July 2024

Overview of Python and classical ML optimizations

Stefana Raileanu – AI Software Solutions Engineer

intel®

Agenda

- oneAPI Introduction
- Intel® Distribution for Python
    - numpy
    - Data Parallel Extensions for Python*
- Classical ML
    - scikit-learn
    - XGBoost
    - daal4py
- Modin*

Intel's oneAPI Ecosystem

Built on Intel's Rich Heritage of CPU Tools Expanded to XPUs

oneAPI

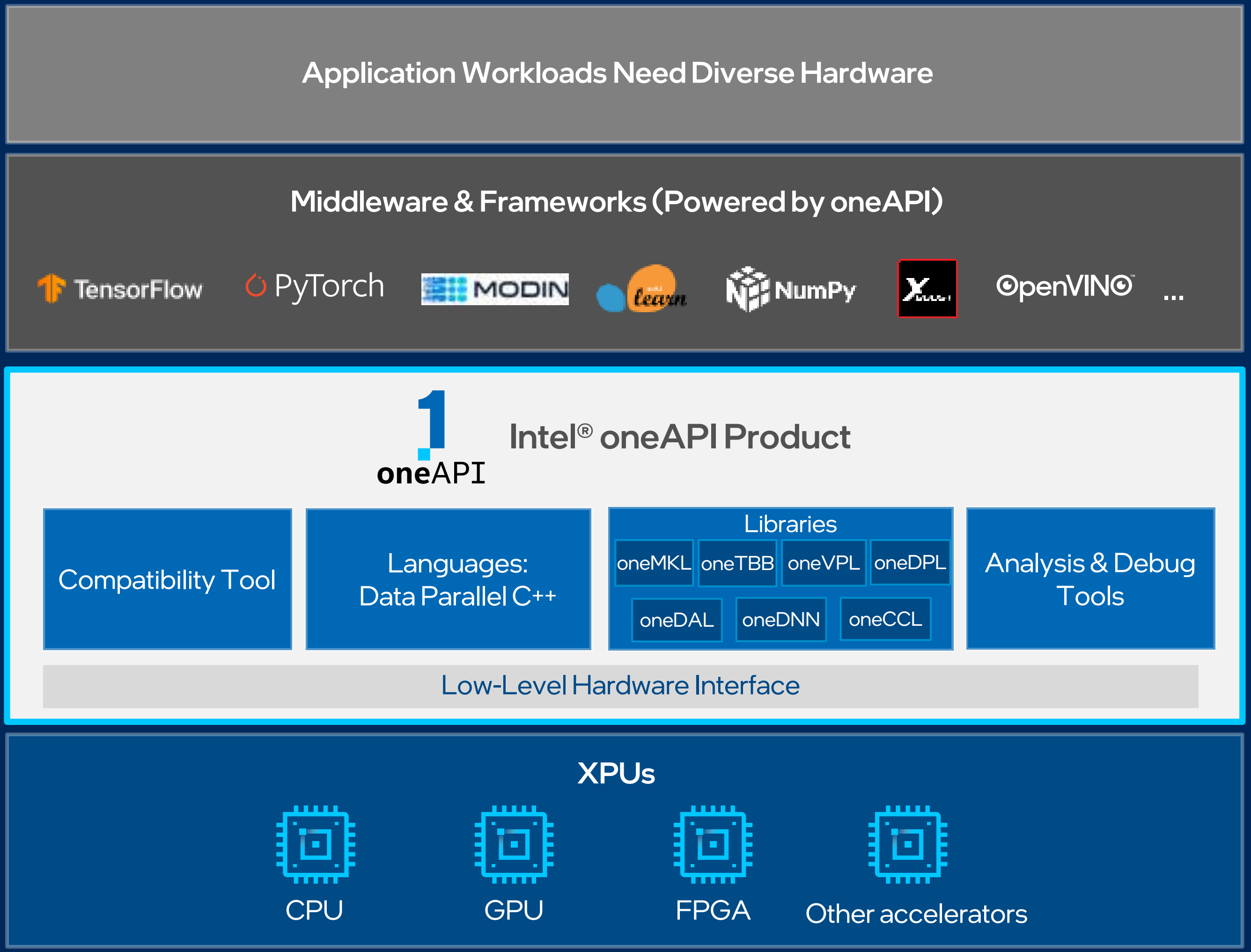
A cross-architecture language based on C++ and SYCL standards

Powerful libraries designed for acceleration of domain-specific functions

A complete set of advanced compilers, libraries, and porting, analysis and debugger tools

Powered by oneAPI

Frameworks and middleware that are built using one or more of the oneAPI industry specification elements, the DPC++ language, and libraries listed on oneapi.com.



[Available Now]

Visit software.intel.com/oneapi for more details
Some capabilities may differ per architecture and custom-tuning will still be required. Other accelerators to be supported in the future.

Linux Foundation's Unified Acceleration Foundation (UXL)

UXL is an evolution of the oneAPI initiative

UXL FOUNDATION
Unified Acceleration

- This cross-industry group is committed to delivering an open accelerator software ecosystem to simplify development of applications for cross-platform deployment.

- Intel will contribute its oneAPI specification to the UXL Foundation to help drive cross-platform development across architectures.

oneAPI elements Intel is donating to UXL

Intel AI Software Portfolio

Engineer Data → Create Models → Optimize & Deploy

Intel-optimized open-source AI tools and frameworks

Modin, SciPy, pandas, NumPy, Apache Spark, Numba

Data Analytics at Scale

dmlc XGBoost, PyTorch, ONNX Runtime, scikit-learn, TensorFlow, DirectML, SigOpt AutoML, intel Neural Compressor

Machine & Deep Learning Frameworks, Optimization and Deployment Tools

Intel distribution

OpenVINO

Write Once
Deploy Anywhere

oneAPI

- Intel® oneAPI Deep Neural Network Library
- Intel® oneAPI Collective Communications Library
- Intel® oneAPI Math Kernel Library
- Intel® oneAPI Data Analytics Library

Open, cross-architecture programming model for CPUs, GPUs, and other accelerators

CLOUD & ENTERPRISE | CLIENT & WORKSTATION | EDGE

Intel® Tiber™ Developer Cloud
cloud.intel.com
Try the latest Intel tools and hardware, and access optimized AI Models

Open Platform for Enterprise AI
Partner

Intel® Tiber™ AI Studio
Full stack ML operating system

Hugging Face
Intel optimizations and fine-tuning recipes, optimized inference models, and model serving

Note: components at each layer of the stack are optimized for targeted components at other layers based on expected AI usage models, and not every component is utilized by the solutions in the rightmost column

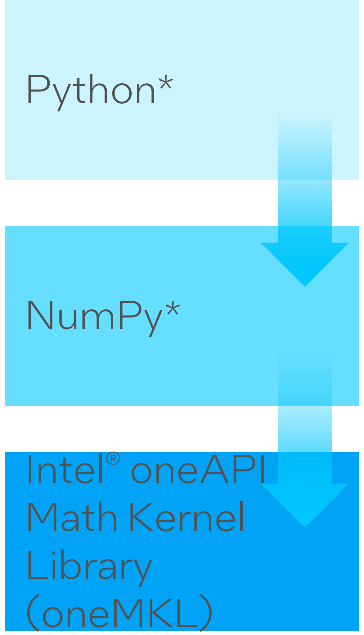
Intel® Distribution For Python*

Optimizations for NumPy and SciPy

Intel® Performance Optimization with NumPy* and SciPy*

The layers of quantitative Python*

- The Python* language is interpreted and has many type checks to make it flexible
- Each level has various tradeoffs; NumPy* value proposition is immediately seen
- For best performance, escaping the Python* layer early is best method



Enforces Global Interpreter Lock (GIL) and is single-threaded, abstraction overhead. No advanced types.

Gets around the GIL (multi-thread and multi-core)
BLAS* API can be the bottleneck

*Basic Linear Algebra Subprograms (BLAS) [CBLAS]

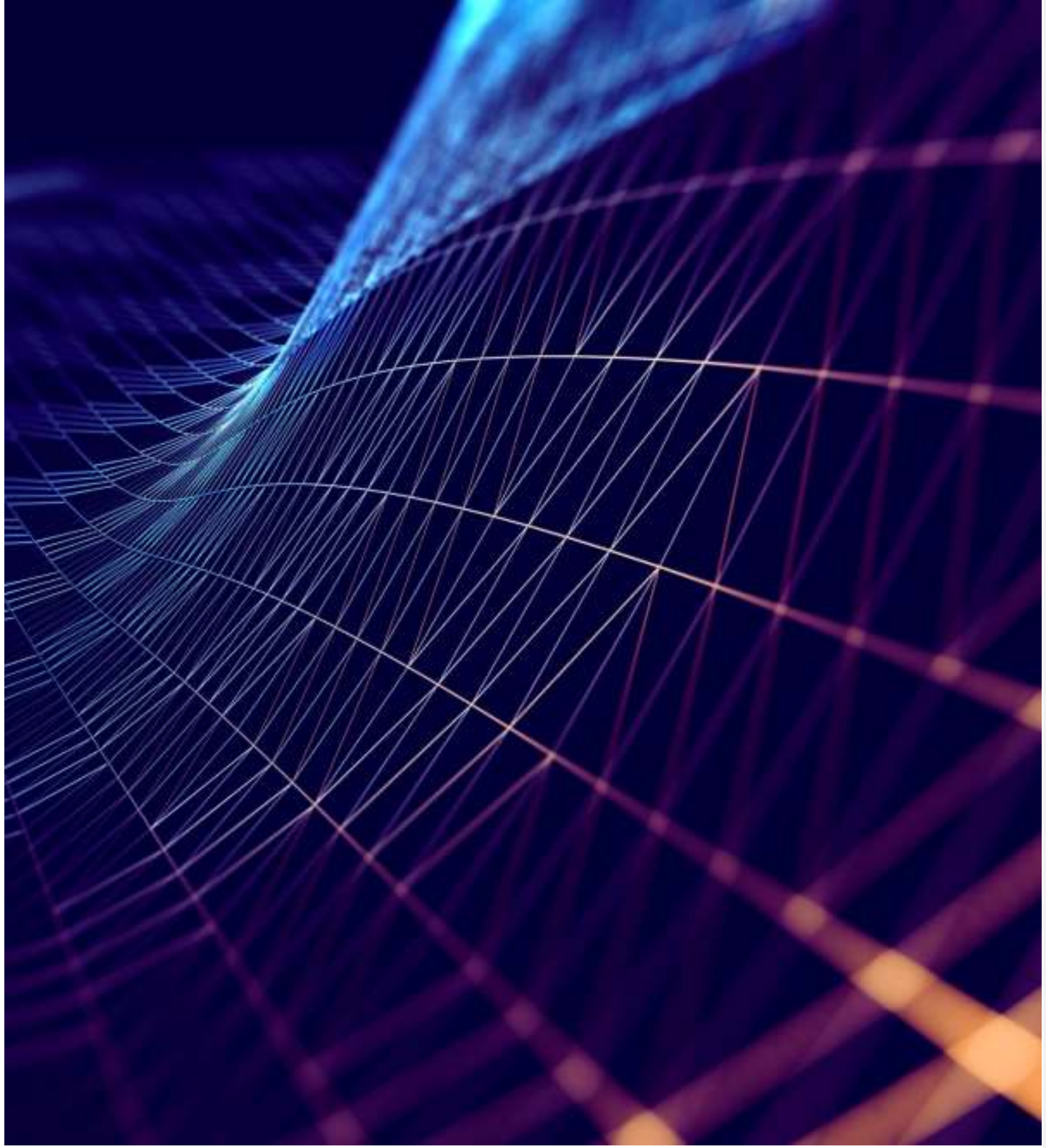
Gets around BLAS API bottleneck
Much stricter typing
Fastest performance level
Dispatches to hardware vectorization

Intel® oneMKL included with Anaconda standard bundle; is Free for Python

NumPy* and SciPy* Optimizations

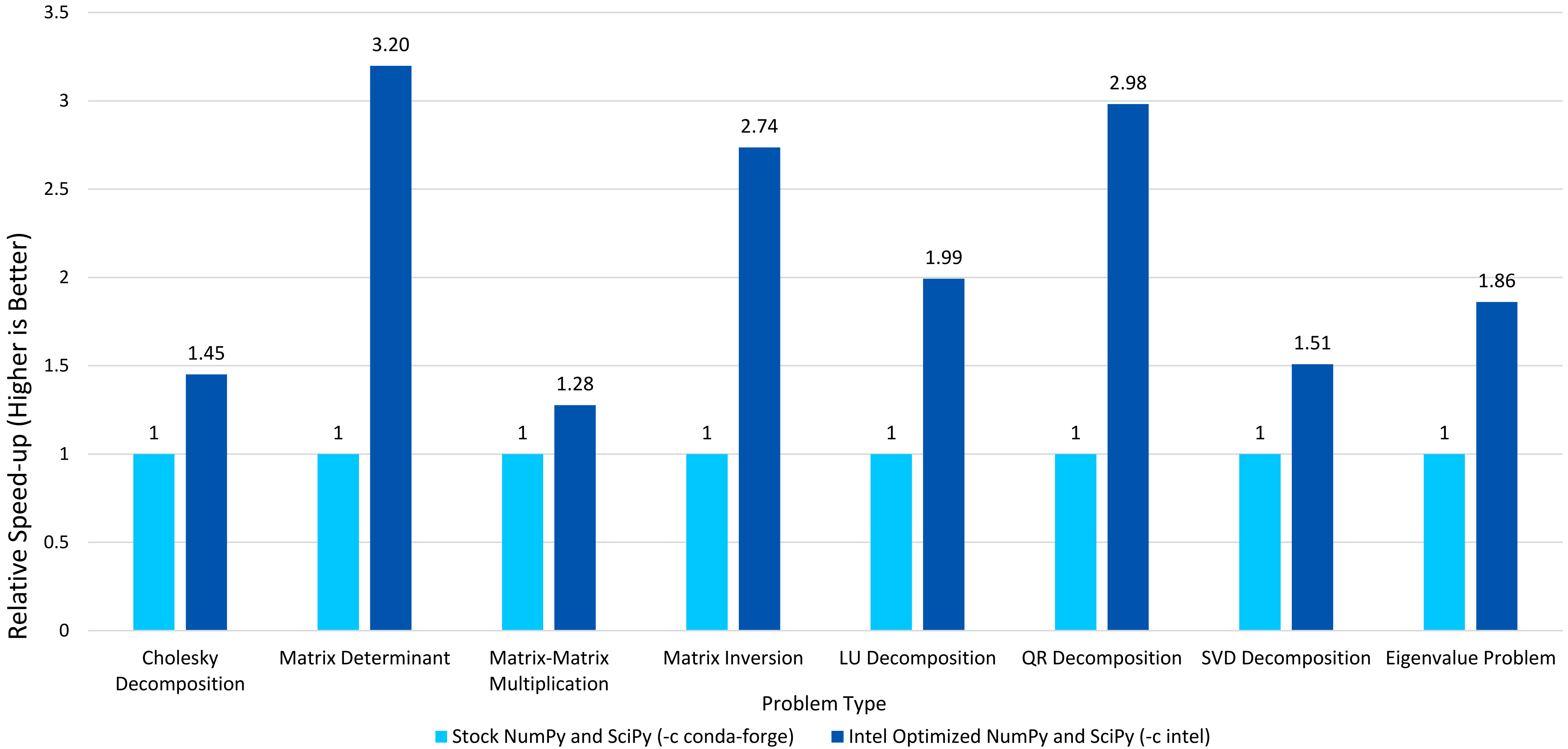
Scope

- BLAS/LAPACK using oneMKL
- oneMKL-based FFT functionality
- Vectorized, threaded universal functions
- Use of Intel® C Compiler, and Intel® Fortran Compiler
- Aligned memory allocation
- Threaded memory copying



Intel Optimized NumPy* and SciPy* Linear Algebra Performance

Performance is Increased up to 3.2x with Intel Optimizations



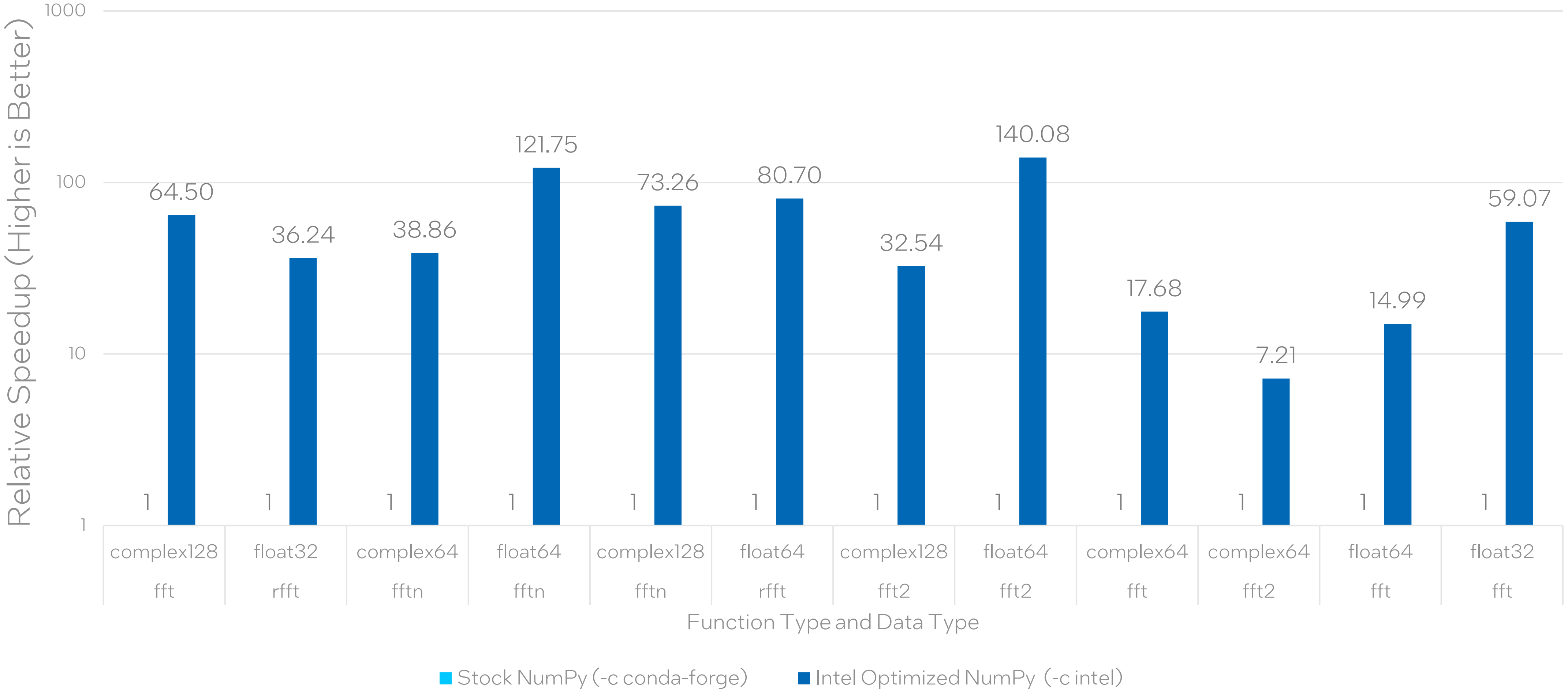
Intel Optimizations for NumPy* & SciPy* compared to conda-forge channel NumPy* & SciPy* Performance for Linear Algebra on Intel® Xeon® Platform 8480+

Testing Date: Performance results are based on testing by Intel as of July 15, 2023. **Configuration Details and Workload Setup: System:** cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (ibench Linear Algebra), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

See backup for workloads and configurations. Results may vary.

Intel Optimized NumPy* Fast Fourier Transform Performance

Performance is Increased up to 140x with Intel Optimizations



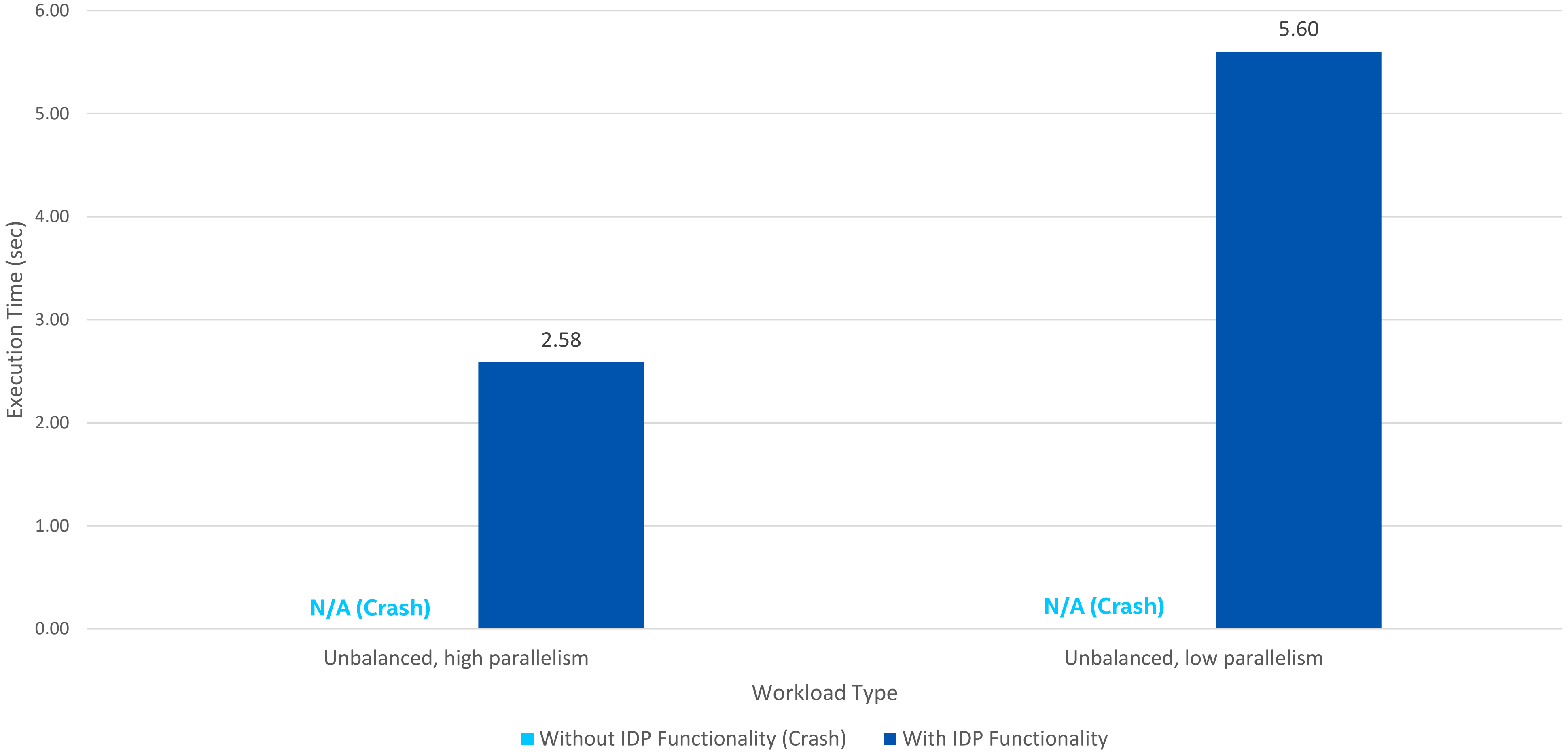
Fast Fourier Transform NumPy* performance intel vs. conda-forge on Intel® Xeon® Platform 8480+

Testing Date: Performance results are based on testing by Intel as of March 5, 2023. **Configuration Details and Workload Setup: System:** cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (fft_benchmark, blackscholes_bench, composability_bench), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

See backup for workloads and configurations. Results may vary.

Intel® Distribution for Python Oversubscription Performance

Successful Unbalanced Workload Performance with Composable Parallelism Enabled



PERFORMANCE USING INTEL® DISTRIBUTION FOR PYTHON* ON INTEL® XEON® PLATINUM 8480+ TO AVOID OVERSUBSCRIPTION PROBLEMS

Testing Date: Performance results are based on testing by Intel as of March 5, 2023. Configuration Details and Workload Setup:System: cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (fft_benchmark, blackscholes_bench, composability_bench), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed. Commands for stock non-IDP functionality: High parallelism: python dask_sh_mt.py, Low parallelism: python numpy_sl_mp.py 4; Commands for IDP functionality: High parallelism: env KMP_COMPOSABILITY=mode=counting python dask_sh_mt.py, Low paralielsm: python –m tbb numpy_sl_mp.py

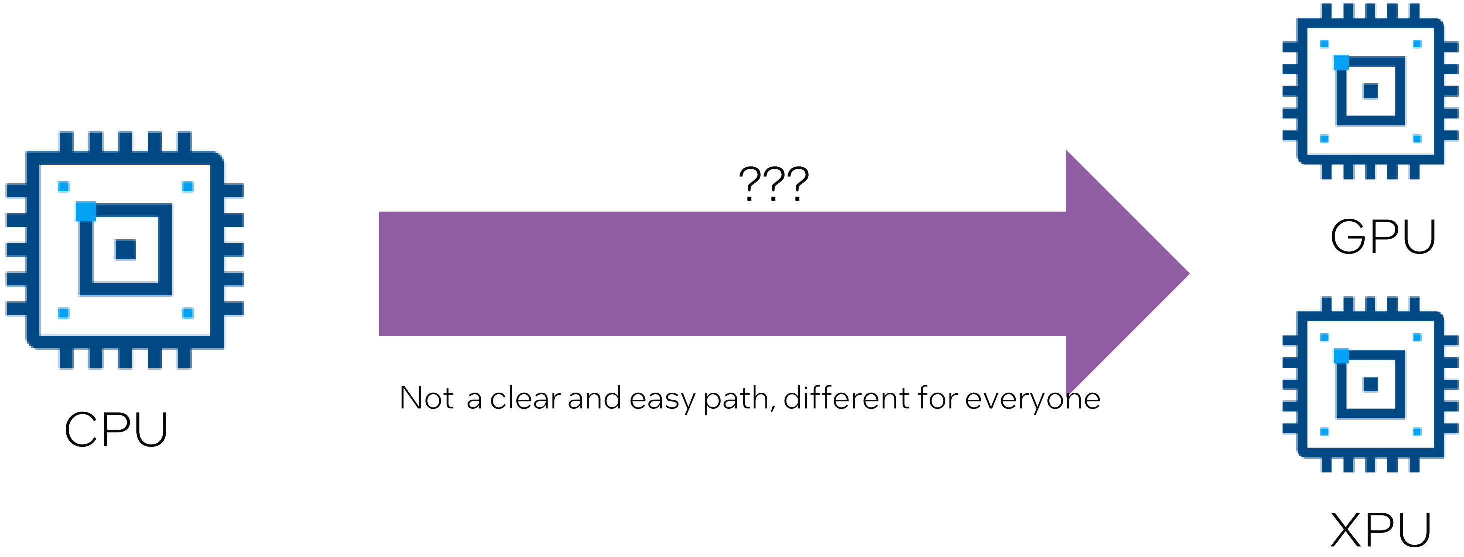
See backup for workloads and configurations. Results may vary.

Data Parallel Extensions for Python*

Current Gaps & Bottlenecks with Python

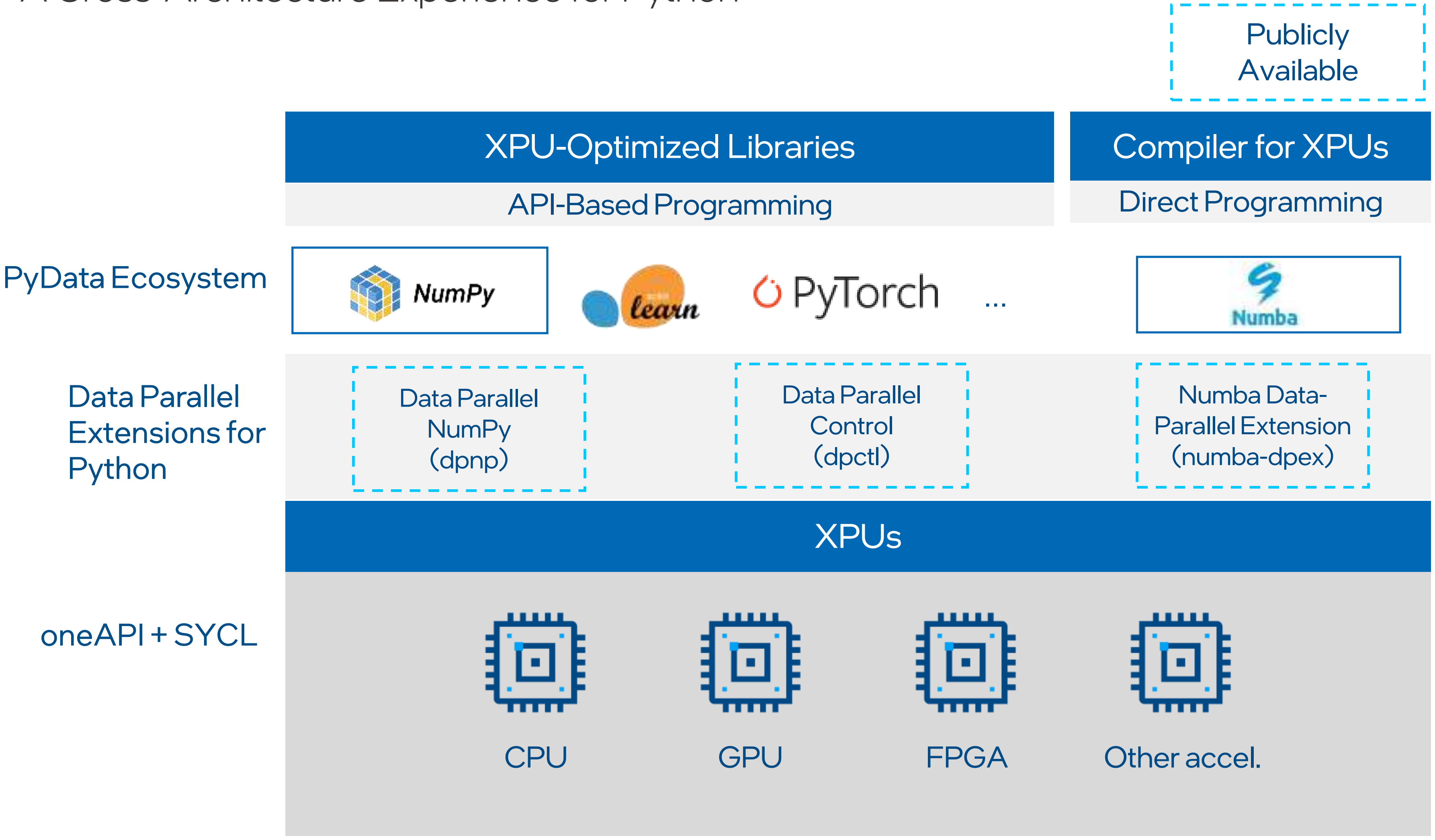
- No heterogenous computing opportunities for Python developers

- Some frameworks/companies build on CPU but no GPU support for this software
- **Vendor lock-in when using certain GPUs and other devices**
  - Significant development and maintenance costs for codes targeting both GPU and CPU, and/or other devices
- Developers need to have a different skillset and take extra time to program



Data Parallel Extensions for Python* language (DPEX)

A Cross-Architecture Experience for Python*



- Simple, unified offload programming model
- Standards-based: Python* Data API Standards + Khronos* SYCL + extensions
- Interoperates with vast Python ecosystem on host
- A free, open-source solution

dpnp: Data Parallel Extension for NumPy* API

Drop-in replacement for NumPy to allow heterogenous computation on SYCL devices

oneAPI

Original CPU script

```
import numpy as np

x = np.array([[1, 1], [1, 1]])
y = np.array([[1, 1], [1, 1]])

res = np.matmul(x, y)
```

Same functionality as NumPy, running on GPU*

*default device is SYCL GPU, if available. No specification required

```
import dpnp as np

x = np.array([[1, 1], [1, 1]])
y = np.array([[1, 1], [1, 1]])

res = np.matmul(x, y) # res resides on gpu
```

Modified XPU script – specify a device to run operations there!

```
import dpnp as np

x = np.array([[1, 1], [1, 1]], device="gpu2")
y = np.array([[1, 1], [1, 1]], device="gpu2")

res = np.matmul(x, y) # res resides on gpu
```

intel.

Get started. Documentation

- Documentation:
- [Data Parallel Extensions for Python* Language]
- [Data Parallel Control Library (dpctl)]
- [Data Parallel Extension for NumPy*]
- [Data Parallel Extension for Numba*]

- Installation:
- The easiest way to install Data Parallel Extensions for Python is to install numba-dpex:

  - Pip: pip install numba-dpex

- These commands install numba-dpex along with its dependencies, including dpnp, dpctl, and required compiler runtimes. Check out the prerequisites here.

Intel® Extension for Scikit-learn*

Intel® Extension for Scikit-learn*

scikit-learn*

```
from sklearn.svm import SVC
X, Y = get_dataset()


clf = SVC().fit(X, y)
res = clf.predict(X)
```

scikit-learn* with
Intel CPU opts

```
from sklearnex import patch_sklearn
patch_sklearn()

from sklearn.svm import SVC

X, Y = get_dataset()

clf = SVC().fit(X, y)
res = clf.predict(X)
```

Same Code,
Same Behavior

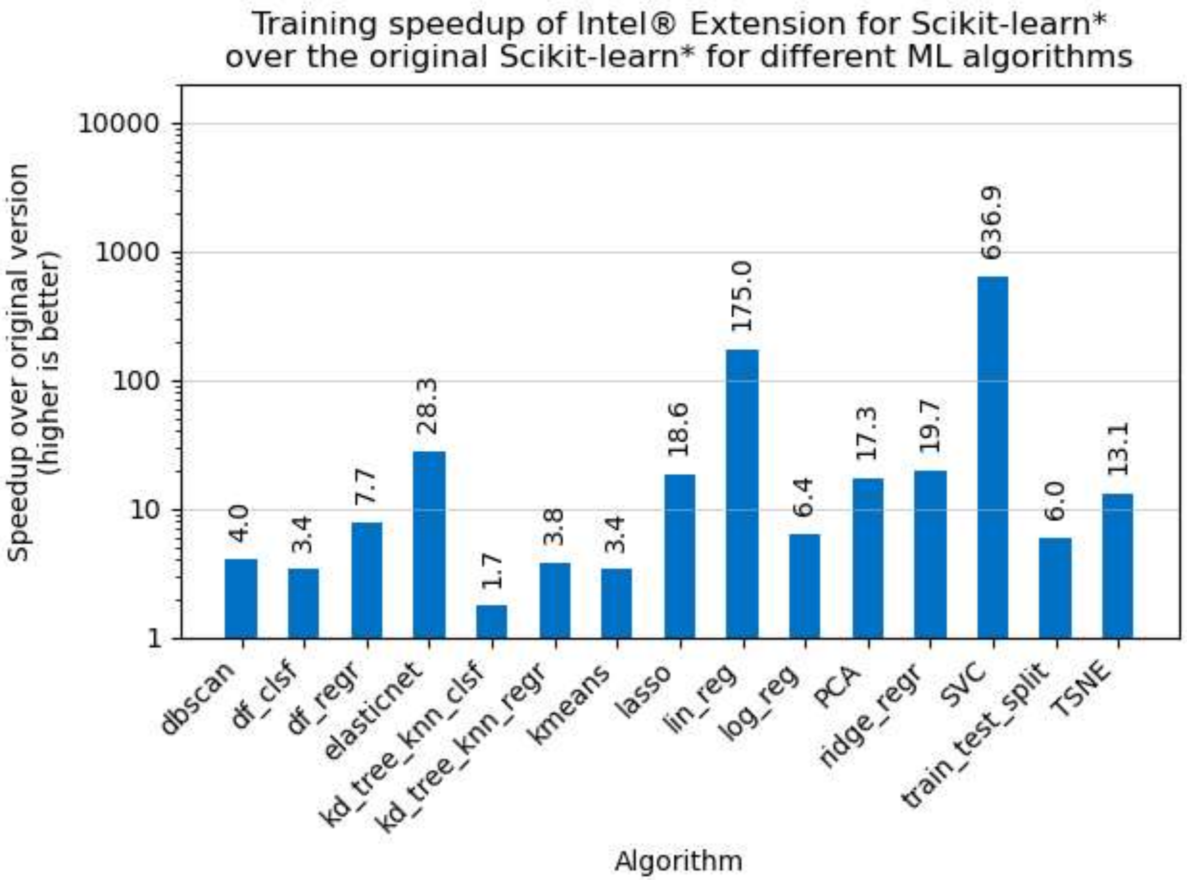
PASSED

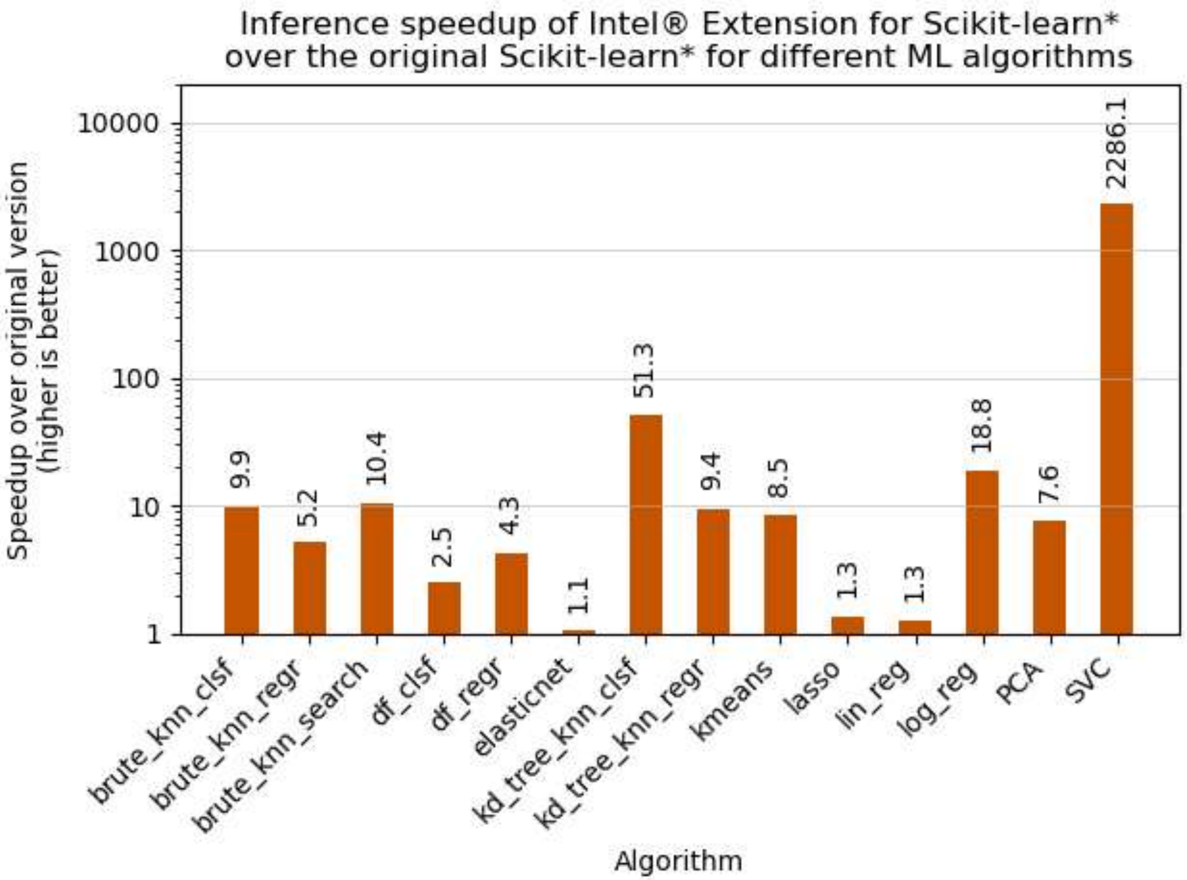
- scikit-learn*, <u>not</u> scikit-learn*-like
- scikit-learn* conformance (mathematical equivalence) defined by scikit-learn* Consortium, continuously vetted by public CI

Available through:

- conda install scikit-learn-intelex
- conda install –c intel scikit-learn-intelex
- conda install –c conda-forge scikit-learn-intelex
- pip install scikit-learn-intelex

Training and Inference Performance Gains with Intel® Extension for Scikit-Learn*





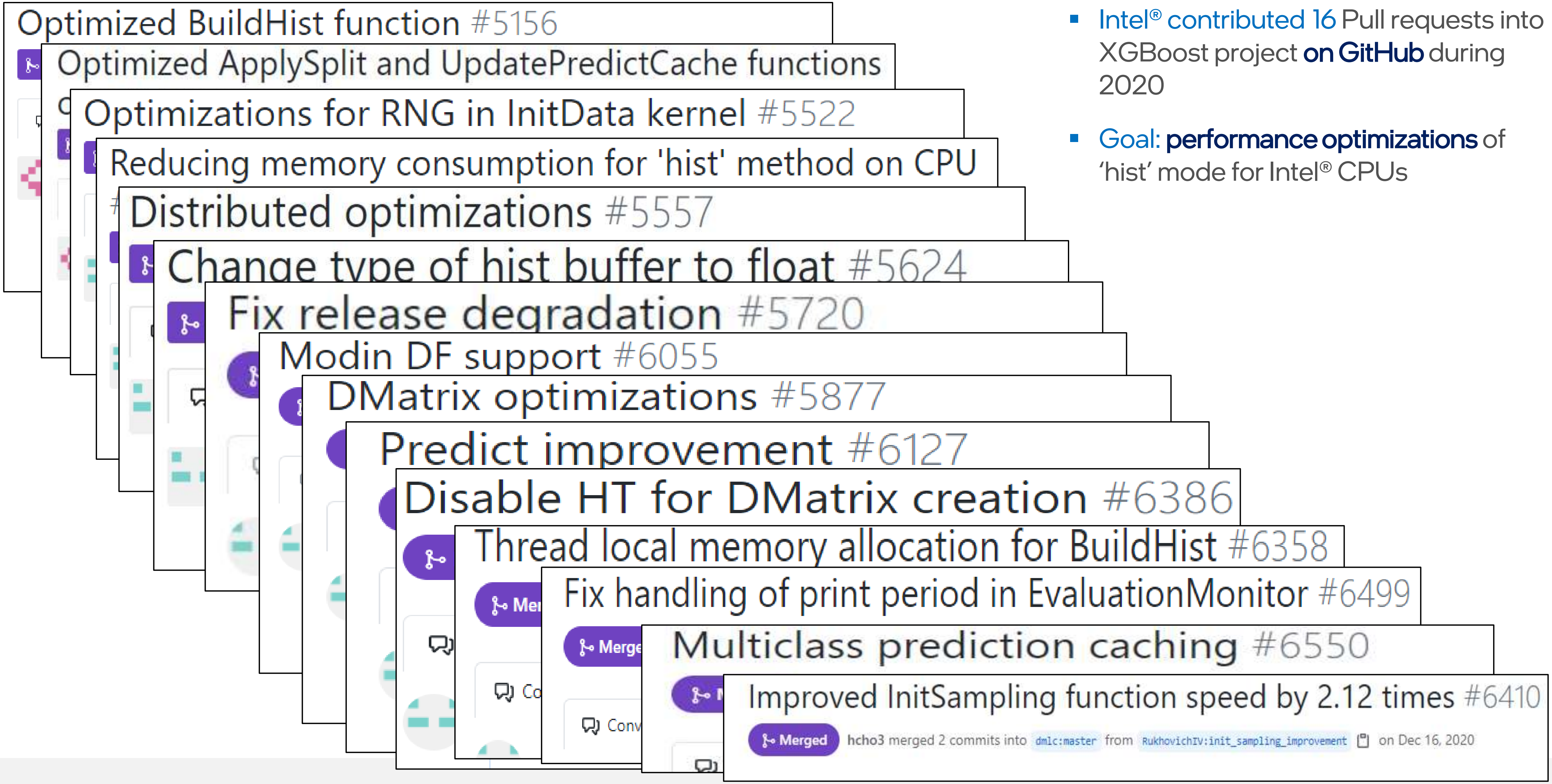
Testing Date: Performance results are based on testing by Intel as of March 21, 2023 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: bare metal (2.0 GHz Intel Xeon Platinum 8480+, two sockets, 56 cores per socket), 512 GB DDR5 4800MT/s, Python 3.10, scikit-learn 1.2.0, scikit-learn-intelex 2023.0.1. Intel optimizations include use of multi-threading implementation for SKLearn algorithms (which are typically single-threaded), as well as other HW/SW optimizations.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. Not product or component can be absolutely secure. Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

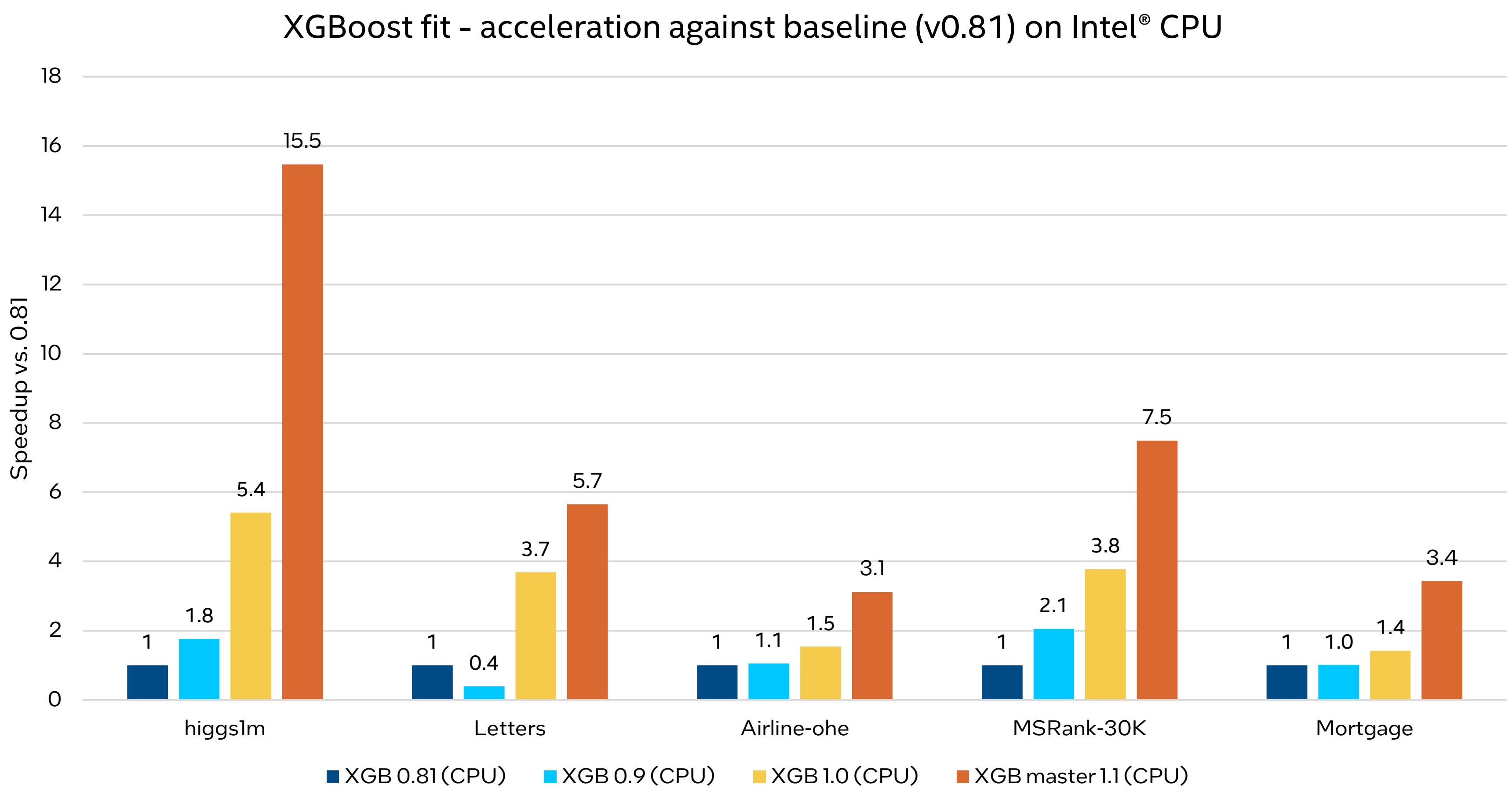
Intel Optimized XGBoost

DMLC XGBoost Acceleration



- Intel® contributed 16 Pull requests into XGBoost project **on GitHub** during 2020
- Goal: **performance optimizations** of 'hist' mode for Intel® CPUs

XGBoost* Fit CPU Acceleration ("hist" method)



+ Reducing memory consumption

Before	28311860	1907812
#5334	16218404	1155156
reduced:	1.75	1.65

CPU configuration: c5.24xlarge AWS Instance, CLX 8275 @ 3.0GHz, 2 sockets, 24 cores per socket, HT:on, DRAM (12 slots / 32GB / 2933 MHz)

Model Builders for the Gradient Boosting Frameworks

Conversion

The first step is to convert already trained model. The API usage for different frameworks is the same:

XGBoost:

```
import daal4py as d4p
d4p_model = d4p.mb.convert_model(xgb_model)
```

LightGBM:

```
import daal4py as d4p
d4p_model = d4p.mb.convert_model(lgb_model)
```

CatBoost:

```
import daal4py as d4p
d4p_model = d4p.mb.convert_model(cb_model)
```

Note

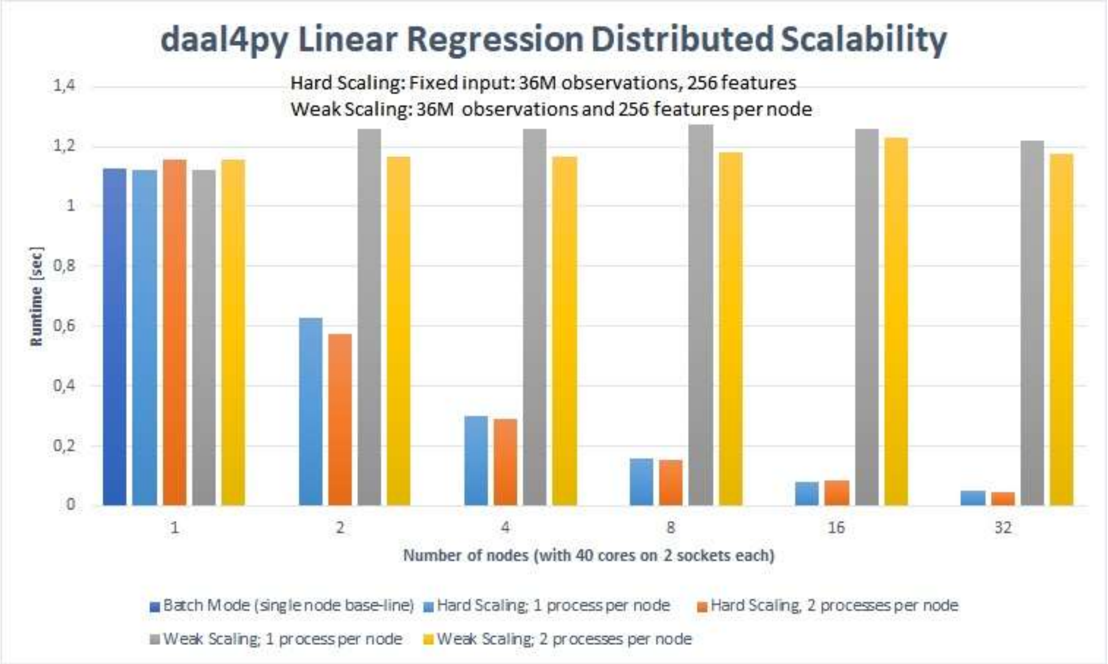
Convert model only once and then use it for the inference.

Source: https://intelpython.github.io/daal4py/model-builders.html

daal4py

Fast, Scalable and Easy Machine Learning With daal4py

- fast and easy to use
- provides highly configurable Machine Learning kernels, some of which support streaming input data and/or can be easily and efficiently scaled out to clusters of workstations
- it uses Intel(R) oneAPI Data Analytics Library to deliver the best performance
- Supported algorithms: https://intelpython.github.io/daal4py/algorithms.html



On a 32-node cluster (1280 cores) daal4py computed linear regression of 2.15 TB of data in 1.18 seconds and 68.66 GB of data in less than 48 milliseconds.

Configuration: 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, Oracle Linux Server release 7.4, using 64-bit floating point numbers

Installation and linear regression example

- Install from PyPI:

```
pip install daal4py
```

- Install from Anaconda Cloud: Conda-Forge channel:

```
conda install daal4py -c conda-forge
```

- Install from Anaconda Cloud: Intel channel:

```
conda install daal4py -c intel
```

```python
import daal4py as d4p
# train, test, and target are Pandas dataframes

d4p_lm = d4p.linear_regression_training(interceptFlag=True)
lm_trained = d4p_lm.compute(train, target)

lm_predictor_component = d4p.linear_regression_prediction()
result = lm_predictor_component.compute(test, lm_trained.model)
```

Full documentation: https://intelpython.github.io/daal4py/contents.html

Modin*

Single Line Code Change for Infinite Scalability

No need to learn a new API to use Modin*

```
import pandas as pd
```

- Accelerate your Pandas* workloads across multiple cores and multiple nodes
- No upfront cost to learning a new API
  - import modin.pandas as pd
- Integration with the Python* ecosystem
- Integration with Ray/Dask clusters (run on what you have, even on a laptop!)
- Installation instructions: https://modin.readthedocs.io/en/stable/getting_started/installation.html

Notices and Disclaimers

Performance varies by use, configuration, and other factors. Learn more at intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Your costs and results may vary.

Intel® technologies may require enabled hardware, software, or service activation.

Intel® optimizations, for Intel® compilers or other products, may not optimize to the same degree for non-Intel products.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Results have been estimated or simulated.

Intel is committed to respecting human rights and avoiding complicity in human rights abuses. See Intel's Global Human Rights Principles. Intel® products and software are intended only to be used in applications that do not cause or contribute to a violation of an internationally recognized human right.

Q&A

Configuration Details

(1) INTEL® XEON ™ BASED SYSTEM CONFIGURATION: Test by Intel as of 07/15/23 and may not reflect all publicly available security updates.. System: cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (ibench Linear Algebra), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

(2) INTEL® XEON ™ BASED SYSTEM CONFIGURATION: Test by Intel as of 03/05/23 and may not reflect all publicly available security updates. System: cloud.intel.com, nodes=1:spr:ppn=2, Intel(R) Xeon(R) Platinum 8480+, 2 sockets, 56 cores per socket, HT On, Intel Turbo Boost On, Total Memory 528GB, RAM 33 MHz, Ubuntu 20.04.5 LTS, 5.18.15-051815-generic, Microcode: 0x2b000310, benchmarks https://github.com/IntelPython/ (fft_benchmark, blackscholes_bench, composability_bench), -c conda-forge environment versions: numpy 1.23.5, scipy 1.10.1, numba 0.56.4 modules installed, -c intel environment versions: numpy 1.21.4, scipy 1.7.3, numba 0.56.3, tbb4py 2021.8.0 modules installed

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. Not product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary

Configuration Details

Testing Date: Performance results are based on testing by Intel® as of October 13, 2020 and may not reflect all publicly available updates.

Configurations details and Workload Setup: CPU: c5.18xlarge AWS Instance (2 x Intel® Xeon® Platinum 8124M @ 18 cores. OS: Ubuntu 20.04.2 LTS, 193 GB RAM. GPU: p3.2xlarge AWS Instance (GPU: NVIDIA Tesla V100 16GB, 8 vCPUs, OS: Ubuntu 18.04.2LTS, 61 GB RAM. SW: XGBoost 1.1: build from sources compiler – G++ 7.4, nvcc 9.1 Intel® DAAL: 2019.4 version: Python env: Python 3.6, Numpy 1.16.4, Pandas 0.25 Scikit-learn 0.21.2.

intel®