Python in HPC

IDEAS Webinar
June 7, 2017

Rollin Thomas (NERSC), William Scullin (ALCF), and Matt Belhorn (OLCF)

Scope of This Webinar

What we want to do:

- Explain what NERSC, ALCF, and OLCF are doing to welcome and support Python users in HPC.
- Provide guidance and best practices to help users improve Python performance at the Centers.
- Point out some great tools that now exist to support developers of Python in HPC.

What we assume:

- You know and use Python and are familiar with the Scientific Python Stack, or
- You know and use HPC and are curious about using Python in your own HPC work.

Getting Started with Python Resources

https://www.python.org/about/gettingstarted/

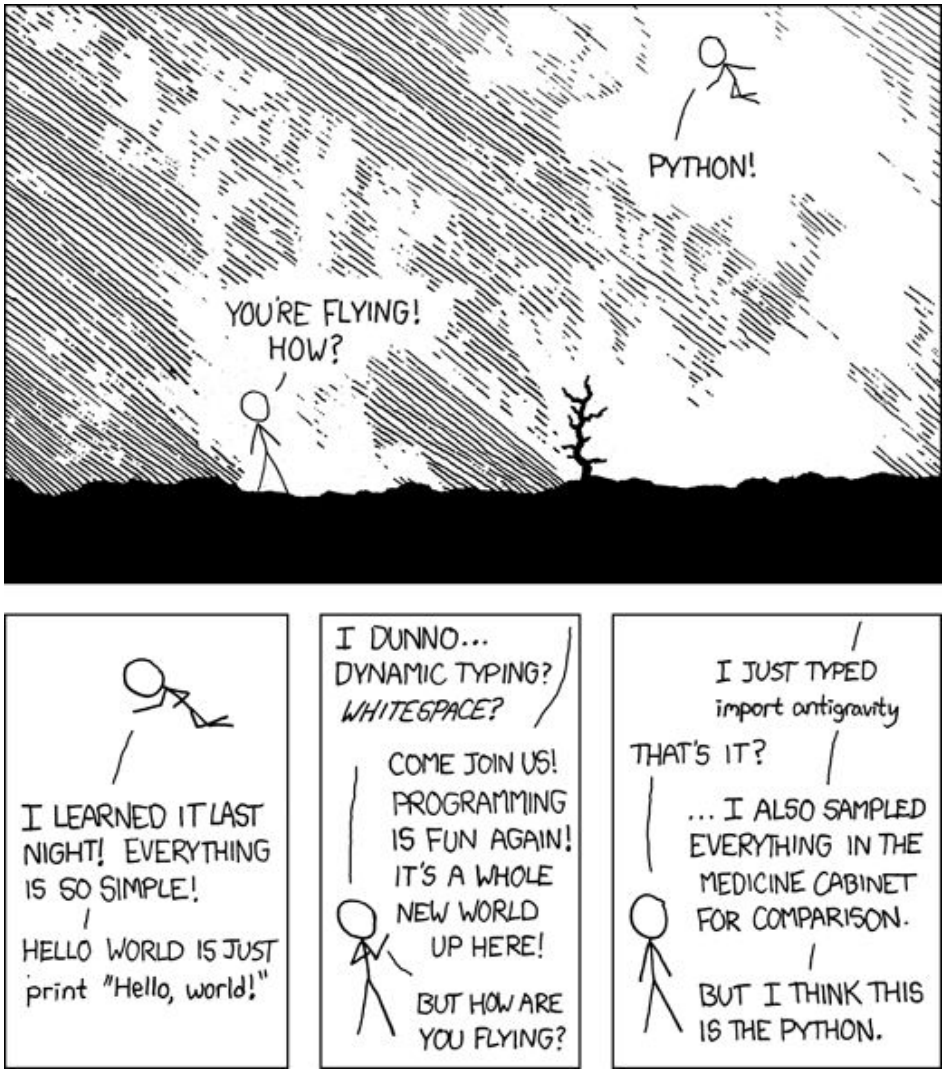
https://wiki.python.org/moin/BeginnersGuide

https://www.codecademy.com/learn/python

https://www.coursera.org/specializations/python

https://software-carpentry.org/lessons/

https://pymotw.com/



https://xkcd.com/353/

Agenda

- **Motivation**

  How is Python relevant to HPC?

- **Practical Matters**

  Using Python at NERSC, ALCF, and OLCF

- **Single Node Performance**

  Threads ● Cython, Extensions ● Profiling

- **Scaling Up Python**

  MPI(4py) ● Caveats ● Parallel I/O

- **Conclusion & More Resources**

[We will pause for 1-2 questions at each breakpoint,
Matt will manage Q&A via Webex chat.]

Motivation

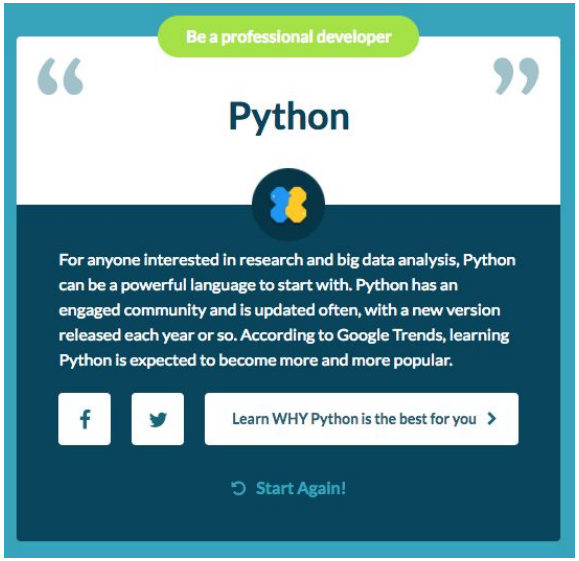
How is Python relevant to HPC?

Python is a Very Popular Language

Aug 2016	Aug 2015	Change	Programming Language	Ratings	Change
1	1		Java	19.010%	-0.26%
2	2		C	11.303%	-3.43%
3	3		C++	5.800%	-1.94%
4	4		C#	4.907%	+0.07%
5	5		Python	4.404%	+0.34%
6	7	^	PHP	3.173%	+0.44%
7	9	^	JavaScript	2.705%	+0.54%
8	8		Visual Basic .NET	2.518%	-0.19%
9	10	^	Perl	2.511%	+0.39%
10	12	^	Assembly language	2.364%	+0.60%

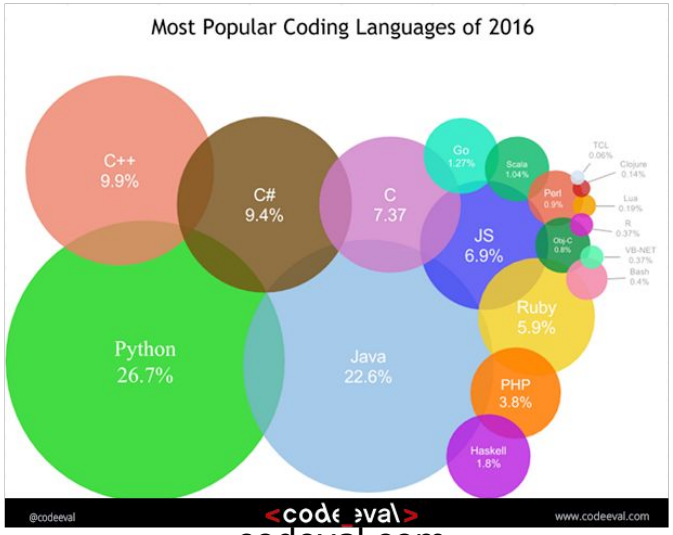
www.tiobe.com/tiobe-index

"What programming language should I learn?"



bestprogramminglanguagefor.me

"What programming languages are good for Data Science?"



codeval.com

"What programming languages are widely used in industry, science, or ML/coding challenges?"

Why is Python Popular?

Makes a great first impression:

- Clean, clear syntax.
- Multi-paradigm, interpreted.
- Duck typing, garbage collection.
- **“Instant productivity!”**

Keeps up with users’ needs:

- Flexible, full-featured data structures.
- Extensive standard libraries.
- Reusable open-source packages (PyPI).
- Package management tools.
- Good unit testing frameworks.
- Extensible with C/C++/Fortran for optimizing high-performance kernels.
- **“Instant productivity, performance when you need it” (?)**

```
from interface import Model

class BasicModel ( Model ) :

    def __init__( self, gaussian_process, training_data, upda
        self.gaussian_process = gaussian_process
        self.training_data    = training_data

        training_size       = len( self.training_data )
        self._input_diffs   = ( self.training_data.inputs[ No
                self.training_data.inputs[ :, None ] )

        self._gram          = numpy.zeros( ( training_size, t
        self._log_gram_det  = None
        self._inv_gram      = numpy.zeros_like( self._gram )
        self._residuals     = numpy.zeros( training_size )
        self._inv_gram_resp = numpy.zeros( training_size )

        if update :
            self._update()

    @property
    def log_p( self ) :
        return -0.5 * ( numpy.dot( self._residuals, self._inv
                self._log_gram_det + len( self.training_data
                numpy.log( 2.0 * numpy.pi ) )

    @property
    def hyperparameters( self ) :
        deque = self.gaussian_process.mean_function.hyperpara
        deque.extend( self.gaussian_process.covariance_functi
        return deque

    @hyperparameters.setter
    def hyperparameters( self, iterable ) :
        deque = collections.deque( iterable )
        self.gaussian_process.mean_function.take_hyperparamet
```

The Scientific Python Stack

Primary Uses:

- **Script workflows for both data analysis and simulations**
- **Perform exploratory, interactive data analytics & viz**

Python at the HPC Center

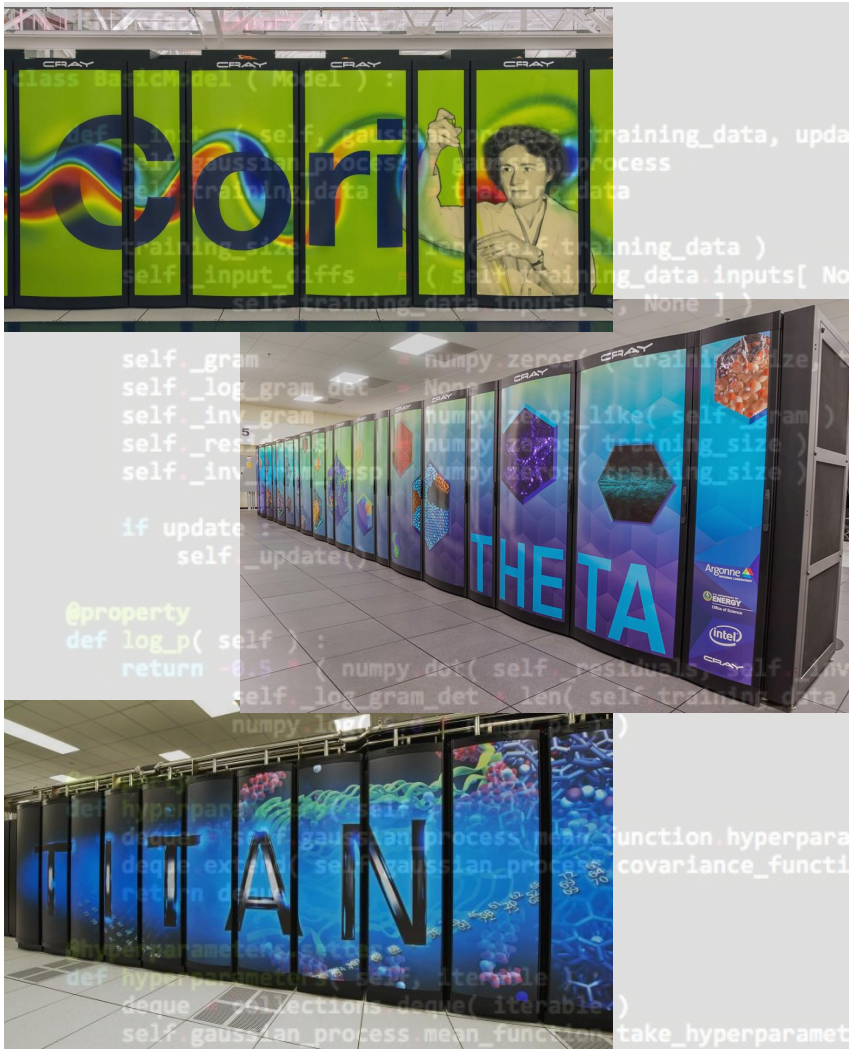
Observation: ***High productivity*** has driven the growth of Python in the sciences.

...Not ***high performance*** (so much).

But supporting Python is no longer optional at HPC centers like NERSC, ALCF, OLCF.

Maximizing Python performance on these systems can be *(ok, is)* challenging:

- Interpreted, dynamic languages are difficult to optimize.
- Python's global interpreter lock (GIL) has consequences for parallelism.
- Language design and implementation choices made without considering realities of HPC.



PyFR: Gordon Bell & SC16 Best Paper Finalist

Towards Green Aviation with Python at Petascale

Peter Vincent*, Freddie Witherden†, Brian Vermeire‡, Jin Seok Park§ and Arvind Iyer¶
Department of Aeronautics, Imperial College London, London, United Kingdom

- Demonstrated use of Python in a high-end HPC context for simulation of real-world flow problems at up to 13.7 DP-PFLOP/s.
- Detailed how a single Python codebase can target multiple platforms, including heterogeneous systems, using an innovative runtime code-generation paradigm.
- Achieved 58% computational efficiency for an un-structured mesh fluid dynamics simulation.

- Performance portability enabled by Python:
  CFD from a single code base supporting CPU and GPU architectures, a few x1000 lines of code.
- **There is a place for Python at the highest levels of performance in supercomputing.**

[http://sc16.supercomputing.org/2016/08/23/finalists-compete-prestigious-acm-gordon-bell-prize-high-performance-computing/]
[http://sc16.supercomputing.org/2016/09/21/sc16-announces-best-paper-nominees]

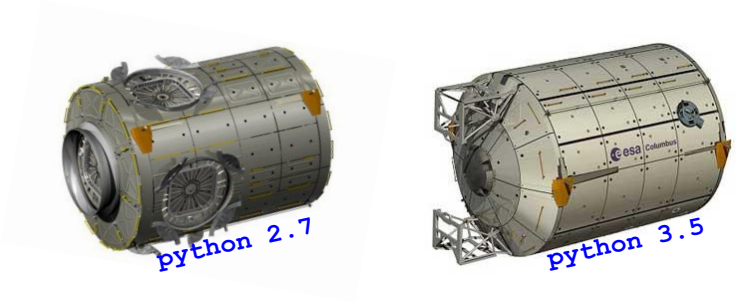
Basic Guidelines for Python in HPC

- **Identify and exploit parallelism at the core, node, and cluster levels.**
- Understand and apply numpy array syntax and its broadcasting rules (skipped here):
  [https://docs.scipy.org/doc/numpy/reference/arrays.html]
  [https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html]
- Measure your codes' performance using profiling tools.
- Ask for help.

Practical Matters

Using Python at NERSC, ALCF, & OLCF

Python at NERSC, ALCF, & OLCF



- **Environment Modules**

  [http://modules.sourceforge.net]

  "The Environment Modules package provides for the dynamic modification of a user's environment via modulefiles."

  ```
  module avail python
  module load python
  module swap python/2.7 python/3.5
  module help…
  ```

- **Or install your own Python (many options).**
- **System Python (e.g. `/usr/bin/python`), *use at your own risk.***

Python Builds and Distributions

Centers may build, install Python & packages from

… source,

… package managers like Spack*,

Spack
[https://spack.readthedocs.io/en/latest/]

… using distribution like Anaconda “and/or” Intel,

ANACONDA Powered by Continuum Analytics intel
[https://docs.continuum.io/anaconda/]
[https://software.intel.com/en-us/distribution-for-python]

… or all of the above.

Centers also let users set up their own!

- Packages depending on MPI should ***always*** be built against system vendor-provided libraries.
- Anaconda distribution comes with Intel MKL built-in. Intel distribution heavily leverages Anaconda tools.

[* Spack: An upcoming IDEAS Webinar topic.]

Customizing I: Virtualenv

User-controlled isolated python environments

- Site packages root under your control
- Activated venvs preclude other python interpreters
- Semi-conflicts with environment modules
  - Setup environment modules prior to activation

```
$ virtualenv -p python2.7 /path/to/my_env
$ . /path/to/my_env/bin/activate
(my_env)$ pip install --trusted-host \
                pypi.python.org -U pip
(my_env)$ CC=cc MPICC=cc pip install -v \
                  --no-binary :all: mpi4py
(my_env)$ deactivate
```

Customizing I: Virtualenv (cont'd)

Your packages with an external interpreter

- Install your own packages in your venv
- Use them with external python within your python scripts
- Mix-and-match with center-provided packages

```
#!/usr/bin/env python2.7
activate_this = '/path/to/env/bin/activate_this.py'
execfile(activate_this, dict(__file__=activate_this))
```

N.B.: Packages installed in the venv will supercede versions installed at the site level.

Customizing II: Conda environments

Anaconda provides the *conda* tool:

[https://conda.io/docs/index.html]

- Create, update, share “environments.”
- Incompatible with virtualenv, replaces it.
- Many pre-built packages organized in custom “channels.”
- Leverage your center’s Anaconda install to create custom environments with the conda tool.

Your own Anaconda/“Miniconda” installation:

```
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
/bin/bash Miniconda2-latest-Linux-x86_64.sh -b -p $PREFIX
source $PREFIX/bin/activate
conda install basemap yt…
```

Your own Intel Python Installation:

Text
Text

```
conda create -c intel -n idp intelpython2_core python=2
source activate idp
```

conda create -c intel -n py3intel intelpython3_full python=3
source activate py3intel

Python at NERSC

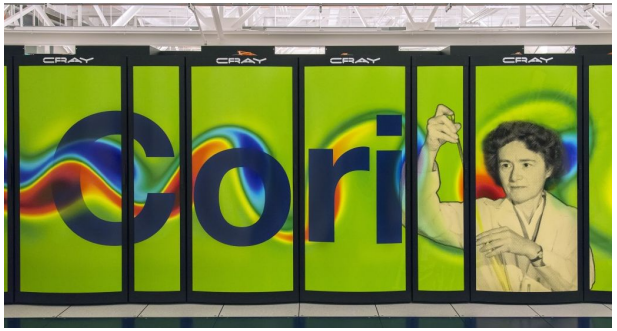
NERSC-built:

```
module load python[/2.7.9]
python_base/2.7.9
numpy/1.9.2
scipy/0.15.1
matplotlib/1.4.3
ipython/3.1.0
```

[default]

Anaconda:

```
module load python/2.7-anaconda
module load python/3.5-anaconda
```



NERSC-built:

None

[default]

Anaconda:

```
module load [python/2.7-anaconda]
module load python/3.5-anaconda
```

Conda env via module (either system)

```
module load python/2.7-anaconda
conda create -n myenv numpy...
source activate myenv
```

[http://www.nersc.gov/users/data-analytics/data-analytics/python/]

Python at ALCF

- Every system we run is a cross-compile environment except Cooley
- pip/distutils/setuptools/anaconda **don't play well with cross-compiling**
- Blue Gene/Q Python is manually mantained
  - Instructions on use are available in: /soft/cobalt/examples/python
  - Modules built on request
- Theta offers Python either as:
  - Intel Python - managed and used via Conda
    - We prefer users to install their own environments
    - Users will need to set up their environment to use the Cray MPICH compatibility ABI and strictly build with the Intel MPI wrappers: http://docs.cray.com/books/S-2544-704/S-2544-704.pdf
  - ALCF Python managed via Spack and loadable via modules

    ```
    module load alcfpython/2.7.13-20170513
    ```

    - A module that loads modules for NumPy, SciPy, MKL, h5py, mpi4py...
    - We build and rebuild alcfpython via Spack to emphasize performance and Cray compatibility
    - Use of virtualenv is recommended - do not mix conda and virtualenv!!!
    - We'll build any package with a Spack spec on request

Python at OLCF

Provided interpreters:

```
module load python[/2.7.9]
             python/3.5.1
```

Major Provided Packages:

```
python_numpy/1.9.2
python_scipy/0.15.1
python_matplotlib/1.2.1
python_ipython/3.0.0
python_mpi4py/1.3.1
python_h5py/2.6.0
python_netcdf4/1.1.7
```

Anaconda:

- Prefer to build your own
- Generally interferes with Tcl Environment Modules

Custom package install paths:

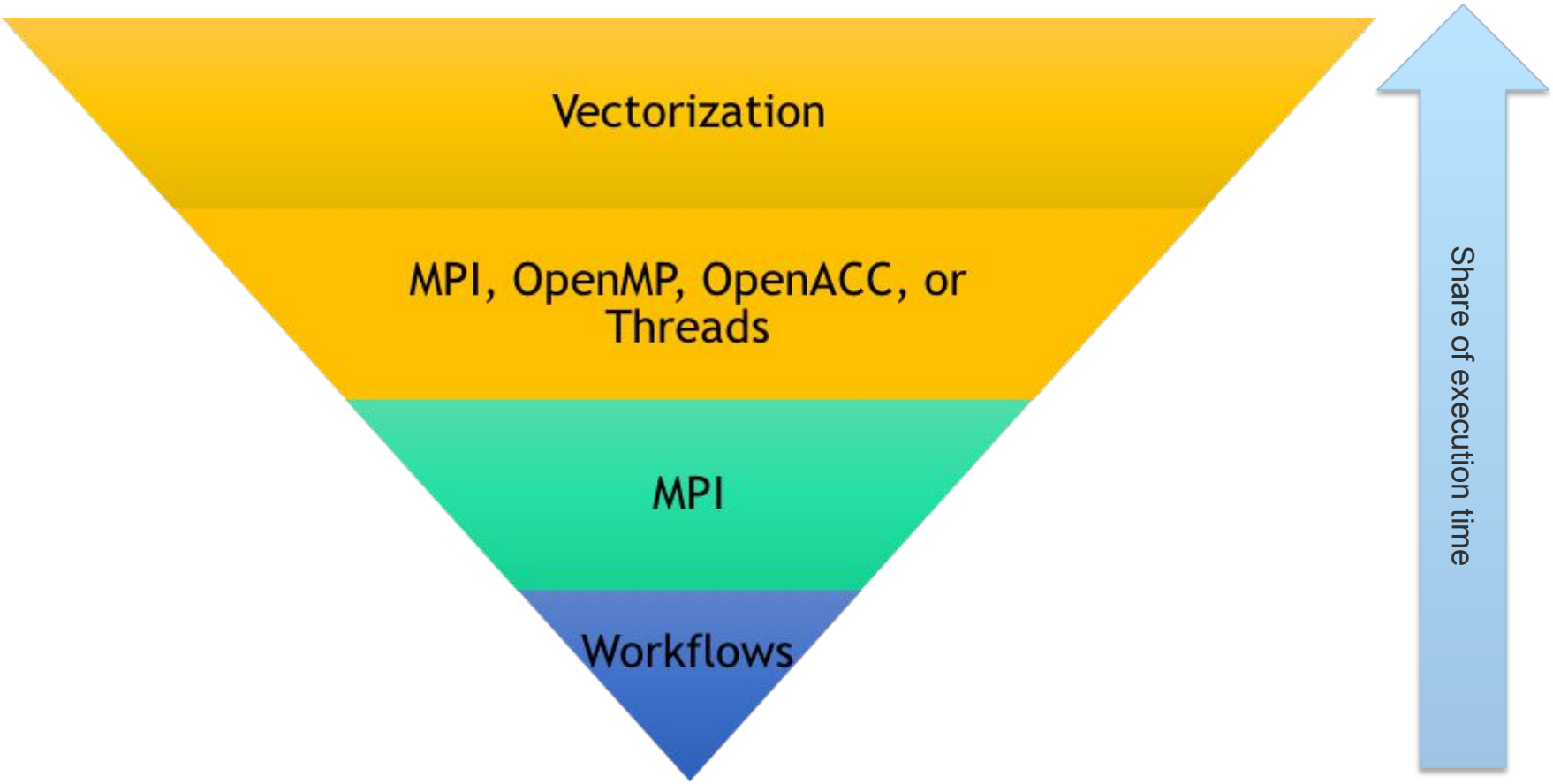
- Prefer NFS project space **`/ccs/proj/${PROJECTID}`**
- Take care with user site-packages, **`${HOME}`**
- Avoid **`/lustre/atlas`**

Further site-specific information on the OLCF Website
[https://www.olcf.ornl.gov/training-event/2016-olcf-user-conference-call-olcf-python-best-practices]

Single Node Performance

Threads ● Cython, Extensions ● Profiling

Structuring a HPC Python code



Parallelism & Python: A Word on the GIL

To keep memory coherent, Python only allows a single thread to run in the interpreter's memory space at once. This is enforced by the Global Interpreter Lock, or GIL.

The GIL isn’t all bad. It:

- Is mostly sidestepped for I/O (files and sockets)
- Makes writing modules in C much easier
- Makes maintaining the interpreter much easier
- Encourages the development of other paradigms for parallelism
- Is almost entirely irrelevant in the HPC space as it neither impacts MPI or threads embedded in compiled modules

For the gory details, see David Beazley's talk on the GIL:
https://www.youtube.com/watch?v=fwzPF2JLoeU

Use Threaded Libraries

- Building blocks like NumPy and SciPy are already built with MPI and thread support via BLAS/LAPACK, MKL
- Don’t reimplement solvers in pure Python
- Many of your favorite threaded libraries and packages already have bindings:
  - PyTrilinos
  - petsc4py
  - Elemental
  - SLEPc

Using Compiled Modules

Methods of using pre-compiled, threaded, GIL-free code for speed include:

- Cython
- f2py
- PyBind11
- swig
- Boost
- Ctypes
- Writing bindings in C/C++
  http://dan.iel.fm/posts/python-c-extensions/

More control: Cython

Cython is a meant to make writing C extensions easy

- Naive usage can offer x12 speedups
- Builds on Python syntax
- Translates .pyx files to C which compiles
- Provides interfaces for using functionality from OpenMP, CPython, libc, libc++, NumPy, and more
- Works best when you can statically type variables
- Lets you turn off the GIL

More control: Cython

Generated by Cython 0.25.2

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

Raw output: calcpipy.c

```
+01: import random
 02:
+03: def calcpi_py(samples):
 04:     """serially calculate Pi using only standard library functions"""
+05:     inside = 0
+06:     random.seed(0)
+07:     for i in range(int(samples)):
+08:         x = random.random()
+09:         y = random.random()
+10:         if (x*x)+(y*y) < 1:
+11:             inside += 1
+12:     return (4.0 * inside)/samples
```

Generated by Cython 0.25.2

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

Raw output: calcpi.c

```
+01: cdef extern from "stdlib.h":
 02:     cpdef long random() nogil
 03:     cpdef void srandom(unsigned int) nogil
 04:     cpdef const long RAND_MAX
 05:
+06: cdef double randdbl() nogil:
 07:       cdef double r
+08:       r = random()
+09:       r = r/RAND_MAX
+10:       return r
 11:
+12: cpdef double calcpi(const int samples):
 13:     """serially calculate Pi using Cython library functions"""
 14:     cdef int inside, i
 15:     cdef double x, y
 16:
+17:     inside = 0
 18:
+19:     srandom(0)
+20:     for i in range(samples):
+21:         x = randdbl()
+22:         y = randdbl()
+23:         if (x*x)+(y*y) < 1:
+24:             inside += 1
+25:     return (4.0 * inside)/samples
 26:
```

More control: f2py

```
$cat calcpi.f90

    subroutine calcpi(samples, pi)
      REAL,INTENT(OUT) :: pi
      INTEGER, INTENT(IN) :: samples
      REAL :: x, y
      INTEGER :: i, inside

      inside = 0

      do i = 1, samples

        call random_number(x)
        call random_number(y)

        if ( x**2 + y**2 <= 1.0D+00 ) then
          inside = inside + 1
        end if

       end do
       pi = 4.0 * REAL(inside) / REAL(samples)
    end subroutine

$f2py --fcompiler=gfortran -m calcpi_fortran -c calcpi.f90
$...
$python -c "import calcpi_fortran; print calcpi_fortran.calcpi(1000000)"
3.14163589478
```

Basic Profiling: cProfile & SnakeViz

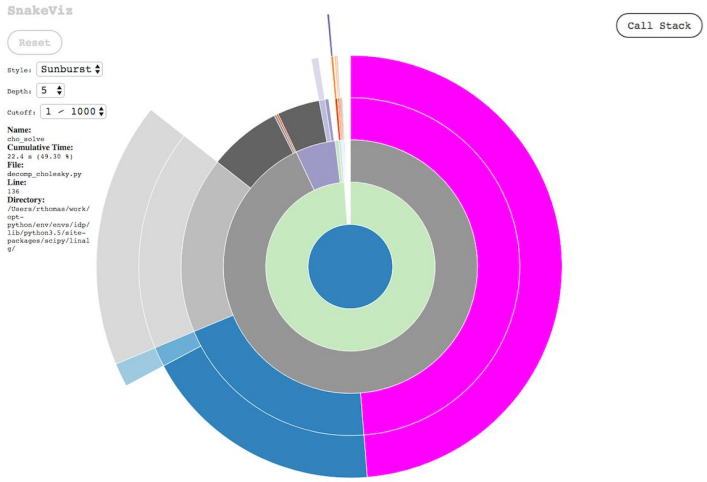
cProfile

Low-overhead profiler, from standard library.

Outputs statistics on what your code is doing:

- Number of function calls,
- Total time spend in functions,
- Time per function call, etc.

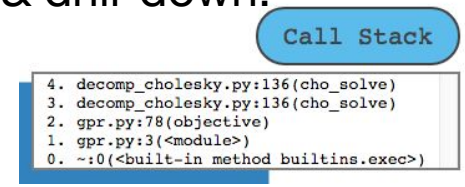
[https://docs.python.org/2/library/debug.html]
[https://docs.python.org/2/library/profile.html#module-cProfile]
[https://docs.python.org/2/library/profile.html#the-stats-class]



SnakeViz

Lets you visualize cProfile output in a browser:

- Statistics mentioned above.
- Visualize call stack & drill-down.




```
> python -m cProfile -o out.prof my-program.py
...
> snakeviz out.prof
snakeviz web server started on 127.0.0.1:8080…
```

[https://jiffyclub.github.io/snakeviz/]

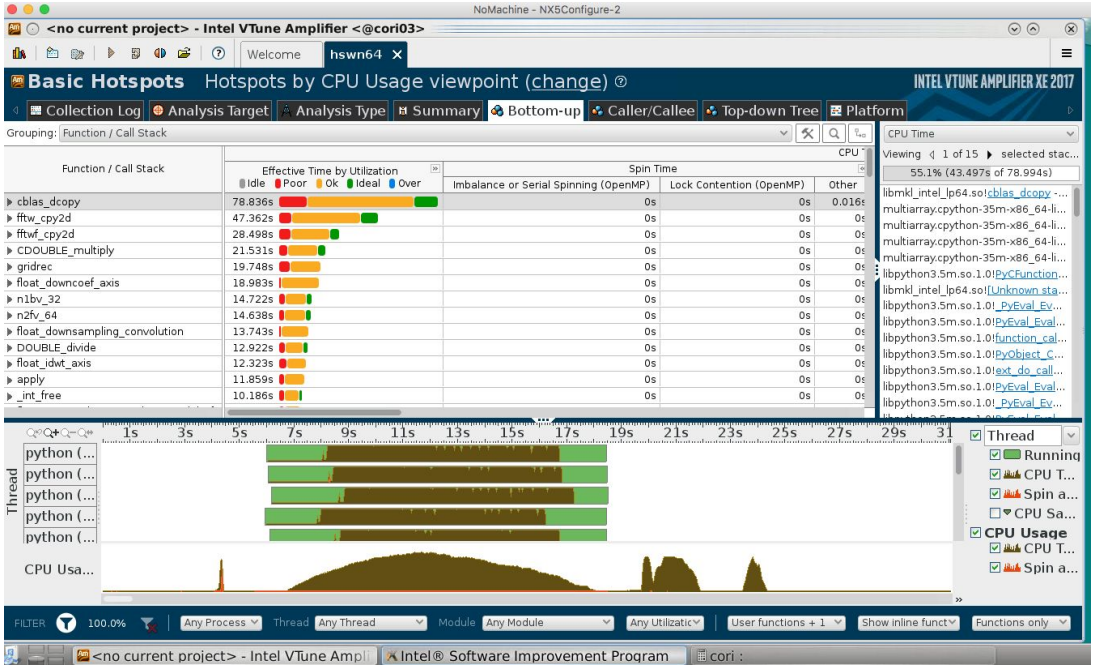
Intel VTune Works with Python Code

VTune Amplifier

Performance analysis profiler.
GUI and command-line interface.

Thread timelines.
Hotspot analysis.
Memory profiling.
Locks & waits.
Filter/zoom in timeline.

Run GUI (**`amplxe-gui`**) over NX!



[https://software.intel.com/en-us/node/628111]
[https://software.intel.com/en-us/videos/performance-analysis-of-python-applications-with-intel-vtune-amplifier]

Part of Intel Parallel Studio, may be available as a module, e.g. at NERSC:

```
> module load vtune
> salloc … --perf=vtune
…
> srun … amplxe-cl -collect hotspots python my-app.py
```

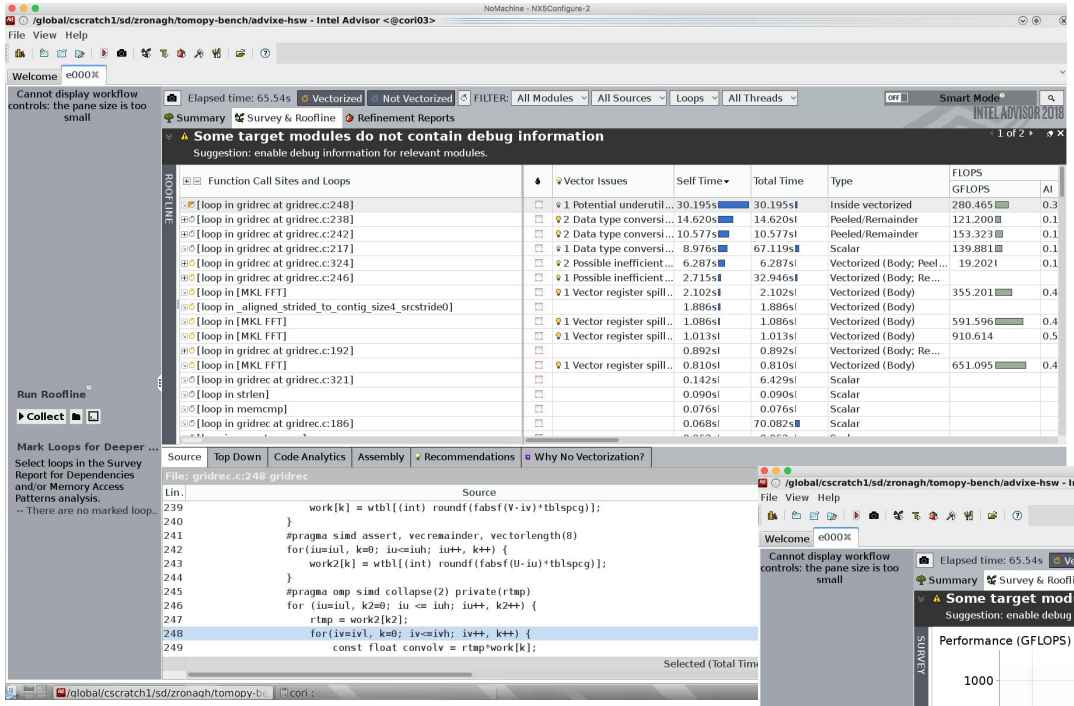
Best practice on KNL:

`-no-auto-finalize`, archive and **`-finalize`** on e.g. Haswell node

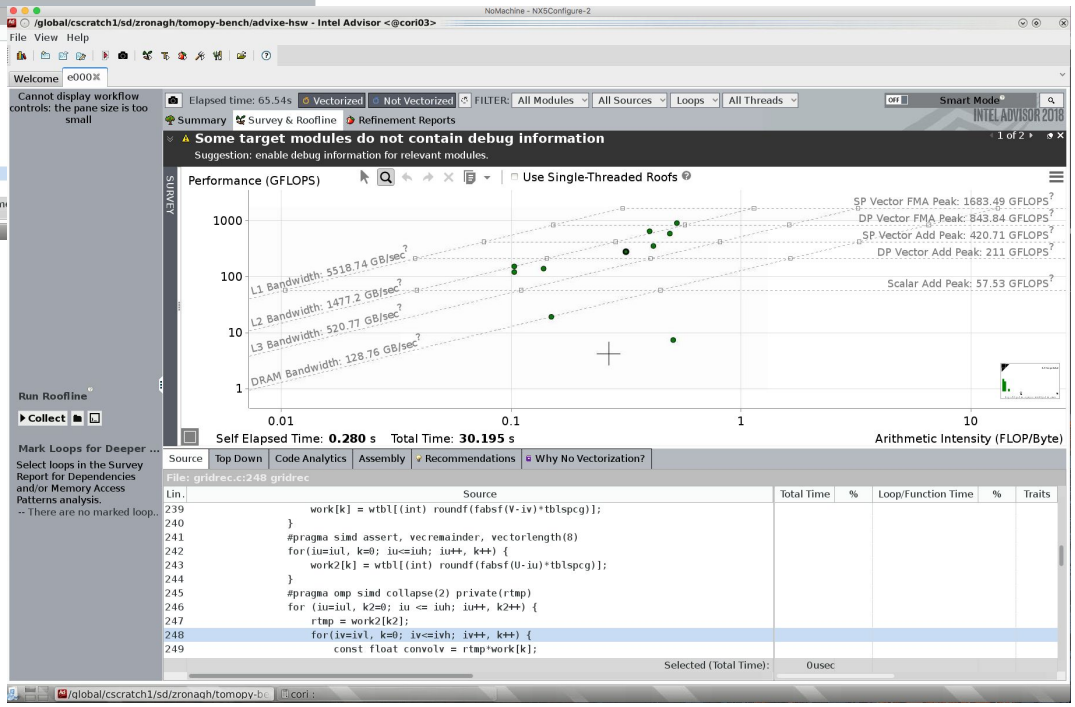
[e.g. http://www.nersc.gov/assets/Uploads/04-vtune.pdf]

Intel Tools Screenshots of TomoPy analysis courtesy Zahra Ronaghi, NERSC

Intel Advisor Works with Python Code



What should I do next? When do I stop?
Suggests optimizations for your C extensions.
Point out vectorization opportunities.
Optimize use of threads.
Works with Python and C/C++/Fortran code.



Roofline analysis*:
Performance of code in relation to hardware limits.
Memory bandwidth or compute bound?

[* Roofline: An upcoming IDEAS Webinar topic.]

Scaling Up Python

MPI(4py) ● Caveats ● Parallel I/O

mpi4py: why MPI?

- It is **(still)** the HPC paradigm for inter-process communications
- MPI makes full use of HPC environments
- Well-supported tools exist for parallel development with MPI – even when using Python
- Python-MPI bindings have been developed since 1996

mpi4py: why mpi4py?

- Pythonic wrapping of the system's native MPI
- provides almost all MPI-1,2 and common MPI-3 features
- very well maintained
- distributed with major Python distributions
- portable and scalable
  - requires only: NumPy, Cython (build only), and an MPI
  - used to run a Python application on 786,432 cores
  - capabilities only limited by the system MPI
- http://mpi4py.readthedocs.io/en/stable/

mpi4py: running

- mpi4py jobs are launched like other MPI binaries:
  - `mpiexec -np ${RANKS} python ${PATH_TO_SCRIPT}`
  - Just running `python ${PATH_TO_SCRIPT}` should always work for a single-rank case
- an independent Python interpreter launches per rank
  - no automatic shared memory, files, or state
  - crashing an interpreter does crash the MPI program
  - it is possible to embed an interpreter in a C/C++ program and launch an interpreter that way
- if you crash or have trouble with simple codes, remember:
  - CPython is a C binary and mpi4py is a binding
  - you will likely get core files and mangled stack traces
  - use ld or otool to check which MPI mpi4py is linked against
  - ensure Python, mpi4py, and your code are available on all nodes and libraries and paths are correct
  - try running with a single rank
  - rebuild with debugging symbols

mpi4py: startup

- Importing and MPI initialization:
- importing mpi4py allows you to set runtime configuration options (e.g. automatic initialization, thread_level) via `mpi4py.rc()`
- importing the MPI submodule calls MPI_Init() by default
  - calling `Init()` or `Init_thread()` more than once violates the MPI standard
  - This will lead to a Python exception or an abort in C/C++
  - use `Is_initialized()` to test for initialization

mpi4py: shutdown

- `MPI_Finalize()` will automatically run at interpreter exit
- use `Is_finalized()` to test for finalization when uncertain if a module called `MPI_Finalize()`
- calling `Finalize()` more than once exits the interpreter with an error and may crash C/C++/Fortran modules

mpi4py and program structure

Any code, even if after `MPI_Init()`, unless reserved to a given rank will run on all ranks:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
mpisize = comm.Get_size()

if rank%2 == 0:
  print("Hello from an even rank: %d" %(rank))

comm.Barrier()
print("Goodbye from rank %d" %(rank))
```

mpi4py: datatypes

- Buffers, MPI datatypes, and NumPy objects aren't pickled
  - Transmitted near the speed of C/C++
  - NumPy datatypes are autoconverted to MPI datatypes
  - buffers may need to be described as a 2/3-list/tuple
    - `[data, MPI.DOUBLE]` for a single double
    - `[data,count,MPI.INT]` for an array of integers
  - Custom MPI datatypes are supported
  - Use the **capitalized methods**, eg: `Recv()`, `Send()`
- All other objects require pickleing
  - pickling and unpickling have significant overheads
  - Use the **lowercase methods**, eg: `recv()`,`send()`
- When in doubt, ask if what is being processed can be represented as memory buffer or only as PyObject

mpi4py: communicators

- The two default communicators exist at startup:
  - `COMM_WORLD`
  - `COMM_SELF`
- For safety, duplicate communicators before use in or with libraries and modules you didn't write
- Only break from the standard are methods:

  `Is_inter()` and `Is_intra()`

mpi4py: collectives and operations

- Collectives operating on Python objects are naive for example:

```
mpirun -n 8 $(which python) ./basic_features.py
###########################################################################
Rank 0 sees local_dict as {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4, 'g': 'gee whiz', 'f': 6, 'h': ('hi', 'there')}
Rank 0 sees local_list_max as [0, 1, 2, 3]
Rank 0 sees local_list_sum as [0, 1, 2, 3]
Rank 0 sees local_string as This is a string.
Rank 0 sees local_tuple as (0, 0, 0, 0, 0, 0, 0, 0)
Rank 0 sees local_np_array as [0 1 2 3 4 5 6 7 8 9]
###########################################################################
Rank 6 sees local_dict as None
Rank 6 sees local_list_max as [0, 6, 12, 18]
Rank 6 sees local_list_sum as [0, 6, 12, 18]
Rank 6 sees local_string as This should be fun!
Rank 6 sees local_tuple as (6, 6, 6, 6, 6, 6, 6, 6)
Rank 6 sees local_np_array as [0 1 2 3 4 5 6 7 8 9]
###########################################################################

Running collective operations

###########################################################################
Rank 0 sees local_dict as a after scatter using None
Rank 0 sees local_list_max as [0, 7, 14, 21] after allreduce using max
Rank 0 sees local_list_sum as [0, 1, 2, 3, 0, 1, 2, 3, 0, 2, 4, 6, 0, 3, 6, 9, 0, 4, 8, 12, 0, 5, 10, 15, 0, 6, 12, 18, 0, 7, 14, 21] after reduce using sum
Rank 0 sees local_string as This is a string. after bcast using defaults
Rank 0 sees local_tuple as [0, 1, 2, 3, 4, 5, 6, 7] after alltoall using defaults
Rank 0 sees local_np_array as [ 0  8 16 24 32 40 48 56 64 72] after allreduce using sum
###########################################################################
Rank 6 sees local_dict as f after scatter using None
Rank 6 sees local_list_max as [0, 7, 14, 21] after allreduce using max
Rank 6 sees local_list_sum as None after reduce using sum
Rank 6 sees local_string as This is a string. after bcast using defaults
Rank 6 sees local_tuple as [0, 1, 2, 3, 4, 5, 6, 7] after alltoall using defaults
Rank 6 sees local_np_array as [ 0  8 16 24 32 40 48 56 64 72] after allreduce using sum
```

- Collective reduction operations on Python objects are mostly serial
- Casing convention applies to methods:
  - lowercased methods will work for general Python objects (albeit slowly)
  - uppercase methods will work for NumPy/MPI data types at near C speed

Parallel I/O and h5py

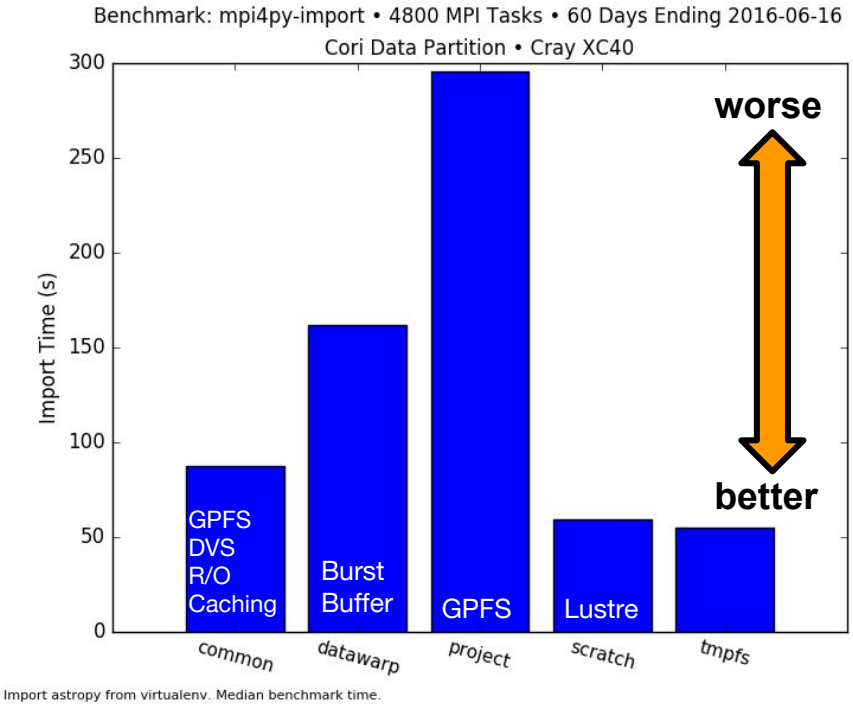
- General Python I/O isn't MPI-safe
- Beware pre-packaged h5py
- (
```
>>> import h5py
>>> h5py.get_config().mpi
True
```
re using:

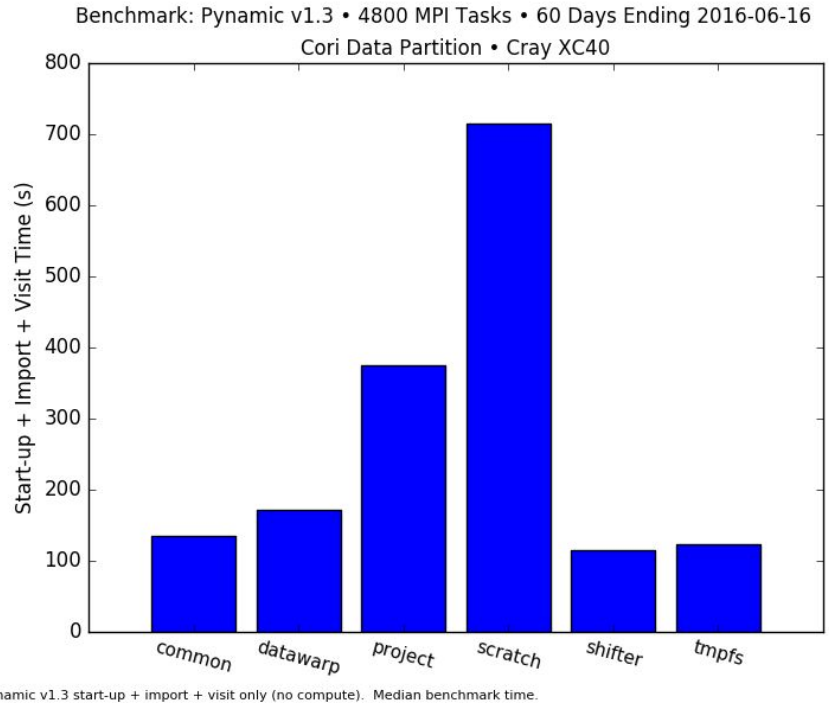
- Requires mpi4py and the mpicc used to compile hdf5, mpi4py, and h5py must match
- As easy to use as:

```
f = h5py.File('myfile.hdf5', 'w',
        driver='mpio', comm=MPI.COMM_WORLD)
```

- All changes to file structure or metadata of a file must be performed on all ranks with an open file

Issues Affecting Python at Scale





- Python's "import" statement is file metadata intensive (.py, .pyc, .so open/stat calls).
- Becomes more severe as the number of Python processes trying to access files increases.
- Result: Very slow times to just start Python applications at larger concurrency (MPI).
- Storage local to compute nodes, use of containers (**Shifter**) helps fix:
  - Eliminates metadata calls off the compute nodes.
  - In containers, paths to .so libraries can be cached via ldconfig.
- Other approaches:
  - Ship software stack to compute nodes (e.g., python-mpi-bcast).
  - Install software to read-only/cache-enabled file systems.
  - See also Spindle (Scalable Shared Library Loading).

Conclusion Next (Questions?)

Conclusion

- NERSC, ALCF, and OLCF recognize, welcome, and want to support new and experienced Python users in HPC.
- Using Python on our systems can be as easy as a `module load`, but can be customized by users.
- We have provided some guidance and best practices to help users improve Python performance in HPC context.
- Try out some of the profiling and performance analysis tools described here, and ask for help if you get stuck.
- While there are many challenges for Python in HPC, if users, staff, & vendors work together, there are many rewards.

More Resources

Your NERSC and LCF Python contacts:

NERSC: Rollin Thomas rcthomas@lbl.gov

ALCF: William Scullin wscullin@alcf.anl.gov

OLCF: Matt Belhorn belhornmp@ornl.gov

Documentation:

NERSC: http://www.nersc.gov/users/data-analytics/data-analytics/python/

OLCF: https://www.olcf.ornl.gov/training-event/2016-olcf-user-conference-call-olcf-python-best-practices/

Other presentations:

ALCF Performance Workshop (May 2017):

Python on HPC Best Practices http://www.alcf.anl.gov/files/scullin-python.pdf

NERSC Intel Python Training Event (March 2017):

Optimization Example http://www.nersc.gov/assets/Uploads/Intel-tomopy-Mar2017.pdf
by Oleksandr Pavlyk (Intel)

Backup Material

Cross-Compiling on Crays with Pip

```
# Instruct Cray compiler wrappers to target the login node architecture so code will run
everywhere
module unload craype-interlagos
module load craype-istanbul

virualenv --python=python2.7 "${VENV_NAME}"
source "${VENV_NAME}/bin/activate"

# If pip is badly out of date, the TLS certificates may not be trusted.
pip install --trusted-host pypi.python.org --upgrade pip

# Set envvars needed to guide pip for cross-compiling and instruct it to build from source
CC=cc MPICC=cc pip install -v --no-binary :all: mpi4py

# Set envvars needed for pip to use external dependencies. See package documentation.
HDF5_DIR="${CRAY_HDF5_DIR}/${PE_ENV}/${GNU_VERSION%.*}"
CC=cc HDF5_MPI="ON" HDF5_DIR="${HDF5_DIR}" pip install -v --no-binary :all: h5py
deactivate "${VENV_NAME}"
module unload craype-istanbul
module load craype-interlagos
```