PYTHON: PRODUCTIVITY WITH PERFORMANCE

Heidi Pan

Scripting Analyzers and Tools Group (Python, R, Julia, Go)
Intel
Python for Data Science & Machine Learning

Kaggle ML and Data Science Survey, 2017

Tools used in work


KDNuggets Analytics, Data Science, Machine Learning Software Poll, top tools share, 2015-2017

From Prototype to Production

High Performance Python
High Performance Python

Python Libraries

Intel® Performance Libraries

more cores, more threads, wider vectors, ...

(generations of processors)
Accelerating Machine Learning

- Efficient memory layout via Numeric Tables
- Blocking for optimal cache performance
- Computation mapped to most efficient matrix operations (in MKL)
- Parallelization via TBB
- Vectorization

Try it out! conda install -c intel scikit-learn
Accelerating K-Means

Performance speedups for Intel® Distribution for Python* scikit-learn on Google Cloud Platform’s 96 vCPU instance Intel® Xeon™ Processors

- **PCA-based**: 23X faster
- **Random**: 21X faster
- **K-means clustering algorithms**: 22X faster

**Time (Geomean, in seconds)**

**System Configuration**: GCP VM, zone us-central1-c; 96 vCPU, Intel Skylake; 360 GB memory. Ubuntu 16.04.3 LTS; Linux instance-1 4.10.0-38-generic #42~16.04.1-Ubuntu SMP Tue Oct 10 16:32:20 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux; Intel® Distribution for Python* from Docker image intelpython/intelpython3_fullopt (created 2017-09-12T20:10:42.86297353592); Stock Python*: pip install scikit-learn

Scaling Machine Learning Beyond a Single Node

scikit-learn

Intel® Data Analytics Acceleration Library (DAAL)

Intel® Math Kernel Library (MKL)

Intel® Threading Building Blocks (TBB)

daal4py

Intel® MPI

Simple Python API

Powered by DAAL

Scalable to multiple nodes

Try it out! conda install -c intel/label/test daal4py
import daal4py as d4p

# initialize distributed execution environment
d4p.daalinit()

# load data from csv files into numpy arrays
files = ["kmeans_dense.csv", ...]
dfin = [loadtxt(x, delimiter=',' ) for x in files]

# compute initial centroids & kmeans clustering
centroids = d4p.kmeans_init(10, t_method="plusPlusDense", distributed=True)
result = d4p.kmeans(10, distributed=True).compute(dfin, centroids.compute(dfin))

mpirun -n 4 -genv DIST_CNC=MPI python ./kmeans.py
Strong & Weak Scaling of K-Means via Daal4py

**Optimization Notice**

*Other names and brands may be claimed as the property of others.

---

**Hardware**
- Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz, EIST/Turbo on 2 sockets, 20 Cores per socket
- 192 GB RAM
- 16 nodes connected with Infiniband

**Operating System**
- Oracle Linux Server release 7.4

**Data Type**
- double

---

**daal4py: K-Means Distributed Scalability**
- 16M observations, 300 features, 10 clusters

**Runtime (sec)**

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>(strong) 2 processes per node; fixed total input size</th>
<th>(weak) 2 processes per node; input size per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>120</td>
</tr>
</tbody>
</table>
Productivity with Performance via Intel® Python*

Intel® Distribution for Python*

Easy, out-of-the-box access to high performance Python

- Prebuilt accelerated solutions for data analytics, numerical computing, etc.
- Drop in replacement for your existing Python. No code changes required.

Learn More: software.intel.com/distribution-for-python
Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

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. For more complete information visit www.intel.com/benchmarks.

Copyright © 2018, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

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
Python at NERSC

Rollin Thomas
NERSC Data and Analytics Services
IXPUG
2018-05-10
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
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

Data                    Model                  Residuals

Science via Python@NERSC
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 -8° < b < +46° in equatorial coordinates and (a) > 18° in Galactic coordinates.

To address this goal, the Legacy Surveys are conducting imaging projects on different telescopes, described in more depth at the following links:

- The BigSky Science
  - Surveys (BASS)
- The DietCam Legacy
  - Survey (DLExtra)
- The Megapixel 2-band
  - Legacy Survey (M2LS)
Interactive Supercomputing

- JupyterHub
- Web Server
- Cori Login Node
- Cori Compute Node
- Notebook Server Process
- Kernel Process
- --qos=interactive
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%)

Users also added:
Numba
Healpy

<10% use:
Dask
PySpark
Theano
PyTorch
PyDAAL

* or concurrent.futures
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
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. Easily provide access to Intel Python Distribution.

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:
```
python setup.py build --mpicc=cc
python setup.py install
```

Cray-provided Compiler wrapper

Cori Aries Interconnect
Containers and Python go well together at NERSC

Motivations, esp. for data science:
- Flexibility
- Consistency
- Convenience
- 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
ldconfig when you build image

Previous 6 months
150 nodes
4800 MPI ranks
import numpy
import astropy

[Median launch time incl. MPI_Init()]
Things will work, but 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.
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
- Expanded Access to KNL + Data Ecosystem
- Developer Workshops, "Dungeons"
- Postdoc Fellowship Program
- Leverage Community Efforts
- Training Docs, Online Modules
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*
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
DESI Optimization & Scaling

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.
PyHPC 2018

At SC18!

8th Workshop on Python for High-Performance and Scientific Computing
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.