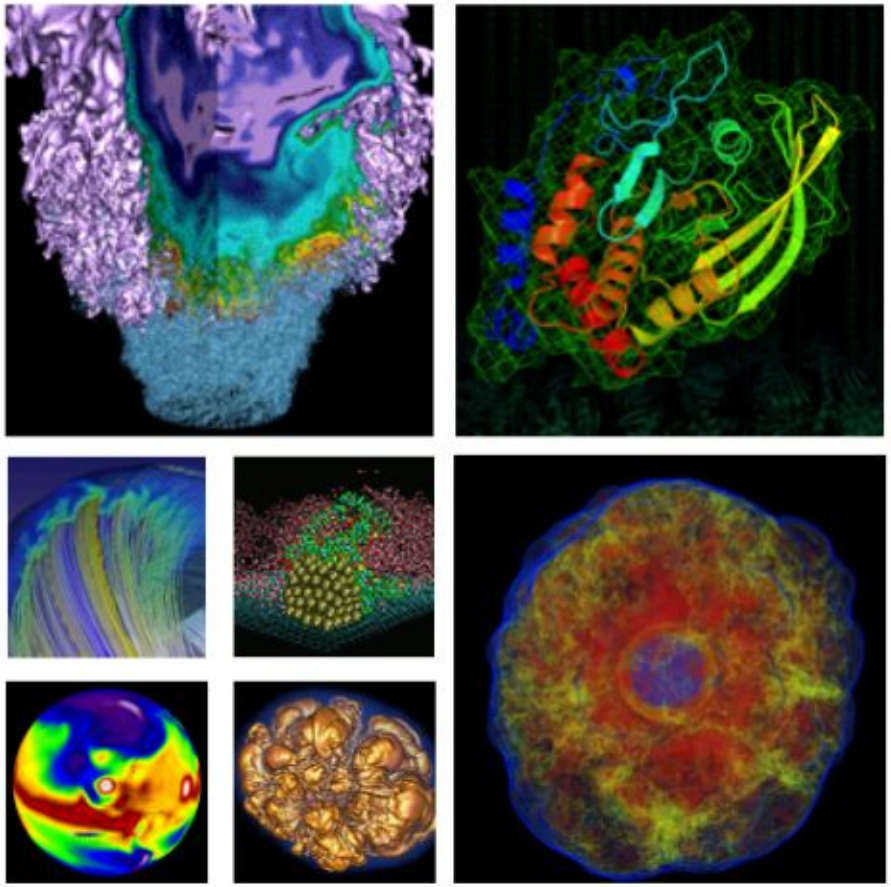
An Introduction to Python at NERSC

NERSC New User Training



Rollin Thomas

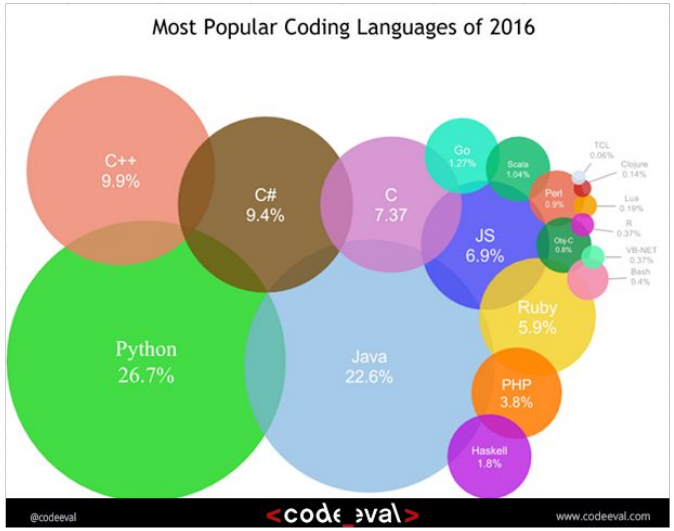
Data & Analytics Services Group

2017-02-24

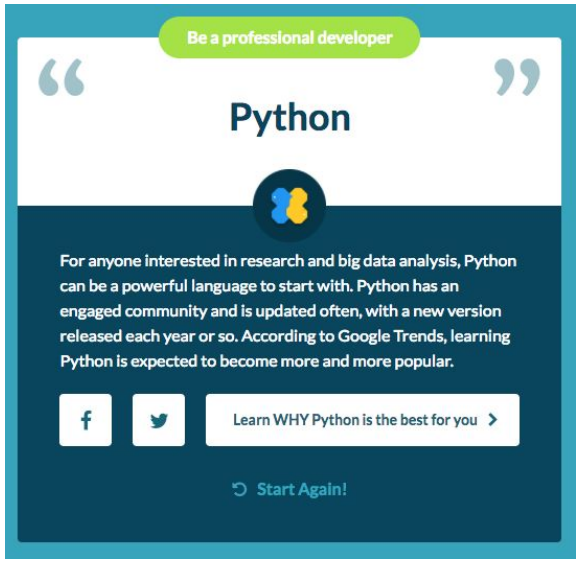
Python is Popular

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



codeval.com



bestprogramminglanguagefor.me

Why Python?

Clean, clear syntax makes it very easy to learn.

Multi-paradigm interpreted language.

Extremely popular language for teaching beginners...

… but stays useful beyond the beginner phase of programming:

Powerful data structures and constructs built into the language and standard libraries. Leveraging of C/C++/Fortran.

Huge collection of useful open source packages to re-use and extend.

```
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
```

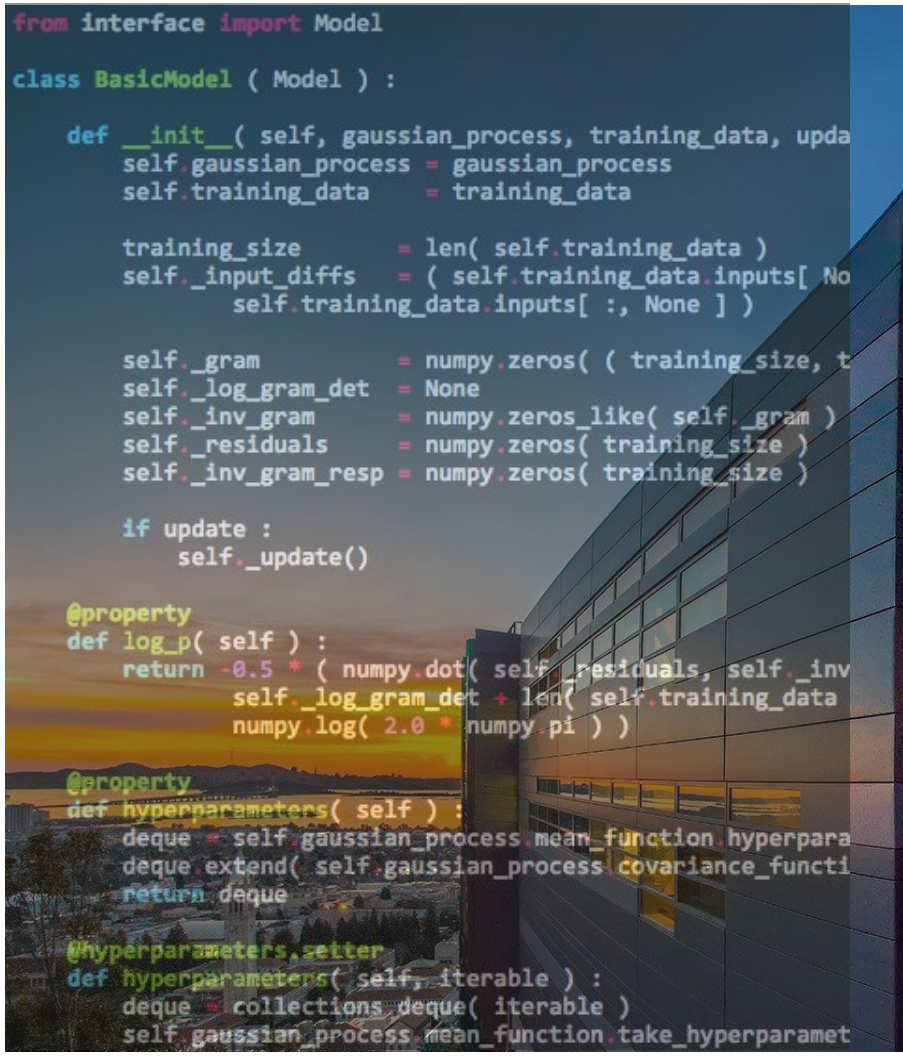
Python at NERSC

Supporting Python is no longer optional at HPC centers like NERSC.

Maximizing Python performance on systems like Cori and Edison can be **challenging**:

- Interpreted, dynamic languages are harder to optimize.
- Python's global interpreter lock is an issue for thread-level parallelism.
- Language design and implementation choices made without considering an HPC environment.

At the same time, users want NERSC to provide a familiar and portable Python environment.



Python Modules at NERSC

Environment modules:

Environment modules project:
http://modules.sourceforge.net/

Always* "module load python"

Don't use /usr/bin/python.
Using #!/usr/bin/env python: OK!

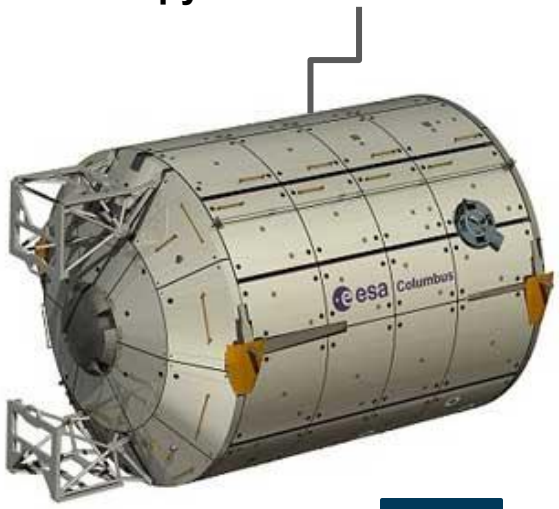
What is there?

```
module avail python
```

* Unless you install your own Python somehow.
Which is totally fine, see later in the talk.

python/2.7.9

python/2.7-anaconda



Python Installations at NERSC

“NERSC-Built” Python

- Python “base” module.
- Add-on modules as desired.
- Meta-module simplifies setup.

Anaconda Python

- “Distribution” for large-scale data analytics, and scientific computing.
- ~200 packages but there is also “miniconda” bare-bones starter.
- Simplified package management and deployment (conda tool).
- Monolithic module, some add-on modules (h5py-parallel).

https://docs.continuum.io/anaconda/

Python Modules on Edison

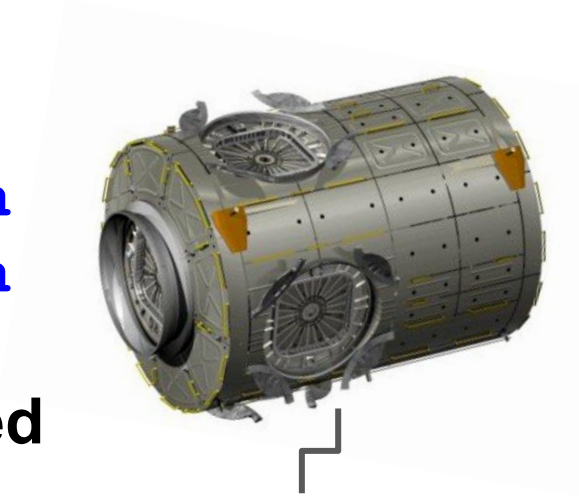
NERSC-built:

```
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
```



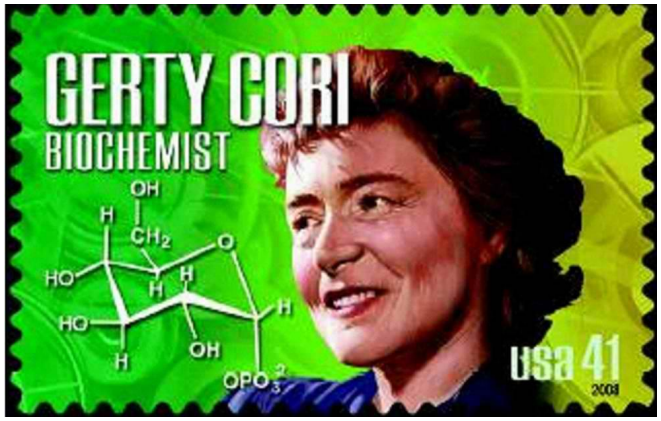
python/2.7.9

Above are the only currently recommended Python modules for Edison.

Python Modules on Cori

NERSC-built:

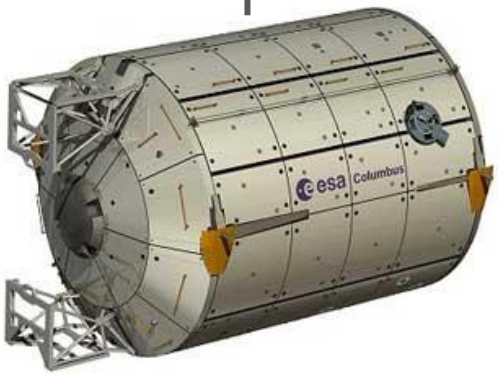
There aren't any.



Anaconda:

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

python/2.7-anaconda



Above are the only currently recommended Python modules for Cori.

Do-It-Yourself Python at NERSC

Anaconda Environment under Modules:

```
module load python/2.7-anaconda
conda create -p $PREFIX numpy…
conda create -n myenv numpy…
```

(won't work for users without .condarc defining "envs_dirs")

```
conda install basemap yt…
```

Your own Anaconda or Miniconda installation:

```
module unload python
wget https://repo.continuum.io/miniconda/Miniconda2-latest-Linux-x86_64.sh
/bin/bash Miniconda2-latest-Linux-x86_64.sh -b -p $PREFIX
source $PREFIX/bin/activate
    <or export PATH=$PREFIX/bin:$PATH>
conda install basemap yt…
```

Tips:

- Conda environments do ***not*** mix with virtualenv.
- Several ML environments via Anaconda at NERSC.

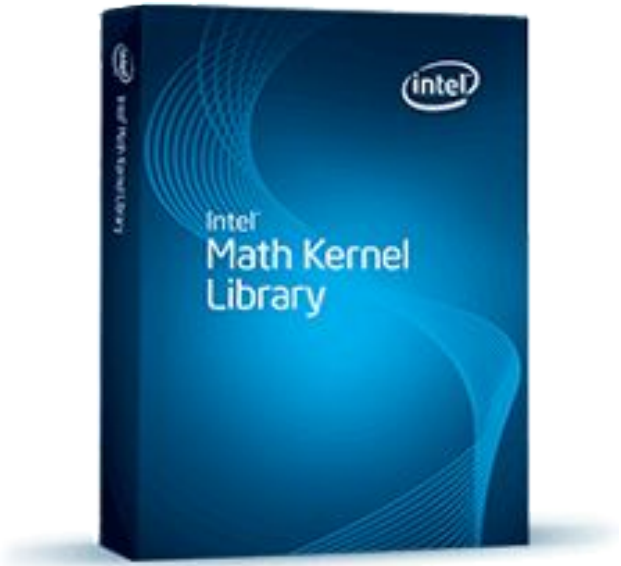
Node Parallelism: Threaded Libraries

Anaconda Python provides access to Intel Math Kernel Library (MKL) *for free:*

- numpy
- scipy
- scikit-learn
- numexpr

MKL Service functions*:

```
>>> import mkl
>>> mkl.get_max_threads()
2
>>> mkl.set_num_threads(1)
>>> mkl.get_max_threads()
1
```



*https://github.com/ContinuumIO/mkl-service

Intel Distribution for Python

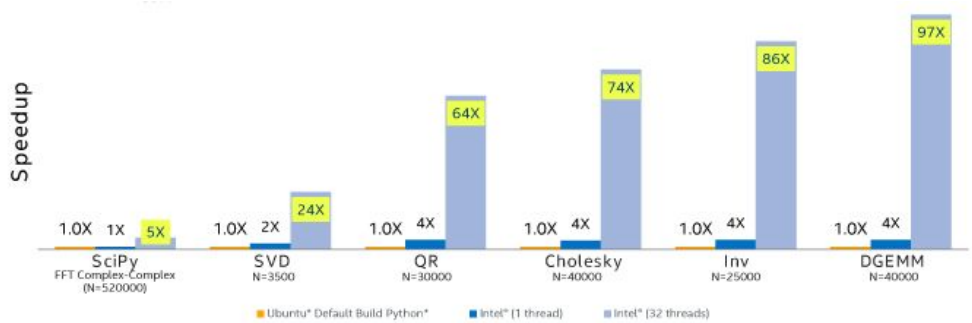
Available through Anaconda as well:

```
conda create -p $SCRATCH/idp \
    -c intel intelpython2_core python=2
source activate $SCRATCH/idp
```

Features:

Leveraging Intel MKL, MPI, TBB, DAAL.

Intel-specific enhancements (FFT, threaded RNG, etc).



Configuration info: - Versions: Intel* Distribution for Python 2.7.11 2017, Beta (Mar 08, 2016), Ubuntu* built Python*: Python 2.7.11, NumPy 1.10.4, SciPy 0.17.0 built with gcc 4.8.4; Hardware: Intel* Xeon* CPU E5-2698 v3 @ 2.30GHz (2 sockets, 16 cores each, HT=OFF), 64 GB of RAM, 8 DIMMS of 8GB@2133MHz; Operating System: Ubuntu 14.04 LTS; MKL version 11.3.2 for Intel Distribution for Python 2017, Beta

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804 .

Multi-Node Parallelism: mpi4py

MPI support via mpi4py (2.0.0)

2.0.0 added last year.

Includes MPI-3 features.

Compiled against Cray libraries.

Built into Anaconda modules on Edison and Cori.

Non-Anaconda route:

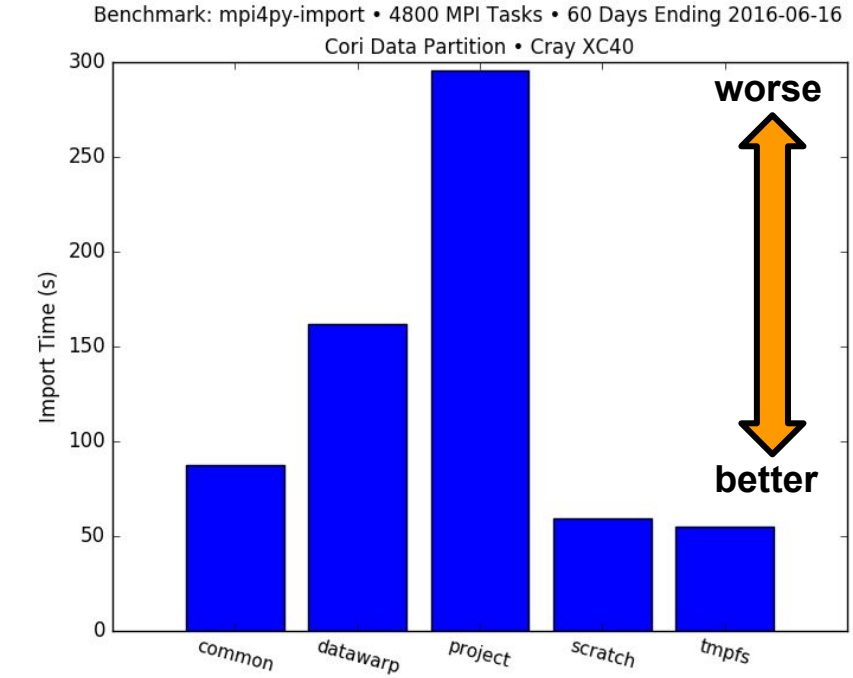
```
module load mpi4py
```

DIY mpi4py builders:

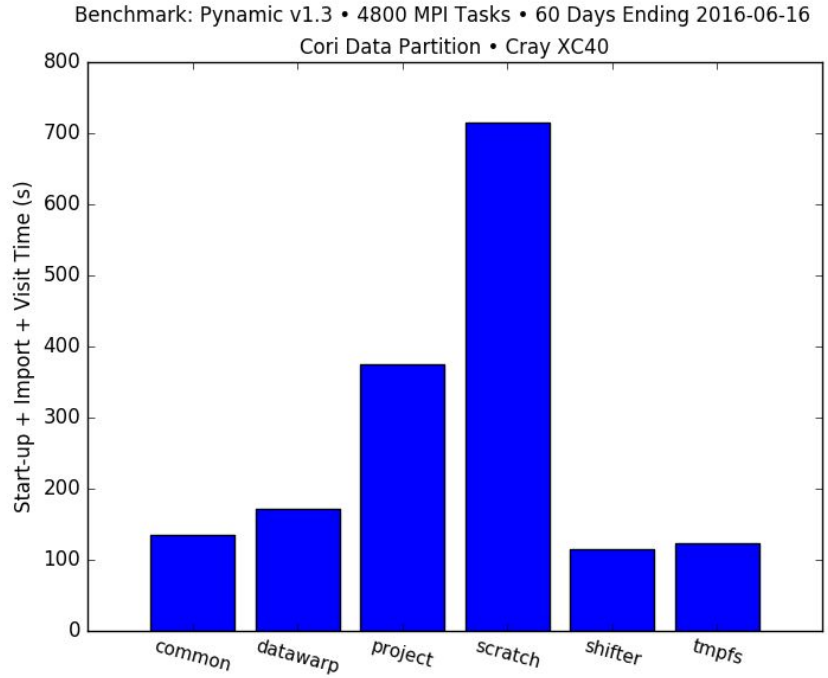
See NERSC website.

```
from mpi4py import MPI

# Initialize MPI.

comm = MPI.COMM_WORLD
mpi_rank = comm.Get_rank()
mpi_size = comm.Get_size()

# Take command line arguments.

seed = int(sys.argv[1])
size = int(sys.argv[2])

# Have root (rank 0) task confirm MPI size.

if mpi_rank == 0 :
    start = time.time()
    print "MPI size", mpi_size
    print

# Different random number per rank.

numpy.random.seed(seed + mpi_rank * 1000)

# Count points in Q1 of unit circle.

x = numpy.random.uniform(size=size)
y = numpy.random.uniform(size=size)
r2 = x * x + y * y
in_circle = numpy.sum(r2 < 1.0, dtype=float)

# Reduce count to root MPI task (rank 0) by summing them all.
# MPI wants things in numpy arrays.

in_circle = numpy.array([in_circle])
total_in_circle = numpy.zeros(1, dtype=float)
comm.Reduce(in_circle, total_in_circle, op=MPI.SUM)

# Reduce total number of points tried in the same fashion.
# Note that this step is actually unnecessary.

in_square = numpy.array([size], dtype=float)
total_in_square = numpy.zeros(1, dtype=float)
comm.Reduce(in_square, total_in_square, op=MPI.SUM)
```

MPI Start-up in Python Apps at Scale



Import astropy from virtualenv. Median benchmark time.



Pynamic v1.3 start-up + import + visit only (no compute). Median benchmark time.

- 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).
- **BEST POSSIBLE PERFORMANCE IS *SHIFTER***:
  - Eliminates metadata calls off the compute nodes.
  - Paths to .so libraries can be cached via ldconfig.
- Other approaches:
  - Pack up software to compute nodes (python-mpi-bcast).
  - Install software to $SCRATCH or /global/common.

Multiprocessing and Process Spawning

You can use multiprocessing for on-node throughput jobs.

```
from multiprocessing import Pool

def f(x):
    return x*x

if __name__ == '__main__':
    p = Pool(5)
    print(p.map(f, [1, 2, 3]))
```

Combining multiprocessing with mpi4py, unreliable results.

Combining it with threaded MKL/OpenMP on especially on KNL is problematic.

Combining mpi4py and subprocess?

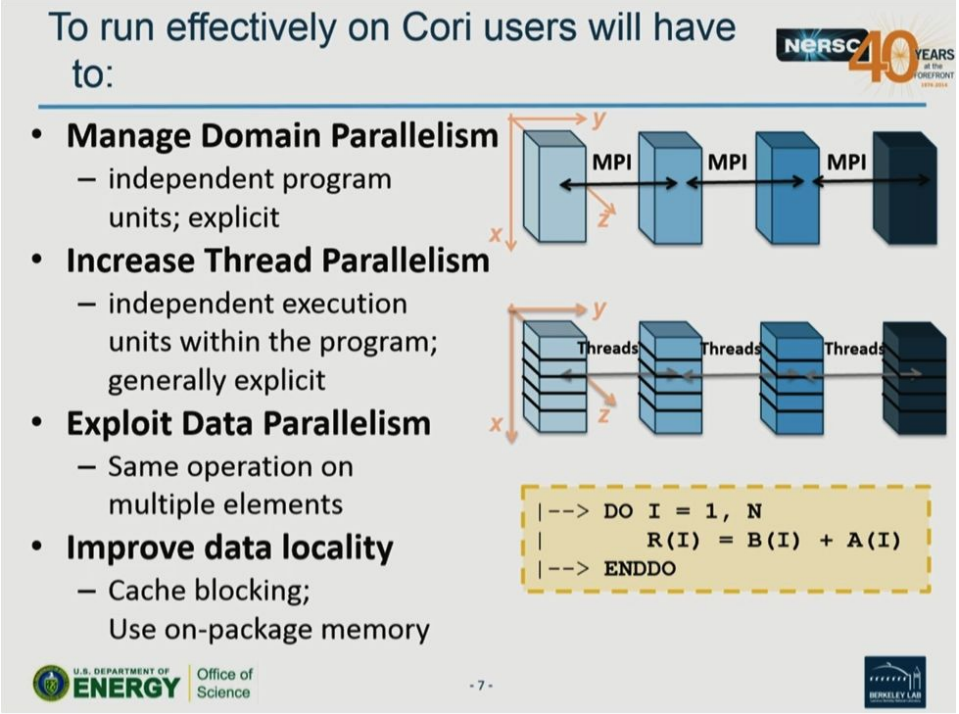
Works to spawn serial, compiled executables.
Just don't compile those with Cray wrappers cc, CC, ftn.
Do `module load gcc` and use gcc, g++, gfortran.

Python on Cori Phase II

Knights Landing (KNL)

2x cores per node
Slower clock rate
Less memory/core.

Single-thread or flat MPI Python won't be great.



Advice:

Leverage threaded, vectorized math/specialized libraries.
Consider writing Cython/C extensions you can vectorize?
Learn about Intel Python and Intel profiling tools.
➡ Training event at NERSC on **Intel Python, March 10!**

Jupyter at NERSC and on Cori

Jupyter Notebook: "Literate Computing."

Code, text, equations, viz in a narrative.

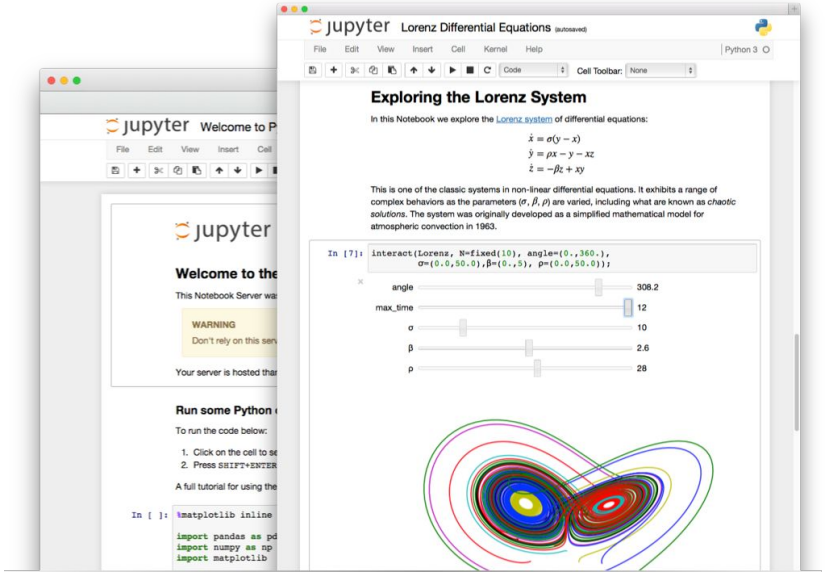
New way to interact with NERSC HPC resources:

Old: Use ssh or NX to get to command line.

New: Open a notebook, create a narrative.

Move to Cori:

- Access to $SCRATCH.
- Interaction with SLURM.
- Eventually Burst Buffer.
- New ways of using Cori.
  - DASK, PySpark, IJulia...



Live Demo

SLURM Magic Commands

jupyter awesome Last Checkpoint: 03/25/2016 (autosaved)

Control Panel | Logout

File Edit View Insert Cell Kernel Help | Notebook saved | Python 2.7

Code | CellToolbar

In [1]:
```
%squeue -u rthomas
```

Out[1]:

	JOBID	USER	ACCOUNT	NAME	PARTITION	QOS	NODES	TIME_LIMIT	TIME	ST	PRIORITY	SUBMIT_TIME	START_TIME
0	2875563	rthomas	mpccc	try.sh	regular	normal_reg	150	1:00	0:00	PD	64832	2016-08-19T15:33:09	NaN

In [2]:
```
%sprio -j 2875563
```

Out[2]:

	JOBID	PRIORITY	AGE	FAIRSHARE	QOS
0	2875563	64832	32	0	64800

In [3]:
```
%sacct -u rthomas
```

Out[3]:

	JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
0	----------	----------	----------	----------	----------	----------	--------
1	2873623	try.sh	debug	mpccc	64	FAILED	1:0
2	2873623.bat+	batch	mpccc	64	FAILED	1:0	NaN
3	2873623.ext+	extern	mpccc	64	COMPLETED	0:0	NaN
4	2873623.0	shifter	mpccc	32	FAILED	1:0	NaN
5	2873630	try.sh	debug	mpccc	64	FAILED	1:0
6	2873630.bat+	batch	mpccc	64	FAILED	1:0	NaN

Conclusion

Python is an integral element of NERSC's Data Intensive Science portfolio.

We want users to have a:

familiar	Python environment
productive	Python experience
performant	Python software stack

Pursuing new ways to empower Python & data users.

Always looking for feedback, advice, and even help:

consult@nersc.gov or https://help.nersc.gov/

rcthomas@lbl.gov

National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY

BERKELEY LAB
Lawrence Berkeley National Laboratory