Outline

1. Python enables HPC science at NERSC
   Orchestration • Workflows • Analytics • HPC Apps

2. How we help Python users at NERSC
   Productivity • Performance

3. Experimental/Observational Science Engagements
   Python in NESAP for Data Projects w/Intel
What is NERSC?

The production user facility for high performance computing and data for the Department of Energy’s Office of Science.
<table>
<thead>
<tr>
<th>Name</th>
<th>System Type</th>
<th>Processor Type</th>
<th>Speed/Cores per Node</th>
<th>Peak Performance</th>
<th># Nodes</th>
<th>Aggregate Memory</th>
<th>Memory per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edison</td>
<td>Cray XC30</td>
<td>Ivy Bridge</td>
<td>2.4/24</td>
<td>2.57 PF/s</td>
<td>5586</td>
<td>357 TB</td>
<td>64 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cori</td>
<td>Cray XC40</td>
<td>Haswell</td>
<td>2.3/32</td>
<td>1.92 PF/s</td>
<td>2388</td>
<td>305 TB</td>
<td>128 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KNL</td>
<td>1.4/68</td>
<td>28 PF/s</td>
<td>9688</td>
<td>1.1 PB</td>
<td>96+16 GB</td>
</tr>
</tbody>
</table>
7K users, 800 projects, 2K papers

~10 billion “NERSC hours” provided to users in 2017.

Advanced Scientific Computing Research

Biological & Environmental Research

Basic Energy Sciences

Fusion Energy Sciences

High Energy Physics

Nuclear Physics

Small Business Innovation Research
Science via Python@NERSC

The Materials Project
Powering Workflows to Understand Properties of Materials

Nbodykit
Modeling Dark Matter and Dark Energy

LHC ATLAS Data Processing Workflow

Sky Survey Catalogs for Cosmology

ML/DL
Python in HPC Jobs at NERSC

Around 3% of NERSC hours on Cori in the past year easily detected as Python jobs*:

```
  srun -n ... python whatever.py ... 
```

This is a lower limit, as users:

- Often make main programs executable
- Use Python in containers to scale up

* Production batch jobs, not use on shared login nodes.
2017 NERSC User Survey
656 total respondents
N=336 reporting use (51%)
Monitored Imports (Cori)

MODS* Statistics
Recent 30 day period
Compute nodes only
NERSC’s modules only

* MODS = Monitoring of Data Services at NERSC = BI Project in DAS
Python in Edge Services

Data Sharing Across Facilities

Interactive Tools

Rich Visualizations and UIs

enables science through . . . Interfaces to HPC resources & workflows

The Legacy Surveys

The Legacy Surveys are producing an inference model catalog of the sky from a set of optical and infrared imaging data, comprising 14,000 deg² of extragalactic sky visible from the northern hemisphere in three optical bands (g, r, i) and four infrared bands. The sky coverage is approximately bounded by −18° < b < +4° in ecliptic coordinates and (b) > 18° in Galactic coordinates. To address this goal, the Legacy Surveys are conducting imaging projects on different telescopes, described in more detail at the following links:

The Baryon Acoustic Oscillations Legacy Survey (BAO) | The Diffuse Legacy Survey (D3LS) | The Megapixel e-Science Legacy Survey (MELiS)
Interactive Supercomputing

Web Browser

JupyterHub
Web Server

--qos=interactive

Cori Login Node

Notebook Server Process

Kernel Process

Cori Compute Node

Notebook Server Process

Kernel Process

Cori Compute Node

Notebook Server Process

Kernel Process

Cori Compute Node

Notebook Server Process

Kernel Process
NERSC’s Python Strategy

Focus on user productivity.
Support familiar, trusted, up-to-date libraries.
Find ways to put performance in user reach.

Examples:

Threaded libraries: Intel MKL
Support cluster scaling: Cray+mpi4py
Close architecture gaps: Containers
NERSC Python: Anaconda

Most well-known and widely used distribution. Designed around analytics, statistics, ML/DL. “Personalized” environments and package manager. Environments are rapid setup, shareable, & reusable.

2016: MKL added, and Intel upstreams optimizations: NERSC drops its builds of Python on Cray the same year.

Other options for HPC: Source builds, Spack, etc.
Handling MPI with mpi4py

Cluster parallelism with MPI via mpi4py:
- MPI-1/2/3 specification support
- OO interface ~ MPI-2 C++ bindings
- Point-to-point and collectives
- Picklable Python objects & buffers

Build mpi4py & dependents with Cray MPICH:

```bash
python setup.py build --mpicc=cc
python setup.py install
```
Containers and Python go well together at NERSC

Motivations, esp. for data science:
- Flexibility
- Convenience
- Consistency
- Reproducibility

Some Options:
- Docker
- Singularity
- Shifter (~Docker on Cray)
- CharlieCloud

Nice recent blog summary of the state of HPC containers:
https://www.stackhpc.com/the-state-of-hpc-containers.html
“Slow Launch” at Scale

Python’s import is metadata intensive, ⇒ catastrophic contention at scale ⇒ it matters where you put your env

Project (GPFS):
For sharing large data files

Scratch (Lustre):
OK, but gets purged periodically!

Common (GPFS):
RO w/Cray DVS client-side caching
Open to users now, was only staff

Shifter (Docker Containers):
Metadata lookup only on compute
Storage on compute is RAM disk
Idconfig when you build image

Median launch time incl. MPI_Init()
General advice from users regarding KNL:

- Your code will run without modification
- Expect some refactoring if you want best performance
- Fine-grained parallelism to exploit 68 cores/node
- Make aggressive use of the 512-bit vector units
- Structure data to stay in KNL’s 16 GB of MCDRAM
Python on Knights Landing

Translation for Python users. At least,

- Understand and use numpy array syntax, broadcast rules, and scalar/"vector" interfaces to functions.
- Use threaded+vectorized libraries and compiled extensions, minimize time outside of using them.
- There may, in fact, be more than one way to do it; Prepare to rethink algorithms, memory usage, etc.
- Layer use of profiling tools to identify/assess hotspots.
**NESAP for Data**

**NERSC Exascale Science Applications Program for Data:**

*Users whose applications process, analyze, and/or simulate data sets or data streams from experiments and instrumentation supported by DOE need help preparing for extreme scale and exascale computing.*

- Early Engagement with Code Teams
- Close Interactions with Vendors
- Developer Workshops, “Dungeons”
- Expanded Access to KNL + Data Ecosystem
- Postdoc Fellowship Program
- Training Docs, Online Modules
- Leverage Community Efforts
Python NESAP for Data Projects

**TomoPy (Python & C):**
Tomographic data processing and image reconstruction  
*PI: Doga Gursoy, Argonne National Laboratory*

**DESI Pipeline (As Pure Python as Reasonably Possible):**
Baryon acoustic oscillations (DESI Project)  
*PI: Stephen Bailey, Lawrence Berkeley Laboratory*

**TOAST (Time Ordered Astrophysics Scalable Tools, Python & C++):**
Cosmic microwave background data analysis and simulation (CMB S4)  
*PI: Julian Borrill, Lawrence Berkeley Laboratory*
TomoPy (D. Gursoy, Z. Ronaghi; O. Pavlyk)

Framework for analysis of synchrotron tomography data

75% Python, ~25% C/C++

I/O handled via dxchange, HDF5

Modular design

Pre-processing
Image Reconstruction
Post-processing

Parallelism within node:
multiprocessing
concurrent.futures
TomoPy Optimizations So Far

Library dependencies:
- Intel Python: NumPy, SciPy, scikit-image
- Local builds: fftw, pyfftw, dxchange, dxfile, olefile

Compilation:
- Build TomoPy C extensions with icc
- Target common-avx512 architecture enabling vectorization

Runtime:
- To use multiprocessing on KNL needed to set KMP_AFFINITY=disabled
- *Using huge memory pages: gridrec algorithm 30% faster on Haswell, 45% faster on KNL*

Code changes, mostly in C layer gridrec algorithm, some preprocessing:
- Appropriate precision (*ceilf, *sinf, *cosf*), avoid upcast/downcast if we can
- Replace *lroundf(x)* with *(int)roundf(x)* to enable vectorization
- Apply icc-specific vectorization pragmas (Intel Compiler)
- 64-byte aligned memory allocation from X/Open-7 posix standard instead of fftw allocator
- Parallelize FFTs with “many FFT” for more slices at once
- *Employ cache blocking to lower miss rate for interpolation step (25% to 4% on KNL)*
- Pre-processing: Encapsulated uses of fft, ifft and fft2, ifft2 by replacing direct calls to pyfftw.interfaces.numpy functions with calls to wrapper functions
TomoPy Gridrec Optimizations

Single node HSW

<table>
<thead>
<tr>
<th>Cores (HSW)</th>
<th>First version</th>
<th>After dungeon</th>
<th>Final version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>478</td>
<td>327</td>
<td>309</td>
</tr>
<tr>
<td>2</td>
<td>268</td>
<td>175</td>
<td>166</td>
</tr>
<tr>
<td>4</td>
<td>149</td>
<td>88</td>
<td>89</td>
</tr>
<tr>
<td>8</td>
<td>79</td>
<td>47</td>
<td>47</td>
</tr>
<tr>
<td>16</td>
<td>45</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>32</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Single node KNL

<table>
<thead>
<tr>
<th>Cores (KNL)</th>
<th>First version</th>
<th>After dungeon</th>
<th>Final version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1248</td>
<td>1074</td>
<td>774</td>
</tr>
<tr>
<td>2</td>
<td>662</td>
<td>547</td>
<td>397</td>
</tr>
<tr>
<td>4</td>
<td>350</td>
<td>275</td>
<td>200</td>
</tr>
<tr>
<td>8</td>
<td>182</td>
<td>139</td>
<td>101</td>
</tr>
<tr>
<td>16</td>
<td>92</td>
<td>69</td>
<td>52</td>
</tr>
<tr>
<td>32</td>
<td>48</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>64</td>
<td>26</td>
<td>21</td>
<td>20</td>
</tr>
</tbody>
</table>
Science Purpose: Spectroscopy for Dark Energy science
- 3D map of the Universe over 10 billion years
- Spectra of 10’s of millions of galaxies and quasars
- Create flux-calibrated 1D tables of flux vs wavelength of Galaxies, quasars, etc. from 2D CCD image frames

Algorithms and Methods
- Scientific Python stack (NumPy, SciPy, etc.; threaded)
- Linear algebra (esp. Hermitian eigen-decomposition)
- Special function evaluations, fitting functions to data
- MPI (mpi4py) data-parallel processing + Shifter to scale up

Production Requirements
- Real-time pressure to do real-time survey planning each day
Simulation Code (Simulate Spectra on CCDs): 1.5-1.7x on HSW, multi-node scaling w/MPI
- Numba JIT compilation to speed up 2 lines of expensive matrix slicing
- MPI work to scale up the code:
  - Broadcast/reduce to scatter/gather where best use, complete initial I/O faster
  - Multi-level Comm scheme to optimally fill nodes
  - Scale tests up to 60 nodes so far, will be used in production soon
  - Single exposure (30 frames simultaneously) in 8 minutes
  - Roughly equal performance between multiprocessing and MPI on single node

Main Extraction Code (1D traces from CCD images)
- Main bottleneck is `legval` in NumPy (scalar/vector args) observed at first Dungeon.
- Precompute `legval` w/large vector input (not scalar): promising but delicate refactor.
- Also `legval` itself: 4x speedup with loop unrolling and Numba.
- Using some of the code as a testbed for initial experimenting with PyPy.
Cosmic Microwave Background: HPC and Data

- APEX-SZ: 330 detectors
- SPT-SZ: 960 detectors
- POLARBEAR-1: 1274 detectors, Dual-Polarization
- POLARBEAR-2/SPT-3G: 8,000/15,000 detectors, Dual-Polarization, 2-3 Color/pixel

Graph showing the growth of data volume and peak flop/s over time.
Preparations for CMB-S4

Future CMB experiments:
10x more detectors
More telescopes
New systematics (weather at different sites)

“TOAST” App Hero Run in June 2017

Simulation and map-making:
50,000 detectors
30X Planck mission data
1 year of simulated observations from Chile
Including effects of atmosphere

Using all of NERSC’s Cori Phase II:
658,784 Intel Xeon Phi (Knights Landing) Cores
Hybrid MPI/OpenMP (150K MPI ranks)
First full-KNL production Shifter (container) job
Hybrid Python/C++

(Julian Borrill, Reijo Keskitalo, Ted Kisner, LBL)
Conclusion

Python fills numerous critical roles at HPC scientific computing centers like NERSC.

Especially true in experimental/observational sciences, data processing/analysis more than analytics for now.

Achieving good Python performance is challenging and users (not often HPC-oriented) need to partner with center staff and vendors/developers to get it.