Numba Data parallel Python

Data Parallel Essentials for Python: Bringing oneAPI to python

Praveen Kundurthy

What is Data parallel Python?

intel®

Numba-Dpex

- Agenda
  - Overview of oneAPI
  - Overview of Intel® oneAPI AI Analytics Toolkit
  - Introduction to Numba-Data parallel extension (numba-dpex)
  - Introduction to Data Parallel Control (dpctl)
  - Device offloading using dpctl
  - Introduce to @njit decorator and @kernel decorator
- Hands On Intel® DevCloud / JLSE
  - Automatic offload using @njit
  - Explicit Parallel offload using @njit
  - Dpctl Demo
  - Compute follows data approach

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

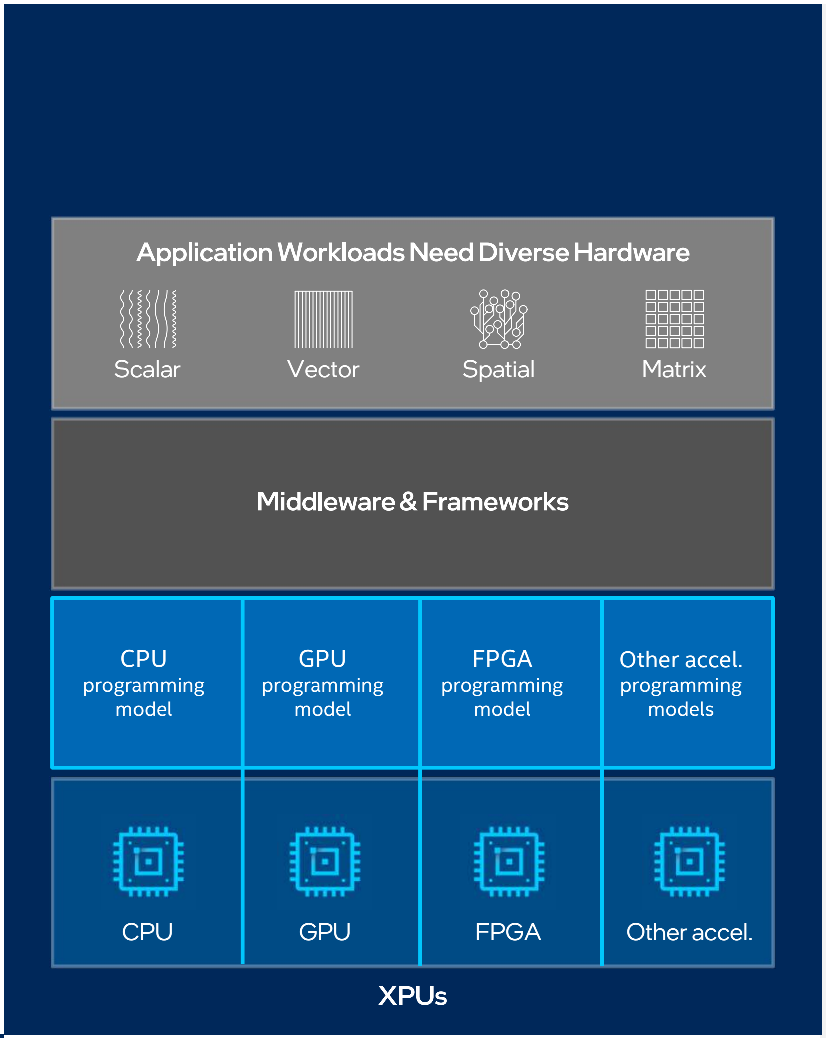
Programming Challenges
for Multiple Architectures

Growth in specialized workloads

Variety of data-centric hardware required

Separate programming models and toolchains for each architecture are required today

Software development complexity limits freedom of architectural choice



Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

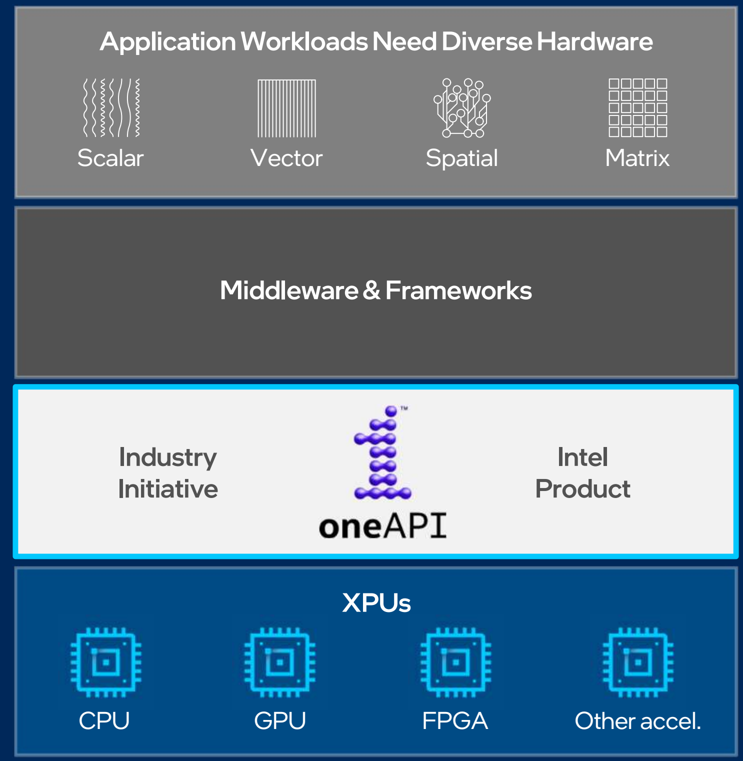
Introducing
oneAPI

Cross-architecture programming that delivers freedom to choose the best hardware

Based on industry standards and open specifications

Exposes cutting-edge performance features of latest hardware

Compatible with existing high-performance languages and programming models including C++, OpenMP, Fortran, and MPI



Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel® oneAPI AI Analytics Toolkit

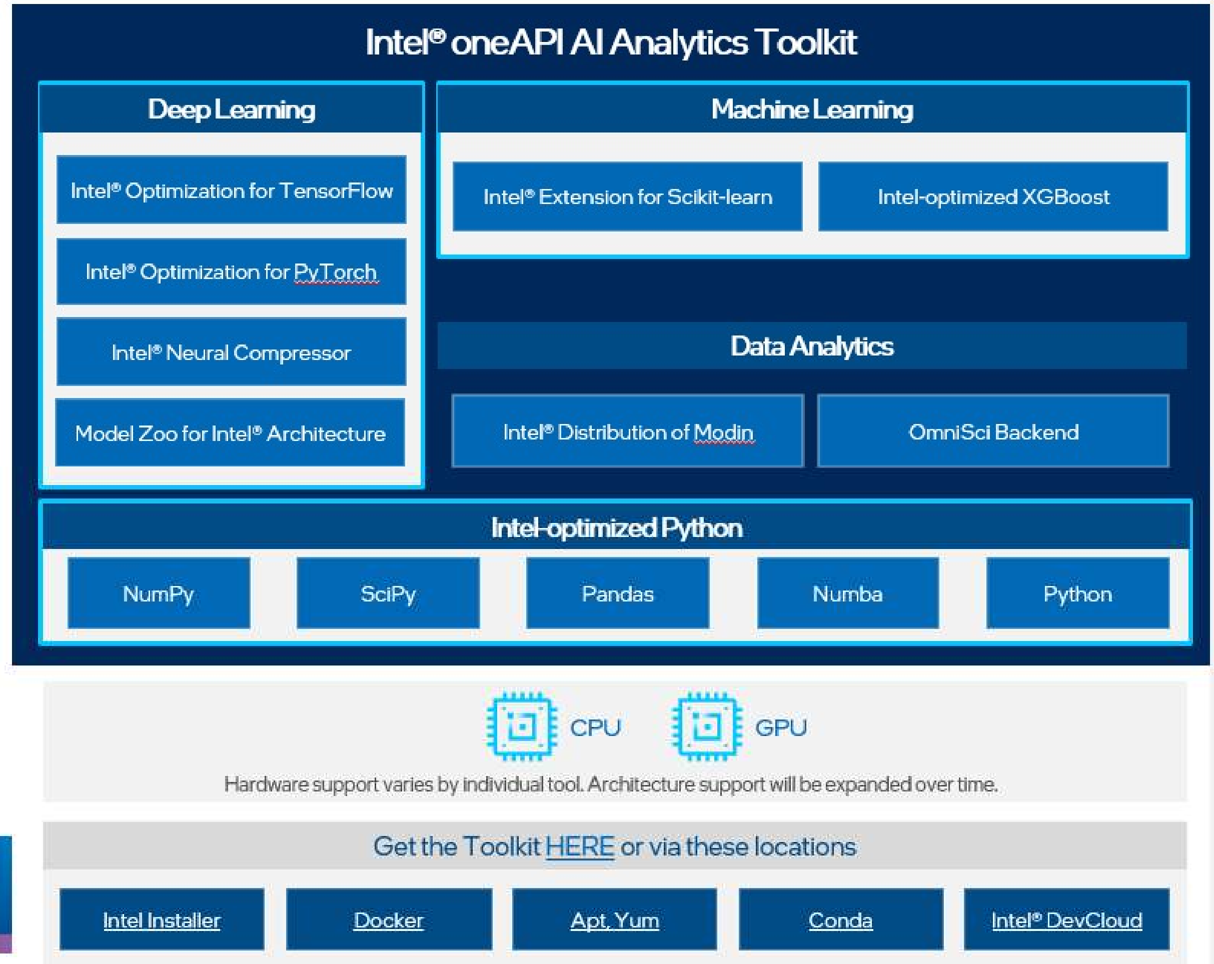
Accelerate end-to-end AI and data analytics pipelines with libraries optimized for Intel® architectures

Who Uses It?

Data scientists, AI researchers, ML and DL developers, AI application developers

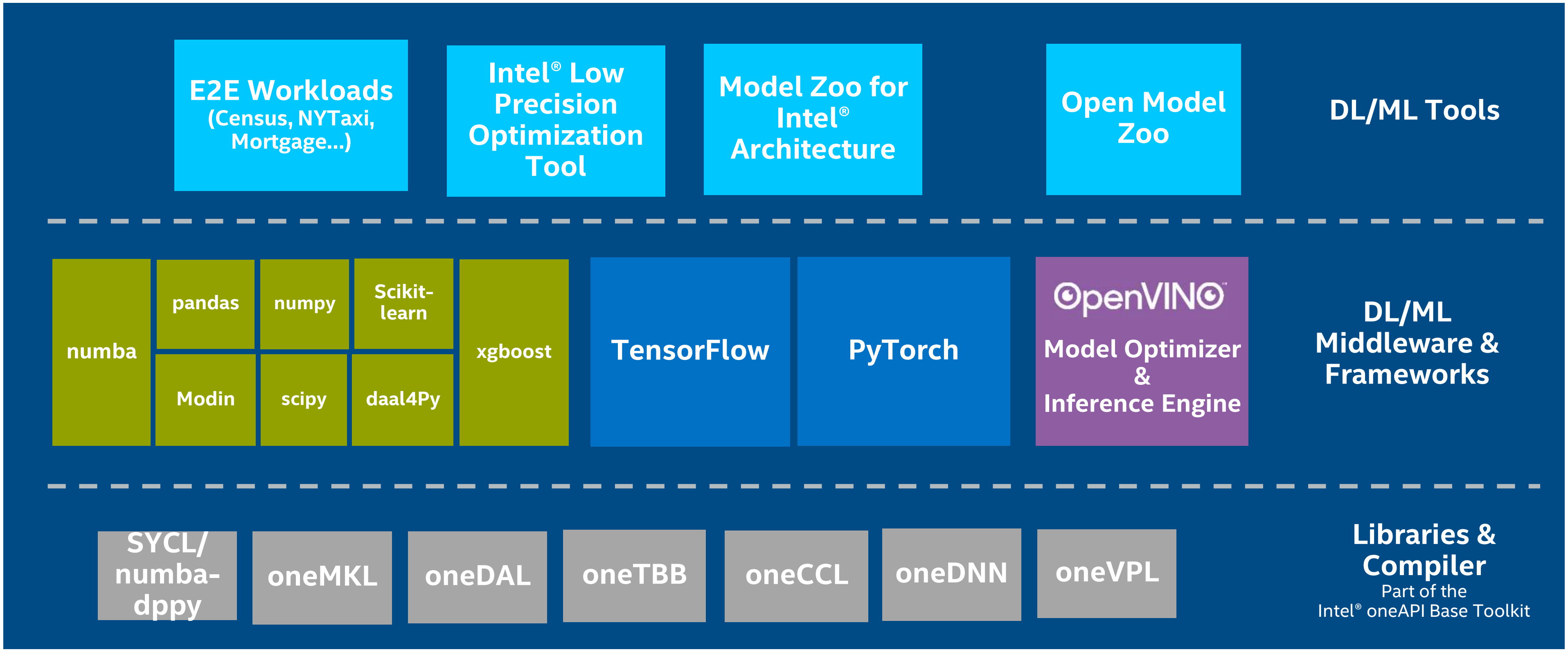
Top Features/Benefits

- Deep learning performance for training and inference with Intel optimized DL frameworks and tools
- Drop-in acceleration for data analytics and machine learning workflows with compute-intensive Python packages



AI Software Stack for Intel XPUs

Intel offers a Robust Software Stack to Maximize Performance of Diverse Workloads

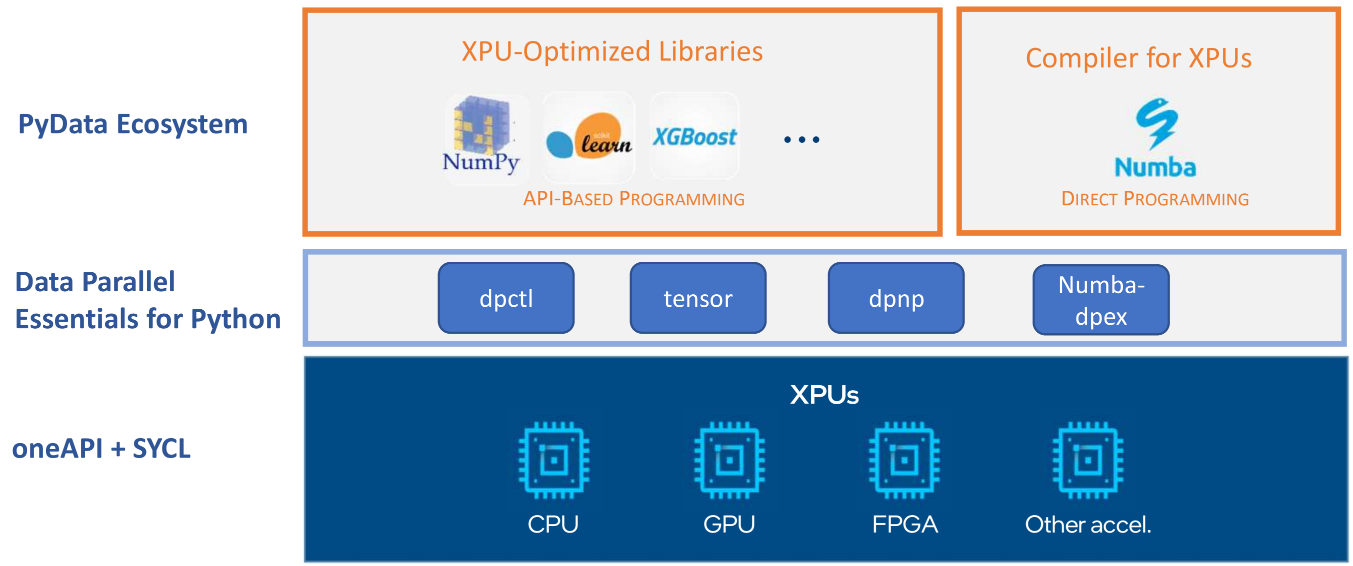


Optimization Notice

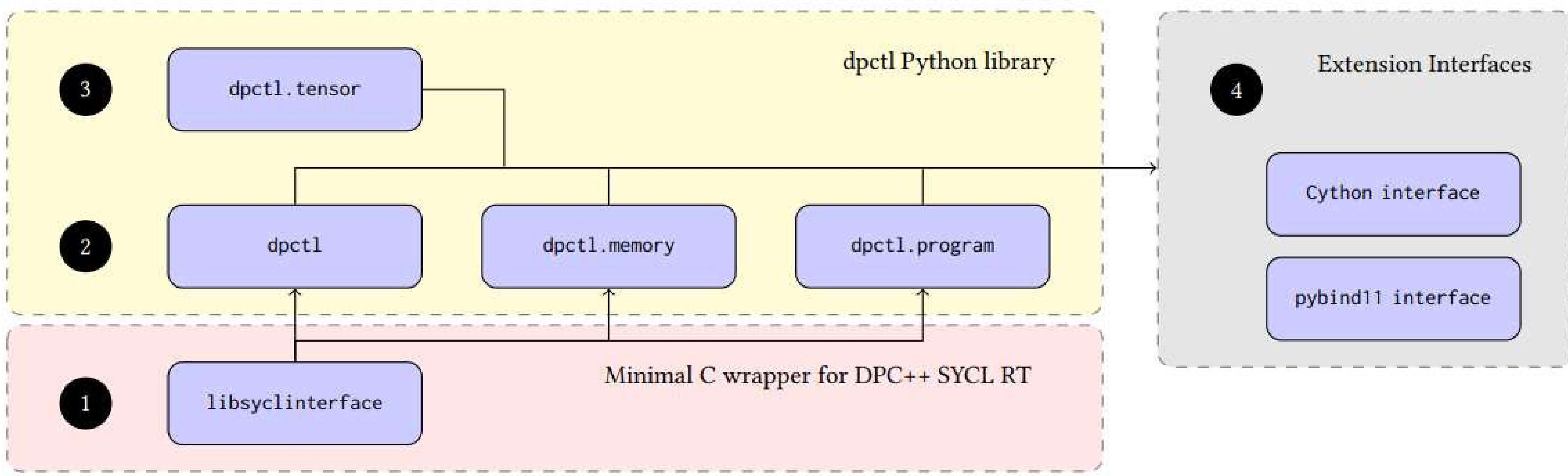
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Data Parallel Essentials for Python

Fostering a oneAPI/SYCL-based ecosystem for PyDATA

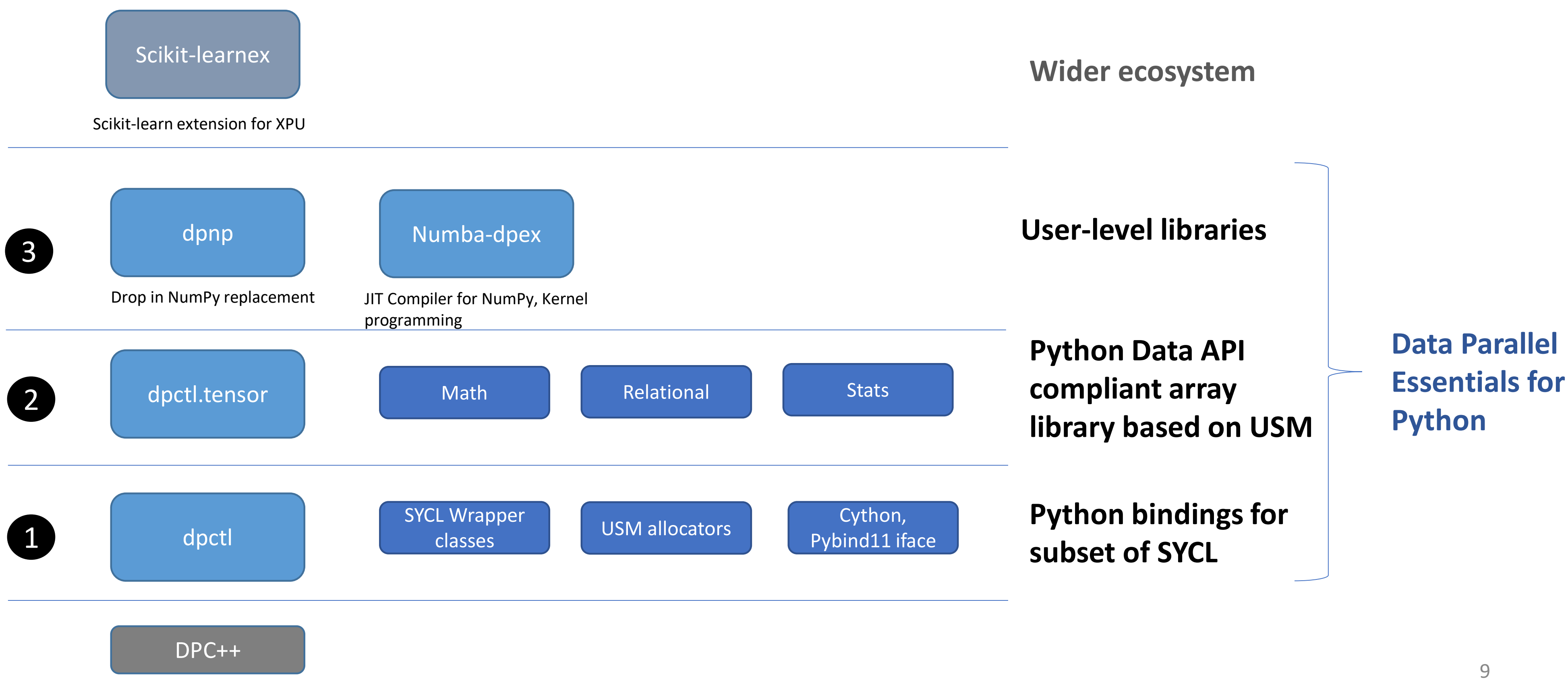


dpctl – Data parallel control



1. Library providing a minimal C API for the main DPC++ SYCL runtime classes
2. Python modules exposing SYCL runtime classes, USM allocators, and kernel bundle
3. A data API standard complaint array library supporting USM allocated memory
4. Native API to use dpctl objects in Cython and pybind11 extensions modules

Current Ecosystem



Compute Follows data

Offload Model

- Pythonic offload model following array API spec (https://data-apis.org/array-api/latest/)
- Offload happens where data currently resides ("compute follows data")

```
X = dp.array([1,2,3])
Y = X * 4
```

executed on default device

```
X = dp.array([1,2,3], device="gpu:0")
Y = X * 4
```

executed on "gpu:0" device

```
X = dp.array([1,2,3], device="gpu:0")
Y = dp.array([1,2,3], device="gpu:1")
Z = X + Y
```

Error! Arrays are on different devices

Programming Model

Compute Follows Data

- Pythonic offload model following array API spec
- Explicit control over execution based on data placement

```
import dpnp as dp
    # Case 1
    #  Allocate X on the default device
    X = dp.array([1,2,3])
    #  scaling of X executed on device of X, result
         placed into Y
    Y = X * 4
    # Case 2
    #  Allocate X on "gpu:1"
    X = dp.array([1,2,3], device="gpu:1")
    # Executed on "gpu:1"
    Y = X * 4
    # Case 3
    X1 = dp.array([1,2,3], device="gpu:1")
    X2 = dp.array([1,2,3], device="gpu:0")
    #  error!
    Y = X1 + X2

    # Arrays can be associated with another device
    # (copy is performed if needed)
    X1a = X1.to_divice(device=dev)
```

Numba-dpex

- Numba is a Just-in-time compiler for Python for NumPy arrays functions, and loops to speed up your applications written directly in Python.
- Numba automatically offloads specific data-parallel sections of a Numba jit function.
- Numba-dpex is a standalone extension to the Numba JIT compiler that adds SYCL programming capabilities to Numba

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

dpctl SyclDevice

- A device represents a specific accelerator in the system.
- Creating a queue for a specific device requires a device_selector.
- This is a python equivalent for cl::sycl::device class

Import dpctl and print device information

Call Sycl Device of type GPU

```
import dpctl

def print_device(d):
    "Display information about given device argument."
    print("Name: ", d.name)
    print("Vendor: ", d.vendor)
    print("Driver version: ", d.driver_version)
    print("Backend: ", d.backend)
    print("Max EU: ", d.max_compute_units)

# Create a SyclDevice of type GPU based on whatever is returned
# by the SYCL `gpu_selector` device selector class.
# d = dpctl.select_cpu_device()
# d = dpctl.select_accelerator_device()
# d = dpctl.select_host_device()
# d = dpctl.select_default_device()
d = dpctl.select_gpu_device():
#d.print_device_info()
print_device(d)
```


Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Numba-dpex

oneAPI

Array-style programming

```
@njit(parallel=True)
def l2_distance(a, b, c)
    return np.sum((a-b)**2)
```

NumPy (array) style programming. Requires minimum code changes to compile existing Numpy code for XPU.

Explicit prange (parfor) loops

```
@njit(parallel=True)
def l2_distance(a, b, c)
    s = 0.0
    for i in prange(len(a))
        s += (a[i]-b[i])**2
    return s
```

Parfor-style programming. Preferred by some users when iteration space requires complex indexing.
Unique for CPU. Intel extends to XPU via numba-dpex. No CUDA alternatives to date

OpenCl-style kernel programming

```
@kernel(access_type={"read_only": ["a", "b"], write_only:["c"]})
def l2_distance(a, b, c)
    i = numba_dpex.get_global_id(0)
    j = numba_dpex.get_global_id(1)
    sub = a[i,j] - b[i,j]
    sq = sub ** 2
    atomic.add(c, 0, sq)
```

Most advanced programming model. Recommended to get highest performance on XPU yet avoiding DPC++.
Nvidia @cuda.jit offers this programming model in Numba

Automatic offload using @njit Decorator

Import njit and prange from numba

Use @njit decorator to directly detect data parallel kernels using numpy expressions

Automatic offload mode for NumPy data-parallel expressions

Use dpctl.device context to offload this to a device

```
import dpctl
import numpy as np
import numba

@numba.njit(parallel=True)
def l2_distance_kernel(a, b):
    sub = a - b
    sq = np.square(sub)
    sum = np.sum(sq)
    d = np.sqrt(sum)
    return d
def main():
    R = 64
    C = 1
    X = np.random.random((R,C))
    Y = np.random.random((R,C))
    device = dpctl.select_default_device()
    print("Using device ...")
    device.print_device_info()
    with dpctl.device_context(device):
        result = l2_distance_kernel(X, Y)
    print("Result :", result)
    print("Done...")
if __name__ == "__main__":
    main()
```


Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Explicit parallel for loop - @njit Decorator

Import njit and prange from numba

Use @njit decorator to directly detect data parallel kernels using numpy expressions

Use prange to specify explicitly a loop to be parallelized

Use dpctl.device context to offload this to a device

```
import numpy as np
from numba import njit, prange
import dpctl

@njit
def add_two_arrays(b, c):
    a = np.empty_like(b)
    for i in prange(len(b)):
        a[i] = b[i] + c[i]
    return a

def main():
    N = 10
    b = np.ones(N)
    c = np.ones(N)
    device = dpctl.select_default_device()
    with dpctl.device_context(device):
        result = add_two_arrays(b, c)

if __name__ == "__main__":
    main()
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

@dppy.kernel Decorator

Import dpctl

Vector addition in parallel using the @ddpy.kernel decorator

Common way of Kernel invocation

Offload this to a device

```
import dpctl
import numba_dppy as dppy
import numpy as np

@dppy.kernel
def data_parallel_sum(a, b, c):
    i = dppy.get_global_id(0)
    c[i] = a[i] + b[i]

def driver(a, b, c, global_size):
    data_parallel_sum[global_size, dppy.DEFAULT_LOCAL_SIZE
](a, b, c)
    print("C ", c)

def main():
    global_size = 10
    N = global_size
    print("N", N)
    a = np.array(np.random.random(N), dtype=np.float32)
    b = np.array(np.random.random(N), dtype=np.float32)
    c = np.ones_like(a)
    with dpctl.device_context("opencl:gpu"):
      driver(a, b, c, global_size)

if __name__ == "__main__":
    main()
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Pairwise distance using @dppy.kernel

Import dpctl

Pairwise distance in parallel using the @ddpy.kernel decorator

Kernel invocation of the Pairwise distance

Offload this to opencl:gpu

```
import dpctl
import numpy as np
import numba_dppy

@numba_dppy.kernel
def pairwise_python(X1, X2, D):
    i = numba_dppy.get_global_id(0)

    N = X2.shape[0]
    O = X1.shape[1]
    for j in range(N):
        d = 0.0
        for k in range(O):
            tmp = X1[i, k] - X2[j, k]
            d += tmp * tmp
        D[i, j] = np.sqrt(d)

def pw_distance(X1, X2, D):
    with dpctl.device_context("opencl:gpu"):
        # pairwise_python[X1.shape[0],numba_dppy.DEFAULT_L
OCAL_SIZE](X1, X2, D)
        pairwise_python[X1.shape[0], 128](X1, X2, D)
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Kmeans using @njit

Import dpctl

Kmeans in parallel using the @njit decorator. Determine the Euclidean distance from the cluster center to each point

Assign points to cluster and update the centroids array after computation

Offload this to opencl:gpu

Parallel for loops using numba.prange

```
import dpctl
import numpy
import numba
@numba.jit(nopython=True, parallel=True, fastmath=True)
def groupByCluster(arrayP, arrayPcluster, arrayC, num_points, num_centroids):
    for i0 in numba.prange(num_points):
        minor_distance = -1
        for i1 in range(num_centroids):
            dx = arrayP[i0, 0] - arrayC[i1, 0]
            dy = arrayP[i0, 1] - arrayC[i1, 1]
            my_distance = numpy.sqrt(dx * dx + dy * dy)
            if minor_distance > my_distance or minor_distance == -1:
                minor_distance = my_distance
                arrayPcluster[i0] = i1
    return arrayPcluster
@numba.jit(nopython=True, parallel=True, fastmath=True)
def calCentroidsSum(
    arrayP, arrayPcluster, arrayCsum, arrayCnumpoint, num_points, num_centroids
):
    for i in numba.prange(num_centroids):
        arrayCsum[i, 0] = 0
        arrayCsum[i, 1] = 0
        arrayCnumpoint[i] = 0

    for i in range(num_points):
        ci = arrayPcluster[i]
        arrayCsum[ci, 0] += arrayP[i, 0]
        arrayCsum[ci, 1] += arrayP[i, 1]
        arrayCnumpoint[ci] += 1
    return arrayCsum, arrayCnumpoint
@numba.jit(nopython=True, parallel=True, fastmath=True)
def updateCentroids(arrayC, arrayCsum, arrayCnumpoint, num_centroids):
    for i in numba.prange(num_centroids):
        arrayC[i, 0] = arrayCsum[i, 0] / arrayCnumpoint[i]
        arrayC[i, 1] = arrayCsum[i, 1] / arrayCnumpoint[i]
def kmeans(
    arrayP, arrayPcluster, arrayC, arrayCsum, arrayCnumpoint, num_points, num_centroids
):
    for i in range(ITERATIONS):
        with dpctl.device_context(base_kmeans_gpu.get_device_selector()):
            groupByCluster(arrayP, arrayPcluster, arrayC, num_points, num_centroids)
        calCentroidsSum(
            arrayP, arrayPcluster, arrayCsum, arrayCnumpoint, num_points, num_centroids)
        updateCentroids(arrayC, arrayCsum, arrayCnumpoint, num_centroids)
    return arrayC, arrayCsum, arrayCnumpoint
def run_kmeans(
    arrayP,arrayPclusters,arrayC,arrayCsum,arrayCnumpoint,NUMBER_OF_POINTS,NUMBER_OF_CENTROIDS,):
    for i in range(REPEAT):
        for i1 in range(NUMBER_OF_CENTROIDS):
            arrayC[i1, 0] = arrayP[i1, 0]
            arrayC[i1, 1] = arrayP[i1, 1]
        arrayC, arrayCsum, arrayCnumpoint = kmeans(
            arrayP,arrayPclusters,arrayC,arrayCsum,arrayCnumpoint,NUMBER_OF_POINTS,NUMBER_OF_CENTROIDS,
)
```


Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Black Scholes using @njit

Import dpctl

Black Scholes in parallel using the @njit decorator

Calculate Calls and puts with the change in the current price and the strike price

Offload this to level_zero:gpu

```
import dpctl
import numba as nb
from math import log, sqrt, exp, erf

# blackscholes implemented as a parallel loop using numba.prange
@nb.njit(parallel=True, fastmath=True)
def black_scholes_kernel(nopt, price, strike, t, rate, vol, call, put):
    mr = -rate
    sig_sig_two = vol * vol * 2
    for i in nb.prange(nopt):
        P = price[i]
        S = strike[i]
        T = t[i]
        a = log(P / S)
        b = T * mr
        z = T * sig_sig_two
        c = 0.25 * z
        y = 1.0 / sqrt(z)
        w1 = (a - b + c) * y
        w2 = (a - b - c) * y
        d1 = 0.5 + 0.5 * erf(w1)
        d2 = 0.5 + 0.5 * erf(w2)
        Se = exp(b) * S
        r = P * d1 - Se * d2
        call[i] = r
        put[i] = r - P + Se

def black_scholes(nopt, price, strike, t, rate, vol, call, put):
    # offload blackscholes computation to GPU (toggle level0 or opencl driv
er).
    with dpctl.device_context("level_zero:gpu"):
        black_scholes_kernel(nopt, price, strike, t, rate, vol, call, put)
```

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Learn More at the Intel® DevCloud for oneAPI

Free Access, A Fast Way to Start Coding

A development sandbox to develop, test and run workloads across a range of Intel® CPUs, GPUs, and FPGAs using Intel's oneAPI software

For customers focused on data-centric workloads on a variety of Intel® architecture

- Learn Data Parallel C++
- Use Intel® oneAPI Toolkits
- Evaluate Workloads
- Prototype Your Project
- Build Cross-architecture Applications

No Downloads | No Hardware Acquisition | No Installation | No Set-up & Configuration

Get Up & Running in Seconds!

https://devcloud.intel.com/oneapi/get_started/

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Running Numba_DPPY Essentials on JLSE CLI

1) qsub -n1 -t 180 -q iris –I
2) module use /soft/restricted/CNDA/modulefiles
3) module add oneapi
4) source $IDPROOT/bin/activate
5) conda create -n <NEW_ENV> --clone $AURORA_BASE_ENV
6) conda activate <NEW_ENV>
7) conda install packaging
8) export SYCL_DEVICE_FILTER=opencl
9) git clone https://github.com/IntelSoftware/Numba_DPPY_Essentials.git
10) Navigate to AI-and-Analytics/Jupyter/Numba_DPPY_Essentials_training

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Hands-on Coding on Intel® DevCloud / JLSE

1) Included with the oneAPI install is the Intel Distribution of Python. The base environment does not have Jupyter lab included so it will be necessary to create a custom python environment.

   a. **`source $IDPROOT/bin/activate`**

   b. Now create a custom environment by cloning your local environment

      i. conda create -n <NEW_ENV> --clone $AURORA_BASE_ENV(takes a few mins)

   c. To activate the new environment

      i. conda activate <NEW_ENV>

      ii. conda install packaging (install needed packaging)

      iii. export SYCL_DEVICE_FILTER=opencl (set this env variable for the samples to work for opencl gpu driver)

      iv. Now you can run the samples in the CLI

   d. Optional: Now install Jupyterlab (if you want to try running the samples from the Jupyter folder)

      i. conda install -c conda-forge jupyterlab

   e. Some of the modules use Ipywidgets (optional)

      i. conda install -c conda-forge ipywidgets

2) Launch Jupyter lab

   a. Navigate to where you cloned the oneAPI samples repo.

   b. **`Enter: Jupyter-lab --no-browser --port=<default is 8888, randomize this>`**

   c. Make note of the addresses printed to your terminal

Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

1) **Important:** From a different local ssh session, separate from the one you used to obtain the iris node tunnel directly to your iris node. It is assumed that an iris node has been allocated by a user.
   a. **It will look like: `username@iris<#>`**
   b. **example: `ssh -v -J jlse -L 8989:localhost:8989 username@iris11`**
      i. **Note:** `the ports need to be free on your local machine`
2) You will need to copy the token provided by jupyter lab from your initial ssh session and paste that into your browser.
   a. Open local browser and enter, example: **`http://localhost:8989/lab?token=8135de98c....`**
3) Navigate to **`Numba_DPPY_Essentials`** and double click on **`Welcome.ipynb`** to get started.

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Summary

- Illustrate How oneAPI Can help solve the challenges of programming in a heterogeneous world
- How to use Data Parallel Python and Data Parallel Control
- Performed 3 code walkthroughs demonstrating:
  - A Pairwise Algorithm using Jit and Kernel decorators on CPU and GPU
  - A Kmeans Algorithm using Jit and Kernel decorators on CPU and GPU
  - A Gpairs Algorithm using Jit and Kernel decorators on CPU and GPU
- Explored via hands on activities the following algorithms in depth
  - Pairwise Algorithm
  - Kmeans Algorithm
  - Gpairs Algorithm

Optimization Notice
Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Thanks for attending the session

NOTICES &

- This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.
- The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request. No product or component can be absolutely secure. Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.
- 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.

- Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and OpenVINO are trademarks of Intel Corporation or its subsidiaries 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

Copyright © 2019, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.