Python for High Performance and Scientific Computing

EuroPython 2011 (23.06.2011, Florence, Italy)

Andreas Schreiber <*Andreas.Schreiber@dlr.de*>
German Aerospace Center (DLR), Cologne
`http://www.dlr.de/sc`

EUROPYTHON 2011
FLORENCE, JUNE 20–26

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

Abstract

“Python is an accepted high-level scripting language with a growing community in academia and industry. It is used in a lot of scientific applications in many different scientific fields and in more and more industries, for example, in engineering or life science). 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. A related topic is the performance of the various interpreter and compiler implementations for Python.

The talk gives an overview of Python’s use in HPC and Scientific Computing and gives information on many topics, such as Python on massively parallel systems, GPU programming with Python, scientific libraries in Python, and Python interpreter performance issues. The talk will include examples for scientific codes and applications from many domains..“

EuroPython 2011:
http://ep2011.europython.eu/conference/talks/python-for-high-performance-and-scientific-computing

Outline

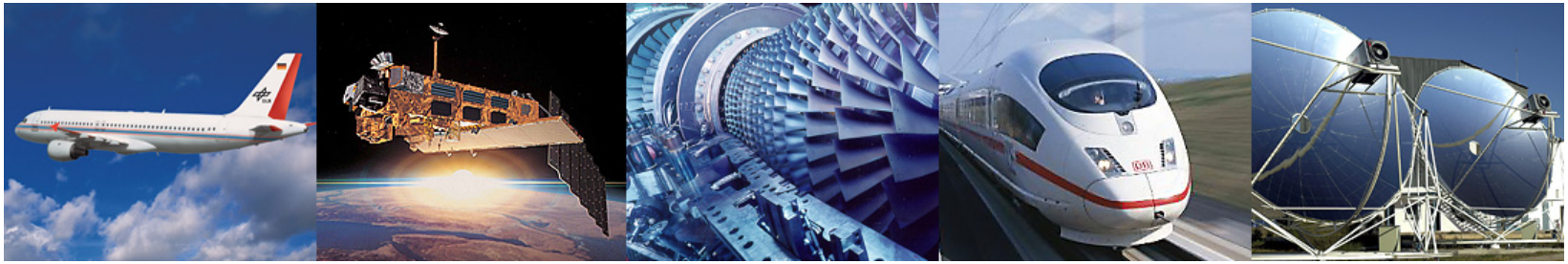
- HPC… and Python
- Tools and Libraries
- Debugging and Profiling
- Example Applications
- Python and HPC communities

Related Talks at EuroPython

- Debugging and Profiling Techniques (Giovanni Bajo)
- Exploit your GPU Power with PyCUDA (and friends) (Stefano Brilli)
- Experiences making CPU-bound tasks run much faster (Ian Ozsvald)
- High-Performance Computing on Gamer PCs (Yann Le Du)
- Python(x,y): Diving into Scientific Python (Vincent Noel)
- and more…


DLR

DLR
German Aerospace Center



- Research Institution
- Space Agency
- Project Management Agency

Locations and employees

6900 employees across
33 institutes and facilities at

- 15 sites.

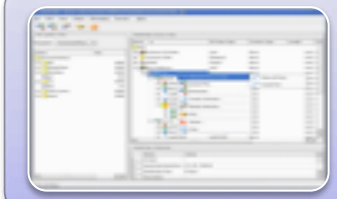
Offices in Brussels,
Paris and Washington.

Software Development at DLR

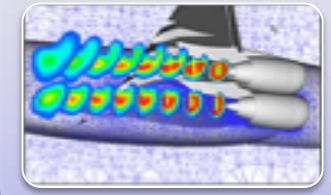
More than 1000 employees develop…

Mission-critical software (e.g., real-time software)

Web-based software for internet and intranet



Supporting software (e.g., data management)

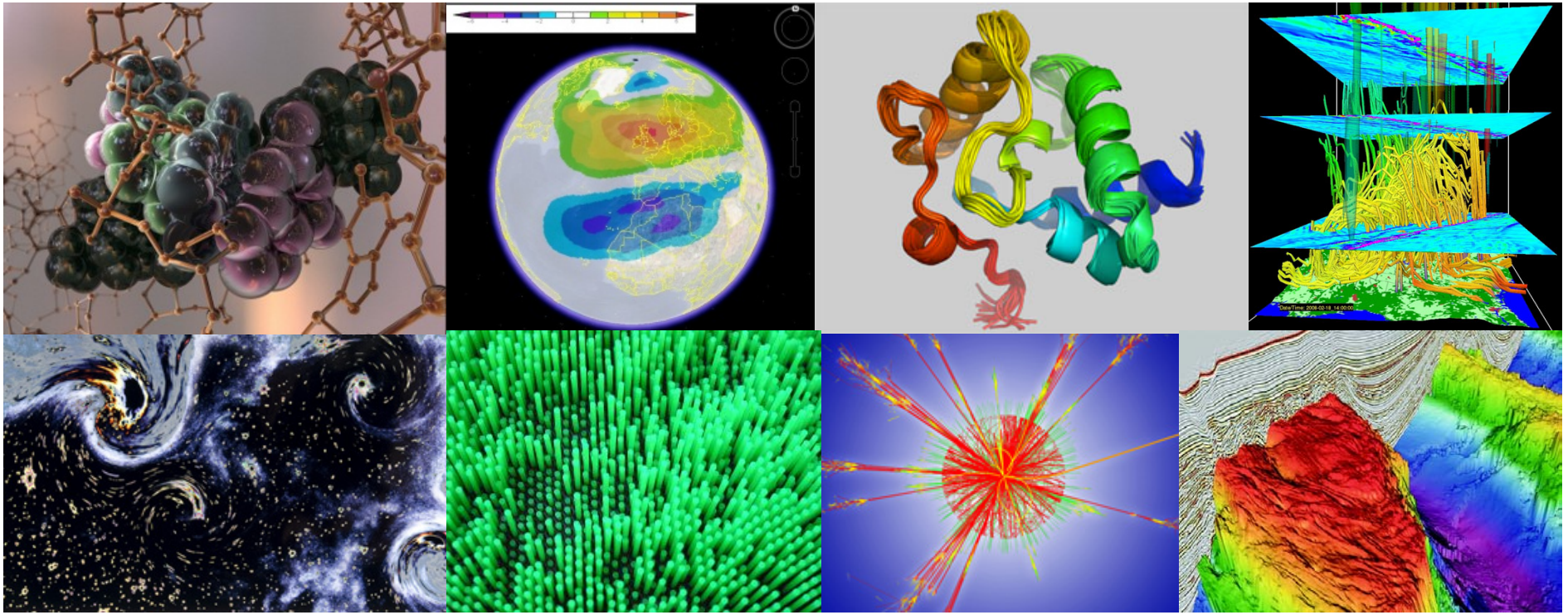


High-Performance and Scientific Computing Software

HPC… and Python

Overview of Applications

Meteorology, Astrophysics, Particle physics, Biology, Genetics, Quantum chemistry, Fluid dynamics, Finance, Oil exploration, …



Images from: http://www.isgtw.org

HPC Systems

- Current top systems have > 500,000 cores and > 8,000 TFlops
- See TOP500 list of supercomputers (http://www.top500.org)
- Top 3 systems (June 2011) are:
  1. K computer, Japan
  2. Tianhe-1A, China
  3. Jaguar, Unites States

#1: *K computer, SPARC64 VIIIfx 2.0GHz, Tofu interc. (RIKEN Advanced Institute for Computational Science (AICS), Japan)*

#2: *Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C (National Supercomputing Center in Tianjin, China)*

#3: *Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz (DOE/SC/Oak Ridge National Laboratory, United States)*

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
- **Education**
- …

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

Why Python is perfect?

- Very easy to learn and easy to use
  (= *steep learning curve*)
- Allows rapid development
  (= *short development time*)
- ***Inherent great maintainability***

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

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

> **Python has the cleanest, most-scientist- or engineer friendly syntax and semantics.**
>
> **Paul F. Dubois**

Tools and Libraries for HPC and SC with Python

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

Scientific Tools and Libraries
Domain Specific Tools

AI
- pyem
- ffnet
- pymorph
- Monte
- hcluster

Biology
- Brian
- SloppyCell
- NIPY
- PySAT

Molecular & Atomic Modeling
- PyMOL
- Biskit
- GPAW

Geo sciences
- GIS Python
- PyClimate
- ClimPy
- CDAT

Electromagnetics
- PyFemax

Astronomy
- AstroLib
- PySolar

Dynamic Systems
- Simpy
- PyDSTool

Finite Elements
- SfePy

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...

NumPy
Arrays

- A NumPy array is an N-dimensional homogeneous collection of “items” of the same “kind”. The kind can be any arbitrary structure and is specified using the data-type.
- Example: Multidimensional NumPy array

```
>>> a = array([[ 0, 1, 2, 3],
               [10,11,12,13]])
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,13]])
```

```
>>> a[1,3]
13
>>> a[1,3] = -1
>>> a
array([[ 0, 1, 2, 3],
       [10,11,12,-1]])
```

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

SciPy
Packages

- Special Functions
- Signal Processing
- Fourier Transforms
- Optimization
- Numerical Integration
- Linear Algebra
- Input/Output
- Statistics
- Fast Execution
- Clustering Algorithms
- Sparse Matrices

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

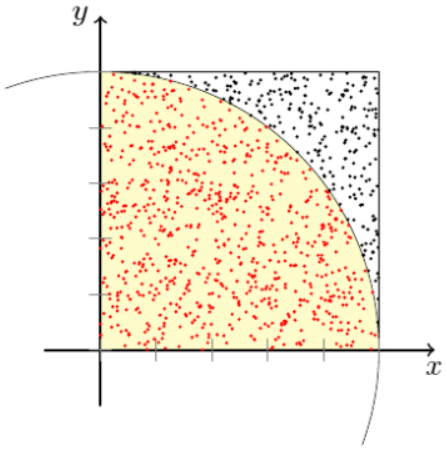
mpi4py

- 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

How mpi4py works...

- mpi4py jobs must be launched with *mpirun*
- Each rank launches its own independent python interpreter
- Each interpreter only has access to files and libraries available locally to it, unless distributed to the ranks
- Communication is handled by MPI layer
- Any function outside of an if block specifying a rank is assumed to be global
- Any limitations of your local MPI are present in mpi4py

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
```

GPGPU

General-purpose computing on graphics processing units

Architecture

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

CUDA

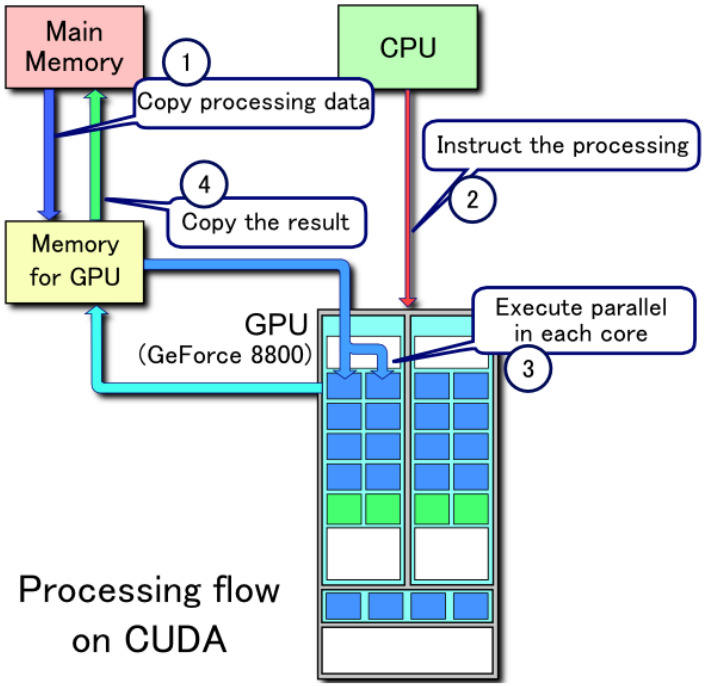
- NVIDIA's platform for C programming on GPGPUs
- Python binding: PyCUDA

OpenCL

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

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

OpenCL Computing Model

Buffers

- used for holding data, shipping data between devices

Kernels

- functions which will operate on buffers in parallel

Queue

- control sequence of operations: transferring buffers, running kernels, waiting on events, etc.

π with PyOpenCL

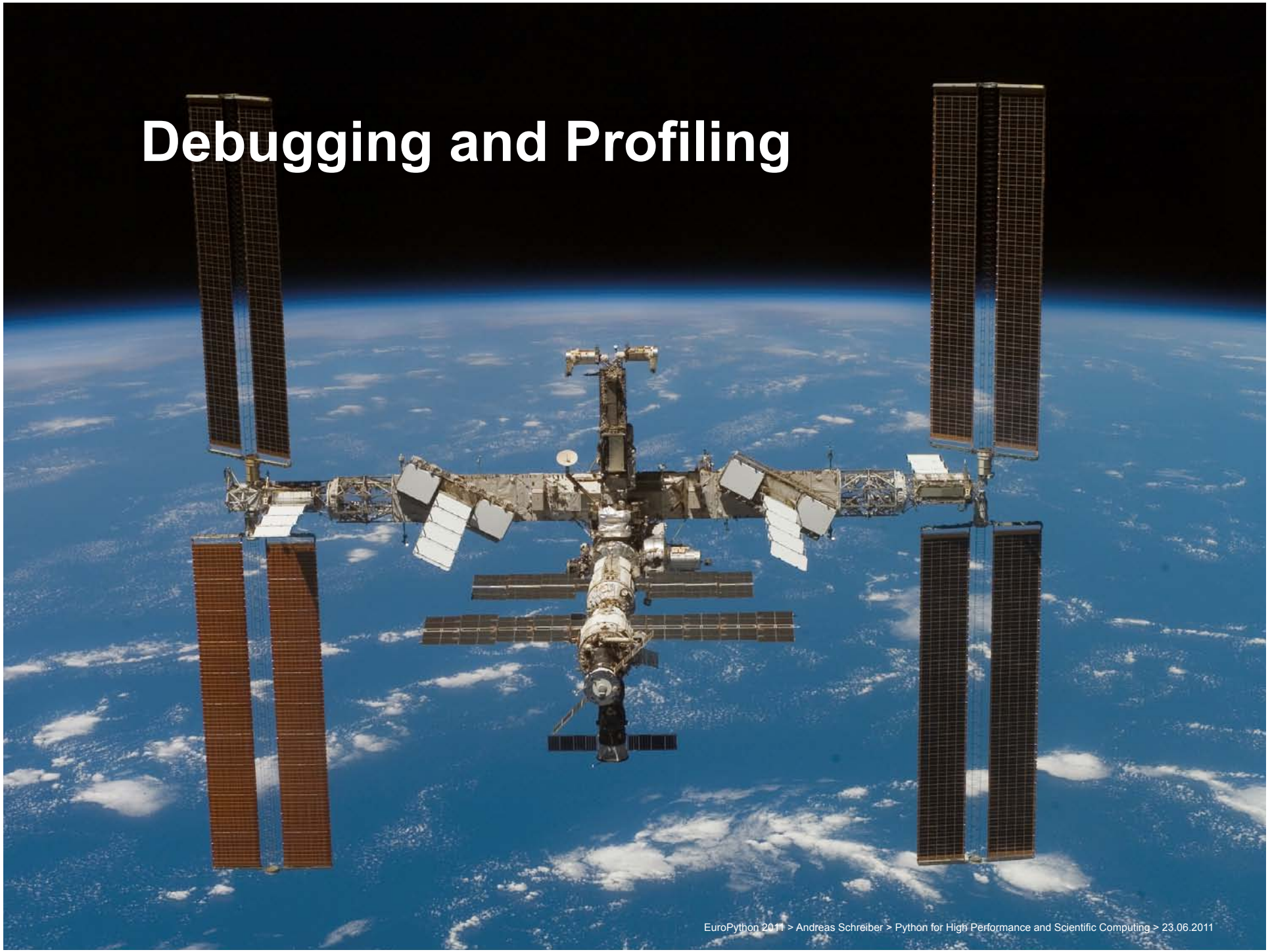
```
import pyopencl as cl
import pyopencl.clrandom
import numpy as np

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

Slides by Michael Garland (NVIDIA Research)

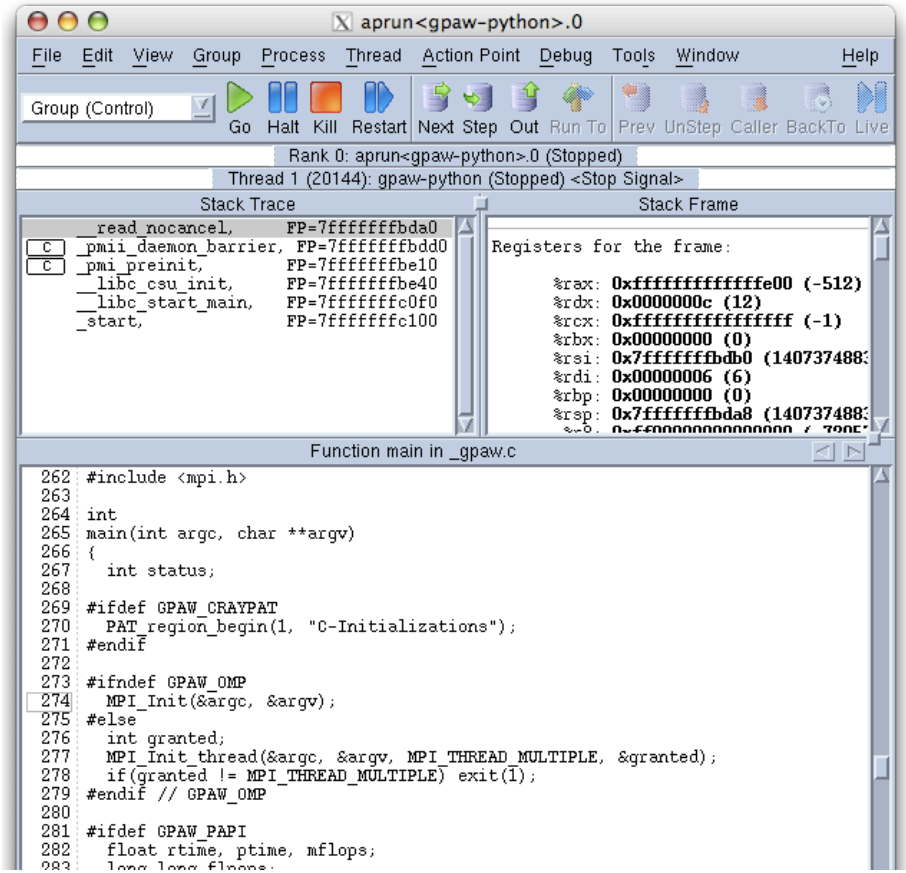
http://bit.ly/jK7nsM

Debugging and Profiling



Debugging Python and C

- Python comes with gdb-like command line debugger pdb
- Helps in debugging serial Python applications
- C-extensions can be debugged with standard debuggers
- Parallel debugging
  - Totalview can be used for debugging C-functions
  - Custom interpreter provides easy initial breakpoint



Profiling Mixed Python-C code

- Finding performance bottlenecks is critical to scalability on HPC platforms
- Number of profiling tools available:
  - gprof, CrayPAT, scalasca - C, Fortran
    - can be used for C-extensions
  - import profile – Python

- TAU Performance System, http://www.cs.uoregon.edu/research/tau/home.php
  - *Excurse: Slides by Sameer Shende (University of Oregon)*

TAU Performance System

Slides by Sameer Shende (University of Oregon)

http://slidesha.re/lsDZSj

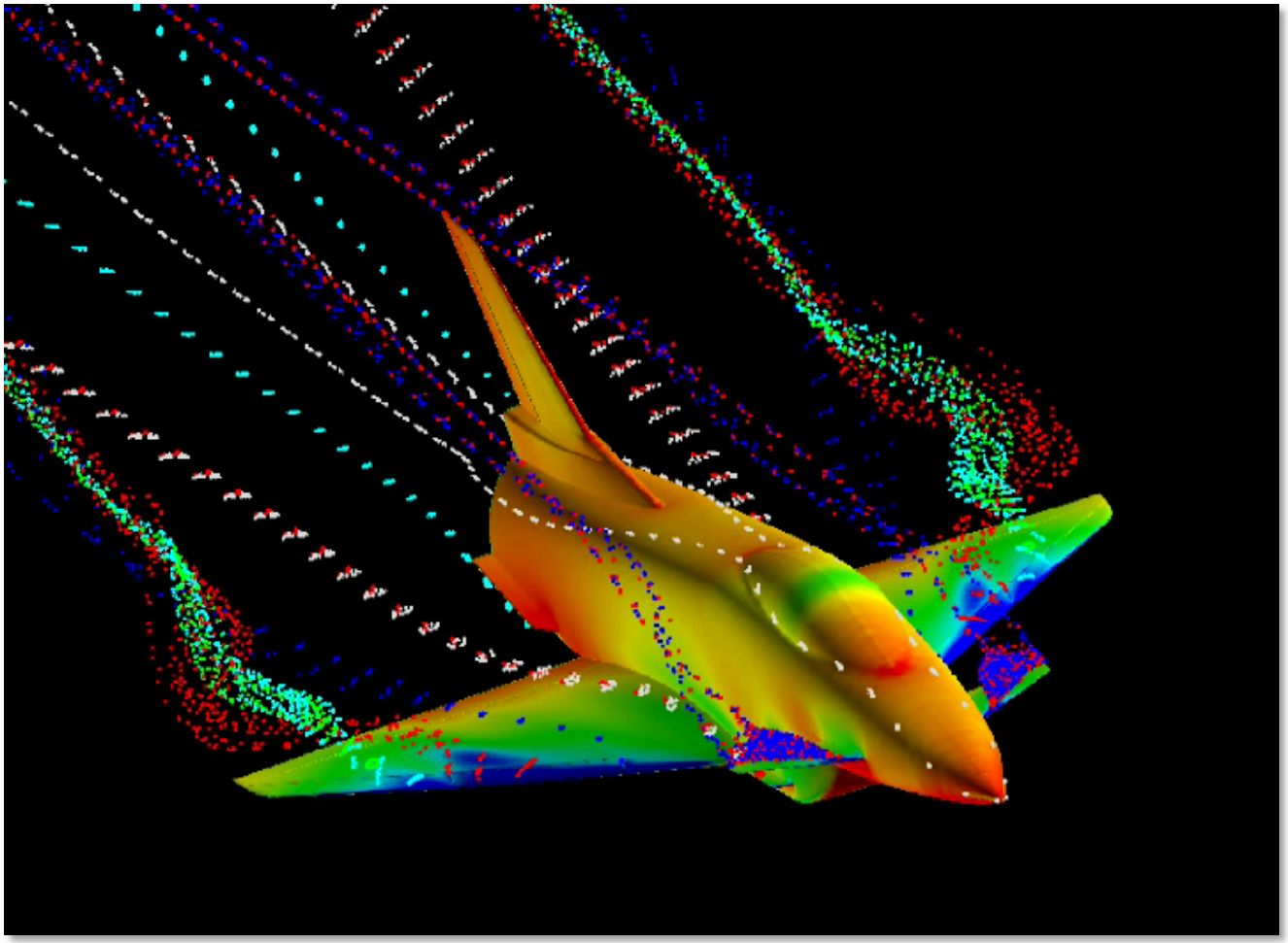
Example Applications

Example:
GIST Maxwell's Equation Solver (GMES)

- Object-Oriented Implementation of the Finite-Difference Time-Domain Method in Parallel Computing Environment
- *Slides by Kyungwon Chun et al. (GIST - Gwagnju Institute of Science and Technology, South Korea)…*

http://bit.ly/mCs8W9

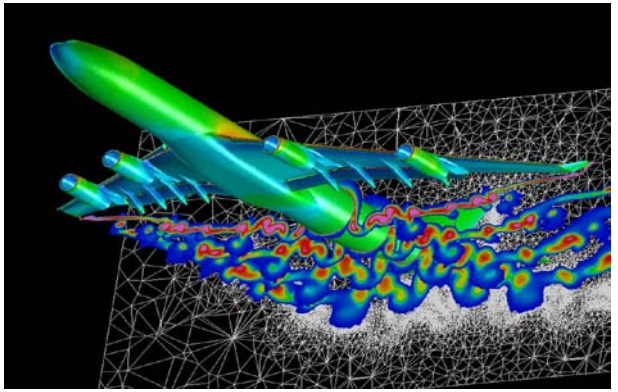
Example: **FlowSimulator –** A Python-controlled framework to unify massive parallel CFD workflows

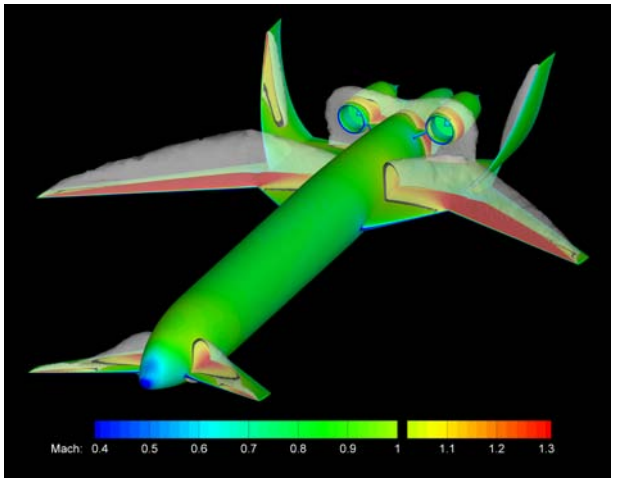


What does CFD mean?

- CFD = Computational fluid dynamics means to
  - solve the compressible flow equations to
  - predict an aircrafts behavior and to
  - optimize its features.

- Numerical approach:
  - Spatial discretization
  - Simplified models for turbulent effects
  - Temporal discretization (to support unsteady flows)

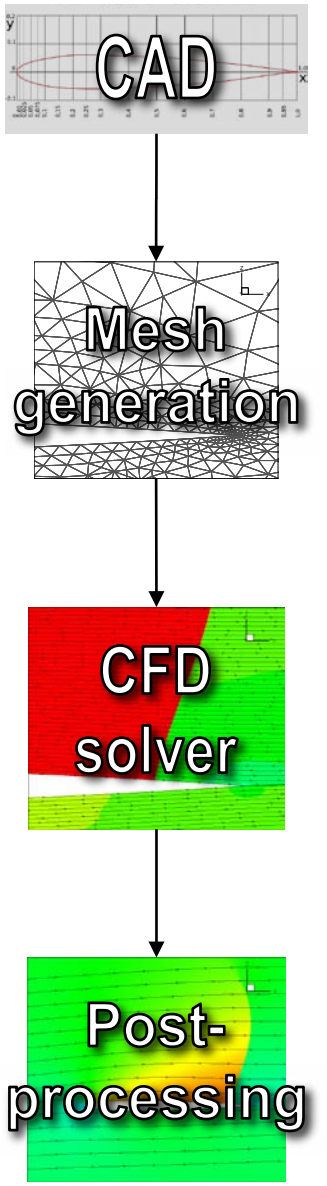




Evolution of CFD

Classical chain

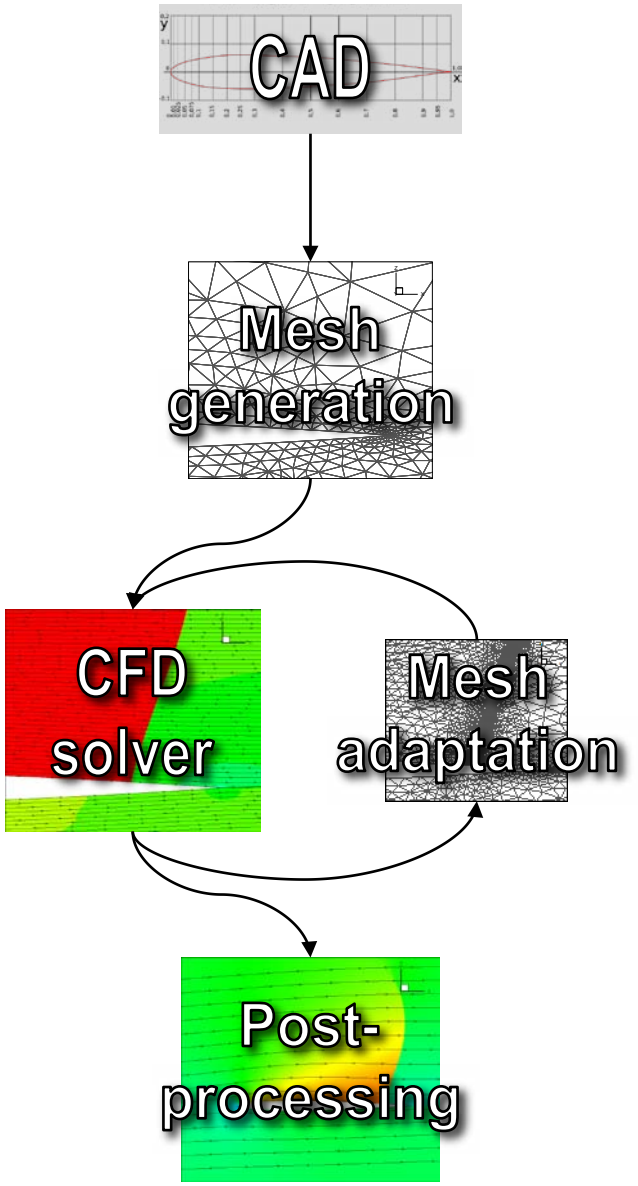
- Simple chain
  - Geometry
  - Mesh generator
  - Pre-processor
  - Flow solver
  - Post-processor
  - Visualization
- Tools
  - stand-alone mesher
  - stand-alone solver (with pre-/postprocessor)
  - shell scripts (e.g. different solver parameters)



Evolution of CFD

Solver-adaptation-loop

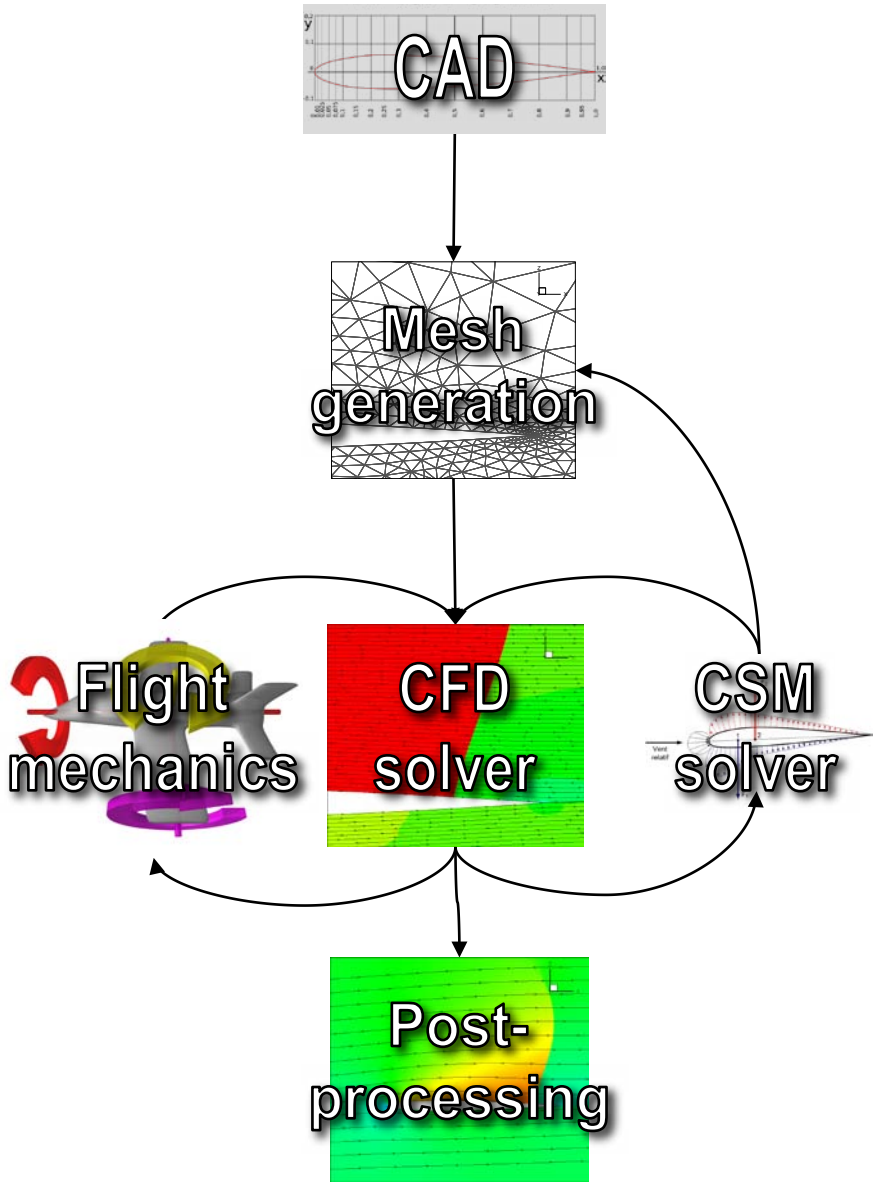
- Chain with solver-adaptation-loop
  - several consecutive solver steps
  - adaptation of mesh in between
- Tools:
  - stand-alone mesher
  - stand-alone solver
  - stand-alone adaptation
  - shell script control
- Solver/adaptation as Python module



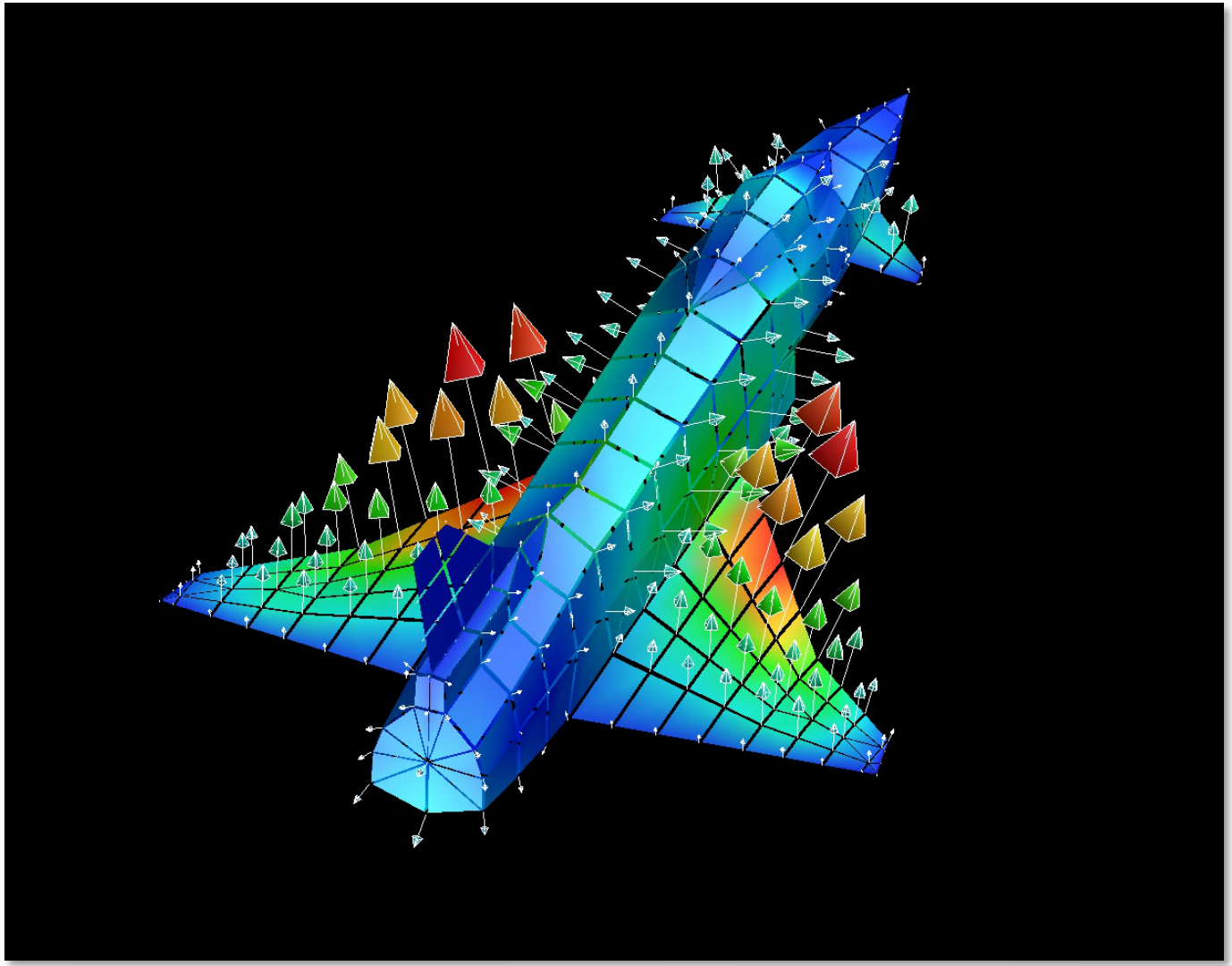
Evolution of CFD

Multi-disciplinary simulations

- Multi-disciplinary chain
  - output of solver is input for
    - structural mechanics (deformation)
    - flight mechanics (control surface movement)
  - output of structural/flight mechanics modifies mesh
- Tools:
  - solver as Python module
  - stand-alone CSM
  - Python-based flight mechanics

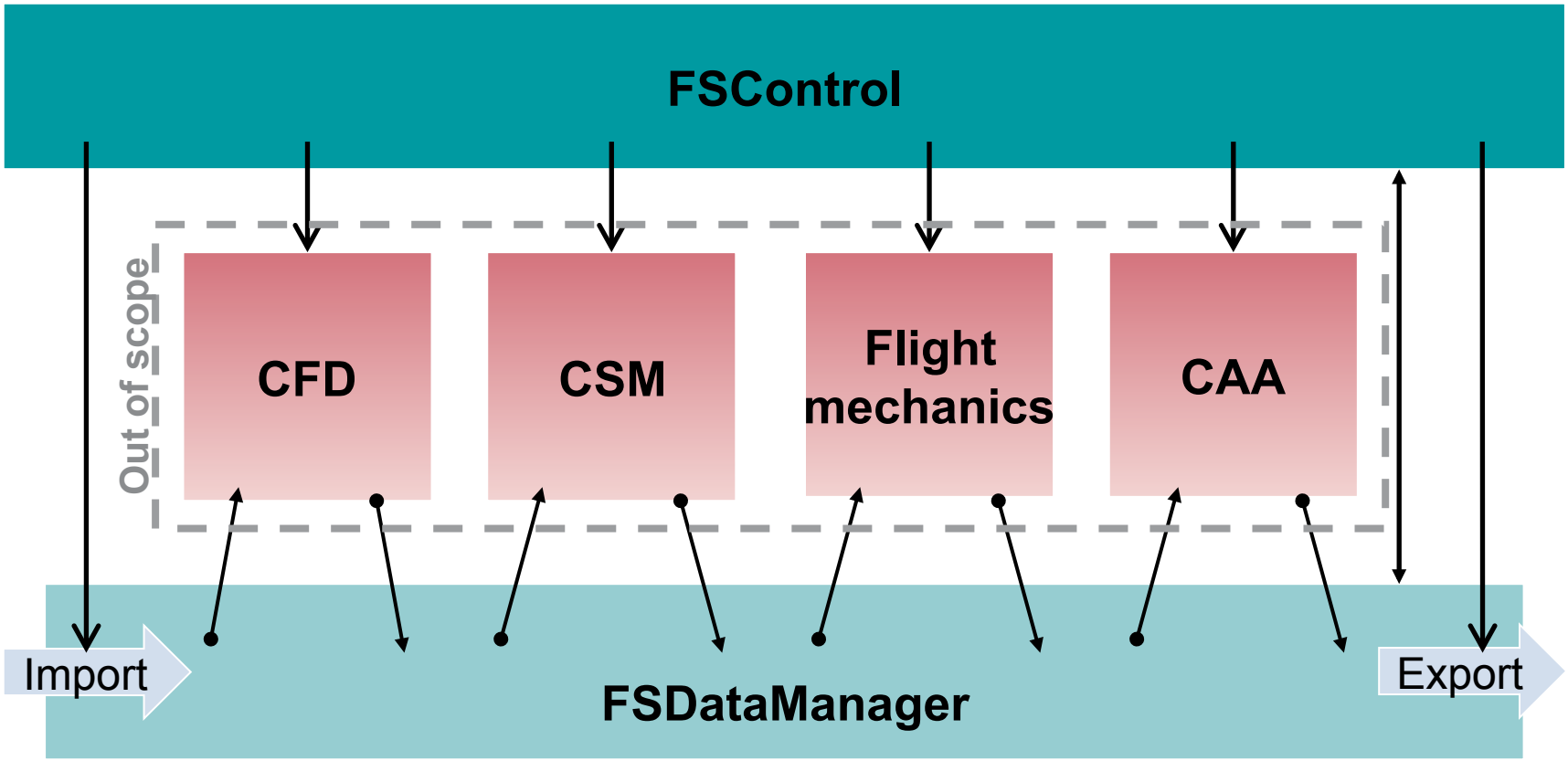


Structure and Flight Mechanics



FlowSimulator Environment

Single Python script controls the whole workflow



FSControl

- Pure Python library
- Set of simple Python classes for different modules/tools:
  - simple interface: Import, Run, Export
  - data stored in FSDataManager

FSDataManager (FSDM)

- C++ library, wrapped to Python
  - FSDataManager is a real Python module!
- Parallelization context
  - all data structures “distributed”
  - facilitated transfers (Gather, Distribute, Send, …)

Python Framework for Coupled Fusion Simulations
Slides by Samantha Foley et al. (ORNL – Oak Ridge National Laboratory, United States)

http://slidesha.re/iLuegW

Python & HPC Communities

Where the Python and HPC Communities Meet…

- **SciPy**
  - Python for Scientific Computing Conference (since 2008)
- **EuroSciPy**
  - Annual European Conference for Scientists using Python (since 2008)
- **SCxx**
  - The International Conference for High Performance Computing, Networking, Storage, and Analysis (annual conference since 1988)
  - Tutorial and BoF session on Python (since 2009)

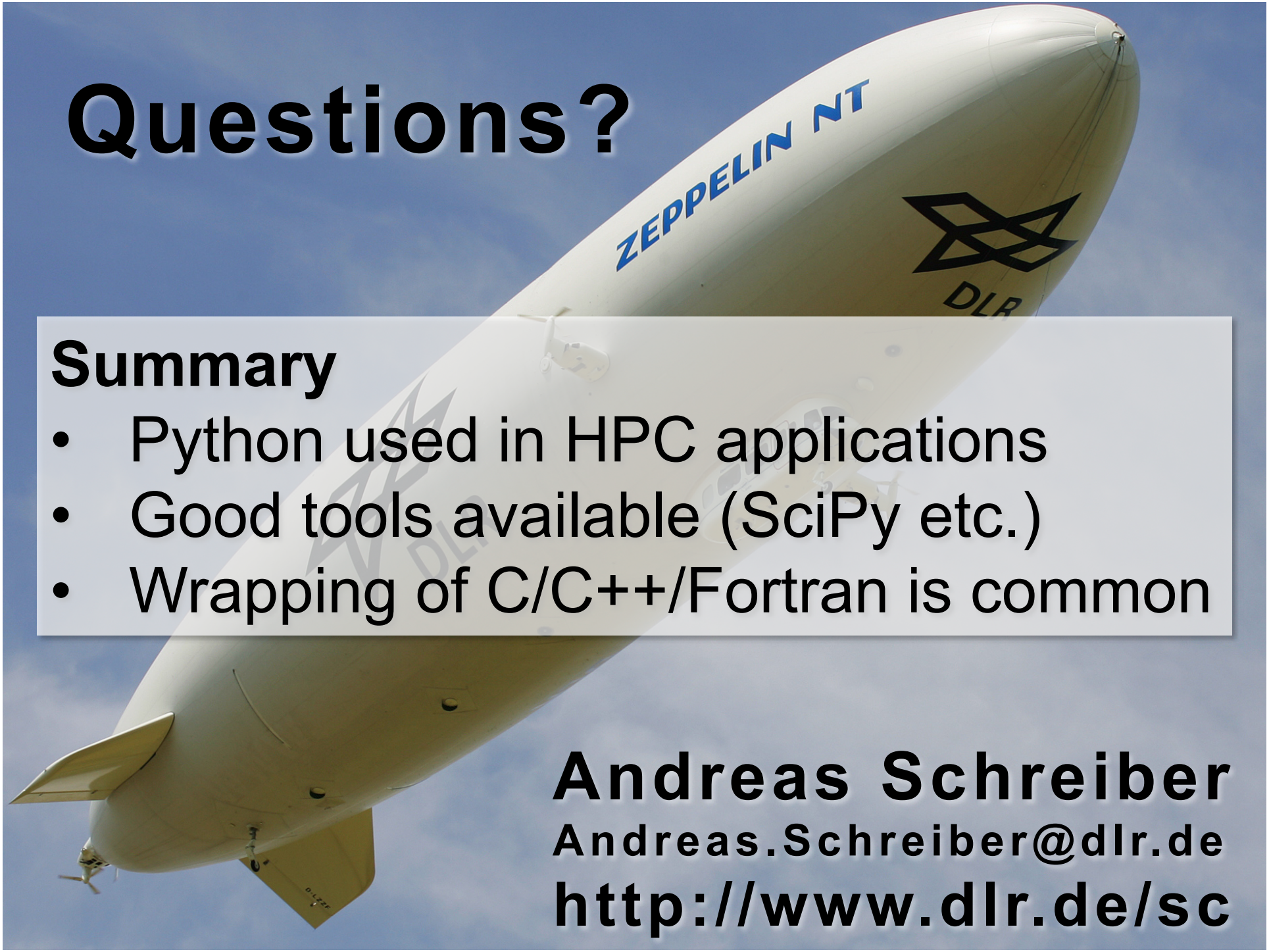
The SC Conference Series (SCxx)
Tutorials (est. 2009)

- Python for Scientific and High Performance Computing
- Given by William R. Scullin (ANL – Argonne National Lab, United States) et al.
- Content
  - *Python language and interpreter basics*
  - *Popular modules and packages for scientific applications*
  - *How to improve performance in Python programs*
  - *How to visualize and share data using Python*
  - *Where to find documentation and resources*

The SC Conference Series (SCxx)

Birds-of-a-Feather Sessions, est. 2009


Credits
Achim Basermann (DLR)
Kyungwon Chun (GIST)
Samantha Foley (ORNL)
Michael Garland (NVIDIA)
Michael Meinel (DLR)
Travis Oliphant (Enthought)
William R. Scullin (ANL)
Sameer Shende (U Oregon)



Questions?

Summary

- Python used in HPC applications
- Good tools available (SciPy etc.)
- Wrapping of C/C++/Fortran is common

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