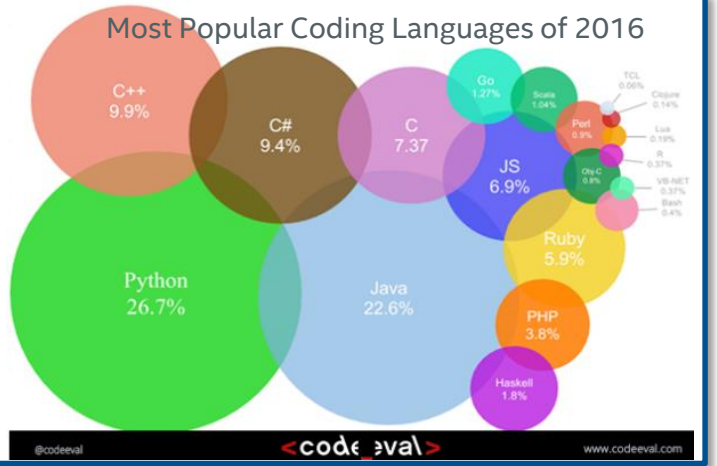
BENCHMARKS INTEL® DISTRIBUTION FOR PYTHON*

Python* Landscape

Adoption of Python continues to grow among domain experts & developers for its productivity benefits



Challenge#1

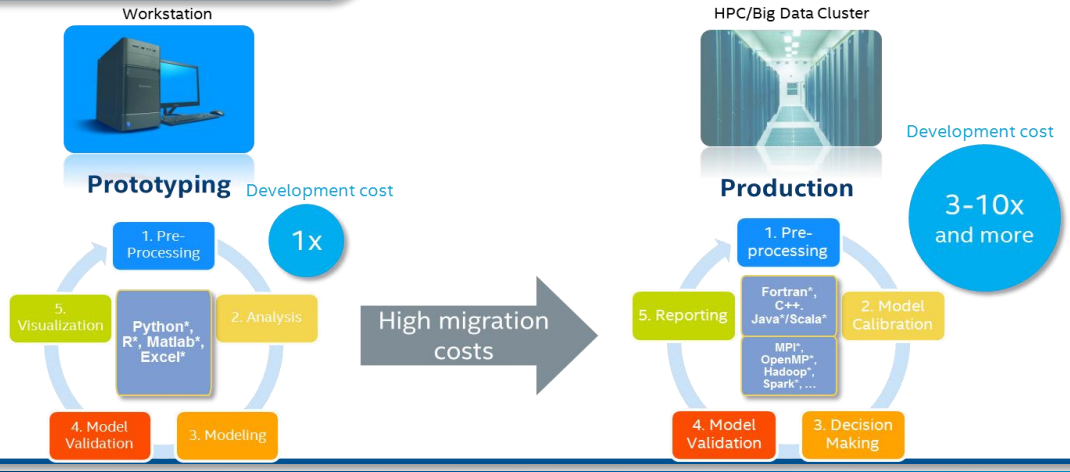
Domain experts are not professional software programmers

Intel's Python Tools

- Accelerate Python performance
- Enable easy access
- Empower the community

Challenge#2

Python performance limits migration to production systems



What's inside Intel® Distribution for Python

High Performance Python* for Scientific Computing, Data Analytics, Machine Learning

FASTER PERFORMANCE	GREATER PRODUCTIVITY	ECOSYSTEM COMPATIBILITY
Performance Libraries, Parallelism, Multithreading, Language Extensions	**Prebuilt & Accelerated Packages**	**Supports Python 2.7 & 3.6, conda, pip**
Accelerated NumPy/SciPy/scikit-learn with Intel® MKL[1] & Intel® DAAL[2]	Prebuilt & optimized packages for numerical computing, machine/deep learning, HPC, & data analytics	Compatible & powered by Anaconda*, supports conda & pip
Data analytics, machine learning & deep learning with scikit-learn, pyDAAL	Drop in replacement for existing Python - No code changes required	Distribution & individual optimized packages also available at conda & Anaconda.org, YUM/APT, Docker image on DockerHub
Scale with Numba* & Cython*	Jupyter* notebooks, Matplotlib included	Optimizations upstreamed to main Python trunk
Includes optimized mpi4py, works with Dask* & PySpark*	Conda build recipes included in packages	Commercial support through Intel® Parallel Studio XE 2017
Optimized for latest Intel® architecture	Free download & free for all uses including commercial deployment	

Intel® Architecture Platforms

Operating System: Windows*, Linux*, MacOS[1]*

[1]Intel® Math Kernel Library
[2]Intel® Data Analytics Acceleration Library

[1] Available only in Intel® Parallel Studio Composer Edition.

Intel® Distribution for Python* 2018, Installing

Standalone Installer

Download full installer from
https://software.intel.com/en-us/intel-distribution-for-python

Anaconda.org
Anaconda.org/intel channel

```
> conda config --add channels intel
> conda install intelpython3_full
> conda install intelpython3_core
```

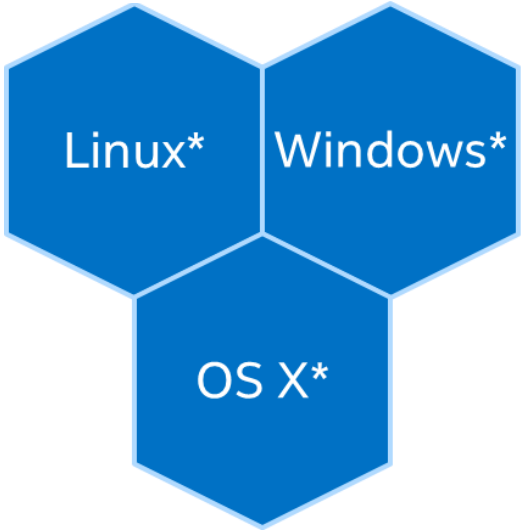
Docker Hub

```
docker pull intelpython/intelpython3_full
```

YUM/APT

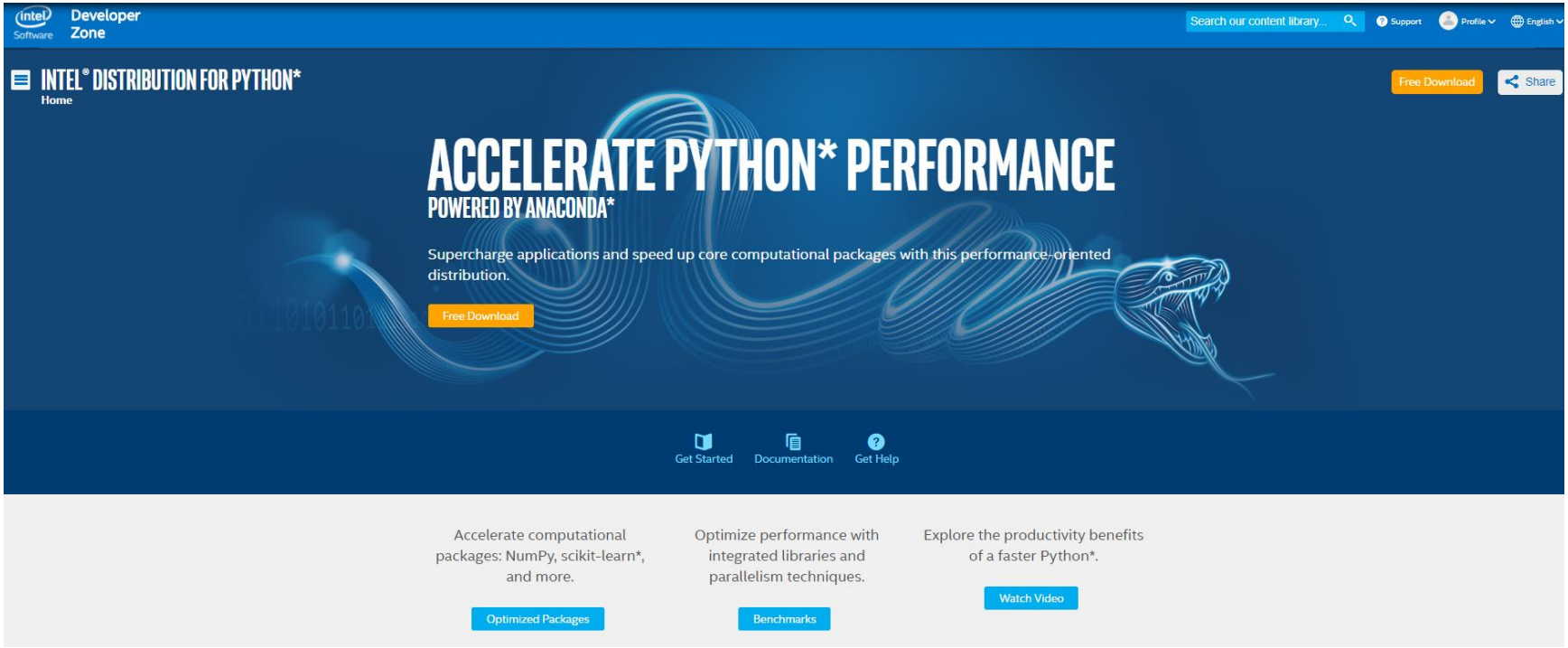
Access for yum/apt:
https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python

2.7, 3.5 & 3.6



Intel® Distribution for Python* 2017/2018, How to Get

https://software.intel.com/en-us/distribution-for-python



Introduction

- Objective:
  - In these activities, you will Install Conda*, IDP, pip numpy, run Intel® IDP and Intel® MKL based codes and
  - make the performance comparisons
- Requirements:
  - Intel® Parallel Studio XE 2018 Composer Edition with Intel® C++ Compiler
  - Intel® Distribution for Python* 2018
  - **Linux*** OS supported by Intel® C++ Compiler
  - Recommended to have at least 3**nd** **generation Intel® Core™ processor** (with Intel® AVX2)
- **Time : 25 min**

Install Conda*

Activity #1

Package, dependency and environment management for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN

- Download miniconda
  $ **wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh**
- Setup
  $ **bash ./Miniconda3-latest-Linux-x86_64.sh**

Install NumPy* (OpenBLAS)

Activity #1

- **pip install numpy**
- **source activate**

#open and navigate cat/less dgemm_pip.py file

- run: **python dgemm_pip.py**
- record execution time ____
- Uninstall numpy : **pip uninstall numpy**

Intel® Distribution for Python* Activity #1,cont

Use transparently with Conda

- Add IDP channel
  $ **conda create -y -n intel3 -c intel python=3 numpy scipy**
- Activate IDP
  $ **source activate intel3**

Intel® Distribution for Python: BLAS, Activity #2

- **. ./mklunset.sh**
- #open and navigate cat/less dgemm_idp.py
- # Run python execution by:
  - **python dgemm_idp.py**
  - record execution times _____
- # Check if mkl is used:
  - **export MKL_VERBOSE=1**
- **Run the code: python dgemm_idp.py**

Intel® Distribution for Python: BLAS, Activity #2,cont

- # Open and navigate mkl_dgemm.cpp file
- # Set mkl and compiler's environments and build the native code:
  - **source /opt/intel/compilers_and_libraries/linux/bin/compilervars.sh intel64**
- # Check version of icc: icc --version or which icc
- # Build the executable:
  - **icc -mkl mkl_dgemm.cpp**
- # Disable verbose mode (mkl_get_version is used)
  - **export MKL_VERBOSE=0**

Intel® Distribution for Python: BLAS, Activity #2,cont

- Run
  - **./a.out**
  - **record execution times _____**
- # compare the overhead:
- # check the overhear and performance results
  - Pip NumPy : ~ 16sec
  - IDP NumPy : ~ 9 sec
  - Intel MKL : ~ 8 sec
- Note: MxNxK = 10K x 10K x 10K

Intel® Distribution for Python, LAPACK, Activity #3

- #Open and navigate LU.py and LU_mkl.cpp files
- # Run python execution by:
  - **python LU.py**
  - record execution times _____
- # Check Check the backend
  - **export MKL_VERBOSE=1**
  - **python.py LU.py**
  - **See the outputs….**

```
Numpy + Intel(R) MKL: THREADING LAYER: (null)
Numpy + Intel(R) MKL: setting Intel(R) MKL to use INTEL OpenMP runtime
Numpy + Intel(R) MKL: preloading libiomp5.so runtime
MKL_VERBOSE Intel(R) MKL 2018.0 Update 2 Product build 20180127 for Intel(R) 64
architecture Intel(R) Advanced Vector Extensions 512 (Intel(R) AVX-512) enabled
processors, Lnx 2.30GHz lp64 intel_thread
DGETRF(1000,1000,0x7f75b6544040,1000,0x564b37b3a140,0) 9.42ms CNR:OFF
Dyn:1 FastMM:1 TID:0  NThr:4
MKL_VERBOSE DLASWP(1000,0x7f75b4e5e040,1000,1,1000,0x564b37b3a140,-1)
2.37ms CNR:OFF Dyn:1 FastMM:1 TID:0  NThr:4

M x N  ==  1000 1000 ... SciPy LU Execution Time ==  0.18822526931762695 sec
```

Intel® Distribution for Python, LAPACK, Activity #3. cont

- #Compile and execute mkl code:
  - icc -mkl LU_mkl.cpp
  - ./a.out
  - record execution times _____
  - # compare the overhead:
    - M x N == 1000 x 1000 … SciPy LU Execution Time ~ t1 sec
    - M x N == 1000 x 1000 … Intel MKL LU Execution Time ~ t2 sec

Legal Disclaimer & Optimization Notice

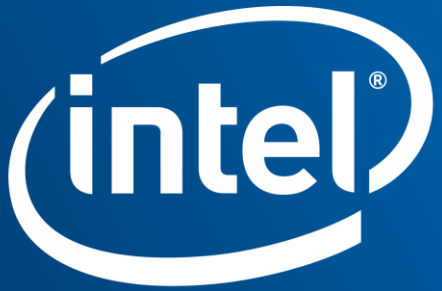
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may 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.

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.

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



Software