbacks of Python for scientific applications. Predictions of future uses of Python are presented. Hints and best practices to the get major improvements in the development of distributed and HPC applications

Outline

- Python
- Examples:
    - Grid- and Cloud-Computing
    - High-Performance-Computing
- Best practices

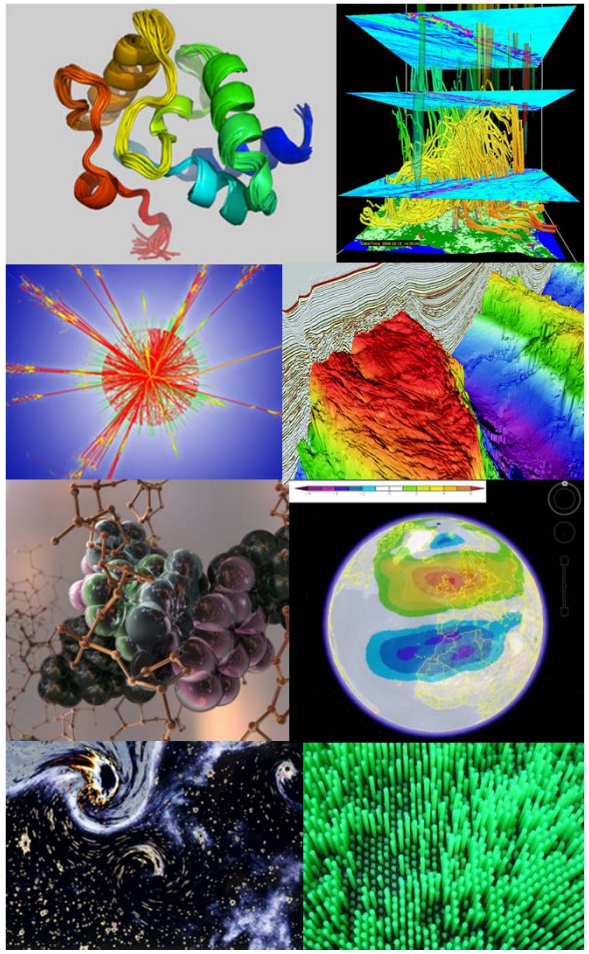
python™

DLR

Grid/Cloud/HPC Applications

Python is used in…

- Computational Fluid Dynamics (CFD)
- Plasma simulation
- Bio molecular simulation
- Artificial intelligence
- Natural language processing
- Data mining
- Scientific visualization
- Robotics
- Computer games
- System administration
- Web sites
- Education
- …



“If it’s good enough for Google and NASA, it’s good enough for me, baby.

What is Python?

```
def factorial(x):
    if x > 1:
        return x * factorial(x - 1)
    else:
        return 1
```

What is Python?

- General dynamic programming language
- Natural expression of procedural code (almost “pseudo code”)
- Allows procedural, object oriented, and functional programming
- Full modularity
- Extendable with modules written in C, C++, Fortran, Java, C#, …
- Embeddable within applications as a scripting interface
- Free with an Open Source license
- It has “Batteries included”
    - *Standard library, covers everything from asynchronous processing to zip files*

python™

DLR

“There seems to be two sorts of people who love Python™: those who hate brackets, and scientists.

Advantages of Python

- Very easy to learn and easy to use
  - *Results in steep learning curve*

- Allows rapid development
  - *Results in short development time*

- **Inherent great maintainability**
  - *Results in protection of investment*

Python for Scientists and Engineers
Reasons for Python in Research and Industry

- Observations
  - Scientists and engineers don't want to write software but just solve their problems
  - If they have to write code, it must be as easy as possible

- Python allows to focus on the problem
  - Similar applies to DSLs such as MATLAB, R, …
  - But with cleaner syntax

“Python has the cleanest, most-scientist- or engineer friendly syntax and semantics.

Paul F. Dubois

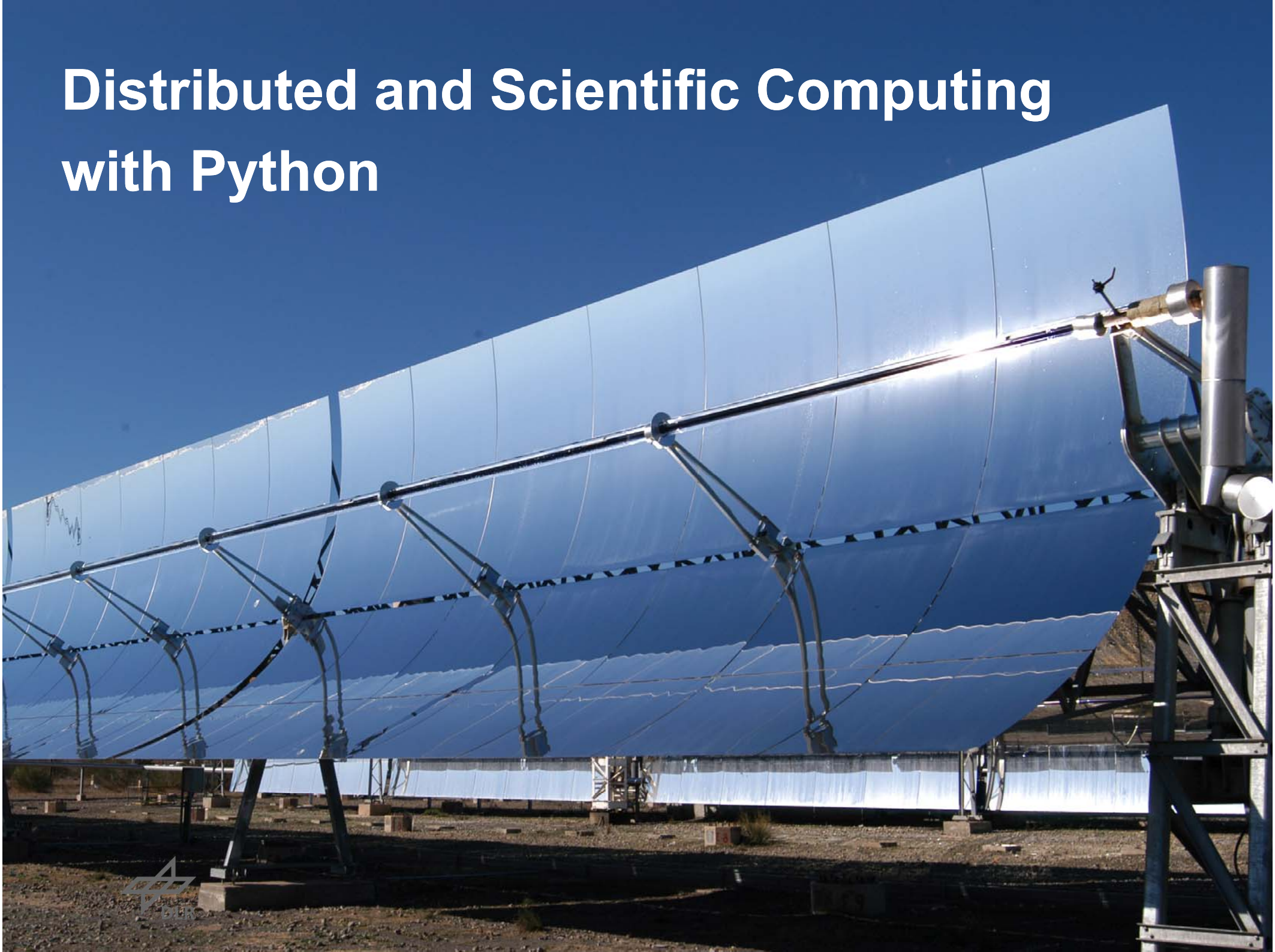
Drawbacks of Python

- **It's slow!**
  - *Need to write computational code in other languages (C etc.)*
  - *But getting improved.*

- No threading
  - *Due to GIL (Global interpreter lock)*

- Lack of static typing
  - *Can result in runtime errors*

Distributed and Scientific Computing with Python



Python for Grid Computing

- Many outdated APIs

- **Simple API for Grid Applications (SAGA)**
  - Standard by OGF
  - High-level interfaces and runtime components
  - SAGA C++ Reference Implementation
    - Support for Globus, UNICORE, Condor, gLite
    - Has a Python-API

SAGA
A Simple API for Grid Applications

- **DIRAC (Distributed Infrastructure with Remote Agent Control)**
  - Written in Python with high-level API

DIRAC

DLR

SAGA

```
import saga

# describe job
jd = saga.job.description()
jd.executable="/home/user/nihao-mpi”

# resource manager. Here: Globus GRAM
js =
saga.job.service("gram://my.globus.hos
t/jobmanager-pbs")

# run job
job = js.create_job(jd)
job.run()
```

Python for Cloud Computing

- **Cloud:**
  - (Compute) resources are owned and managed by a third party

- **Amazon Web Service (AWS)**
  - Amazon's Cloud infrastructure
  - Python API: **boto**

- **Google App Engine (GAE)**
  - PaaS Cloud platform
  - For hosting Web applications, includes many APIs

- **OpenStack**
  - Open Source Cloud architecture

boto

Python Interface to Amazon Web Services

- Website: https://github.com/boto/boto

- Convenient access to almost all Amazon Web Services
  - For example: *Simple Storage Service (S3), Elastic Compute Cloud (EC2), SimpleDB, CloudFront, Elastic Load Balancer (ELB), Virtual Private Cloud (VPC), Elastic Map Reduce (EMR), …*

boto

Amazon Elastic Compute Cloud (EC2)

Amazon Simple Storage Service (S3)

```
from boto.ec2.connection import EC2Connection

conn = EC2Connection(AWS_ACCESS_KEY_ID,
                     AWS_SECRET_ACCESS_KEY)

# Launching instance
conn.run_instances("<ami-image-id>")


from boto.s3.connection import S3Connection

conn = S3Connection(AWS_ACCESS_KEY_ID,
                    AWS_SECRET_ACCESS_KEY)
# Launching instance
conn.create_bucket("mybucket")
```

DLR

OpenStack

- Website: http://www.openstack.org/

- IaaS cloud architecture
- Initiated by Rackspace and NASA
- Written in Python
- Components
  - Compute
  - Object Store
  - Image Service

Scientific Tools and Libraries
General Tools

Very general scientific computing
- NumPy
- SciPy

Visualization
- Matplotlib
- VisIt
- MayaVi
- Chaco
- VTK

High Performance Computing

Parallel Computing
- PETSc
- PyMPI
- Pypar
- mpi4py

GPGPU Computing
- PyCUDA
- PyOpenCL
- Copperhead

Scientific Tools and Libraries
Domain Specific Tools

AI
- pyem
- ffnet
- pymorph
- Monte
- hcluster

Biology
- Brian
- SloppyCell
- NIPY
- PySAT

Electromagnetics
- PyFemax

Astronomy
- AstroLib
- PySolar

Dynamic Systems
- Simpy
- PyDSTool

Finite Elements
- SfePy

Molecular & Atomic Modeling
- PyMOL
- Biskit
- GPAW

Geo sciences
- GIS Python
- PyClimate
- ClimPy
- CDAT

Scientific Tools and Libraries
Special Topics

Wrapping other languages

- weave (C/C++)
- f2py (Fortran)
- Cython
- Ctypes (C)
- SWIG (C/C++)
- RPy / RSPython (R)
- MatPy (Matlab)
- Jython (Java)
- IronPython (.NET)

NumPy

- Website: http://numpy.scipy.org/

- Offers capabilities similar to MATLAB within Python

- N-dimensional homogeneous arrays (ndarray)
- Universal functions (ufunc)
    - basic math, linear algebra, FFT, PRNGs
- Simple data file I/O
    - text, raw binary, native binary
- Tools for integrating with C/C++/Fortran
- Heavy lifting done by optimized C/Fortran libraries
    - ATLAS or MKL, UMFPACK, FFTW, etc...

SciPy
Scientific Tools for Python

- Website: http://www.scipy.org
- Large library of scientific algorithms
- Extends NumPy with many tools for science and engineering
- Computationally intensive routines implemented in C and Fortran

Parallel Programming

Threading

- Useful for certain concurrency issues, not usable for parallel computing due to Global Interpreter Lock (GIL)

subprocess

- Relatively low level control for spawning and managing processes
- multiprocessing - multiple Python instances (processes)
- basic, clean multiple process parallelism

MPI

- mpi4py exposes your full local MPI API within Python
- As scalable as your local MPI

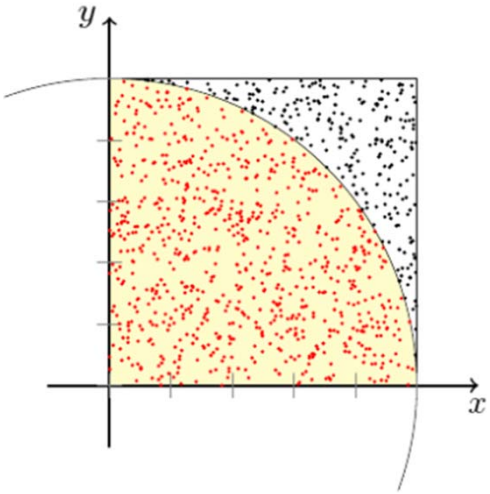
GPU (OpenCL & CUDA)

- PyOpenCL and PyCUDA provide low and high level abstraction for highly parallel computations on GPUs

mpi4py

- Website: http://mpi4py.scipy.org/
- Wraps native MPI implementations
  - Prefers MPI2, but can work with MPI1
- Works best with NumPy data types, but can pass around any serializable object
- Provides all MPI2 features
- Well maintained
- Distributed with Enthought Python Distribution (EPD)
- Requires NumPy
- Portable and scalable

Example: Calculating π with mpi4py




```
from mpi4py import MPI
import numpy as np
import random

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
mpisize = comm.Get_size()
nsamples = int(12e6/mpisize)

inside = 0
random.seed(rank)
for i in range(nsamples):
    x = random.random()
    y = random.random()
    if (x*x)+(y*y)<1:
      inside += 1

mypi = (4.0 * inside)/nsamples
pi = comm.reduce(mypi, op=MPI.SUM, root=0)

if rank==0:
    print (1.0 / mpisize)*pi
```

DLR

GPGPU

General-purpose computing on graphics proc. units

Architecture

- Many cores per node
- Good for stream processing (independent vertices, many of them in parallel)

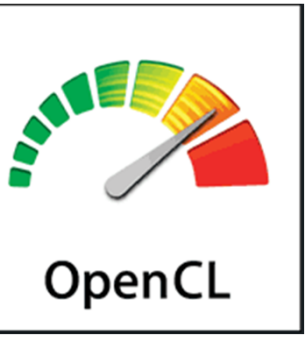
CUDA

- NVIDIA's platform for C programming on GPGPUs
- Python binding: PyCUDA

OpenCL

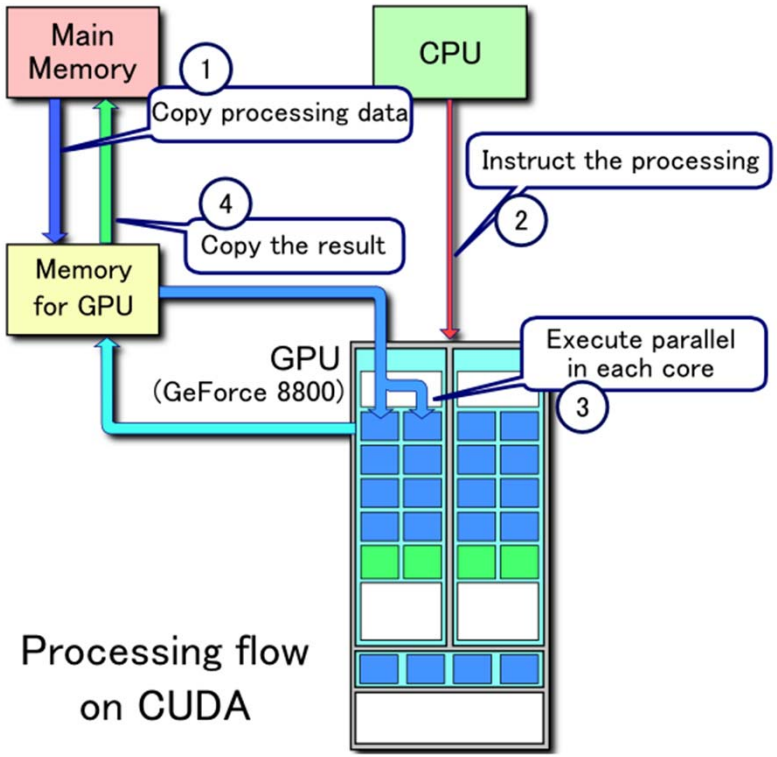
- Framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors
- Open standard
- Python binding: PyOpenCL

Copperhead



PyCUDA

- Website: http://mathema.tician.de/software/pycuda
- PyCUDA lets you access NVIDIA‘s CUDA parallel computation API from Python
- Integration with NumPy
- All CUDA errors are automatically translated into Python exceptions



PyOpenCL

- Website: http://mathema.tician.de/software/pyopencl
- Convenient access to full OpenCL API from Python
- OpenCL errors translated into Python exceptions
- Object cleanup tied to object lifetime (follows Resource Acquisition Is Initialization)
- NumPy ndarrays interact easily with OpenCL buffers

π with PyOpenCL

```
import pyopencl as cl
import pyopencl.clrandom
import numpy as np

nsamples = int(12e6)

# set up context and queue
ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)

# create array of random values in OpenCL
xy =
pyopencl.clrandom.rand(ctx,queue,(nsamples,2),np.float32)

# square values in OpenCL
xy = xy**2

# 'get' method on xy is used to get array from OpenCL
into ndarray
print 4.0*np.sum(np.sum(xy.get(),1)<1)/nsamples
```

Copperhead

Data-Parallelism Embedded in Python

- Subset of Python for GPUs

- *Slides by Michael Garland (NVIDIA Research)*

Copperhead: Data Parallel Python

- Consider this intrinsically parallel procedure

```
def saxpy(a, x, y):
    return map(lambda xi,yi: a*xi + yi,  x,  y)
```

… or for the lambda averse …

```
def saxpy(a, x, y):
    return [a*xi + yi for xi,yi in zip(x,y)]
```

- This procedure is both
  - completely valid Python code
  - compilable to a corresponding CUDA kernel

Hello GPU programming


```
» from copperhead import *

» @cu
  def saxpy(a, x, y):
      return map(lambda xi,yi: a*xi+yi, x, y)

» x = [1.0, 1.0, 1.0, 1.0]  # can use NumPy or
» y = [0.0, 1.0, 2.0, 3.0]  # CuArrays, too

» gpuResult = saxpy(2.0, x, y)
» cpuResult = saxpy(2.0, x, y, target=cpu0)
```

JIT compile & execute

Best Practices

Use Python for Infrastructure and System Administration Software

- **Use cases**
  - Managing resources
  - Build infrastructure software
  - Analyzing and monitoring systems
  - Administration user interface

- **Why**
  - Python is available on almost any architecture (i.e., any architecture, for which a C compiler exist)
  - Performance of Python not relevant in most cases
  - Python scripts are easier to comprehend and less error prone than shell scripts (with complex AWK expressions etc.)

- **Example**: OpenStack

Use Python as an Integration Language

- **Use cases**
  - Integration of different computational codes written in C, C++, Fortran, …
  - Provide high level script layer to codes

- **Why**
  - Numerical/HPC parts still in more suitable languages
  - Reuse of existing codes provided with high level interface
  - Performance of Python not relevant in most cases

- **Example**: CFD code written in C++ wrapped with Python

Use NumPy/SciPy for Numerical Applications

- **Use cases**
  - Simulations, data analysis, image processing, visualization, data conversion, statistics, …
  - In general, applications usually written in MATLAB etc. today

- **Why**
  - NumPy/SciPy is powerful and has good support
  - Performance of NumPy improves
  - Growing community
  - *see all other advantages of Python*

Prediction: Python+SciPy will replace MATLAB etc. for many applications

Use Python for Students

- **Use cases**
  - "First language" for engineering and science students (probably, except computer science)
  - Teaching numerical algorithms
  - Learning principles of parallel programming

- **Why**
  - Pythons syntax and semantics is easy to learn
  - Code is very much like "pseudo code"… but it executes

Prediction: Growing use of Python in science and engineering demands students with Python knowledge

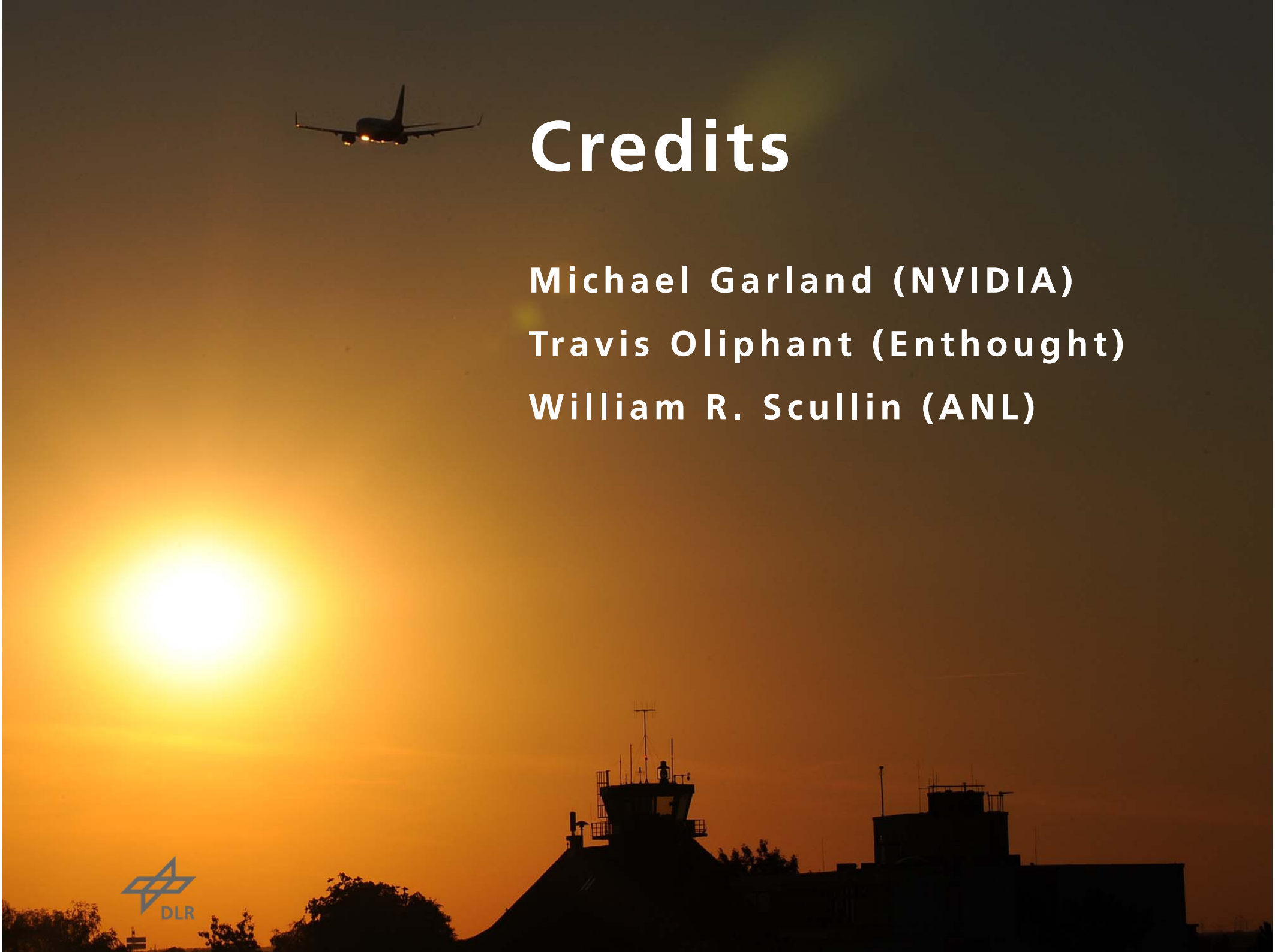
DLR

Use Python Even for Small Scripts

- **Use cases**
  - People write “small” scripts for simple tasks as shell scripts, in Perl, etc.
  - Many small scripts evolve to large software systems
  - People leave the team

- **Why**
  - Python code is much more maintainable than code other languages
  - Other team member can easily join the development
  - No need to re-implement code

Software will be developed by teams, so maintainability and good software engineering practices are important



Credits

Michael Garland (NVIDIA)

Travis Oliphant (Enthought)

William R. Scullin (ANL)

Questions?

Summary

Python is a maintainable language
Good tools available (SciPy etc.)
Best practices evolve

Andreas Schreiber
Andreas.Schreiber@dlr.de
http://www.dlr.de/sc

Contact

python@dlr.de
@onyame

Andreas Schreiber
Andreas.Schreiber@dlr.de
http://www.dlr.de/sc