

Nikita Grigorian

# Dpctl Portable Data Parallel Extensions for Python



# Dpctl (Data Parallel Control) Overview

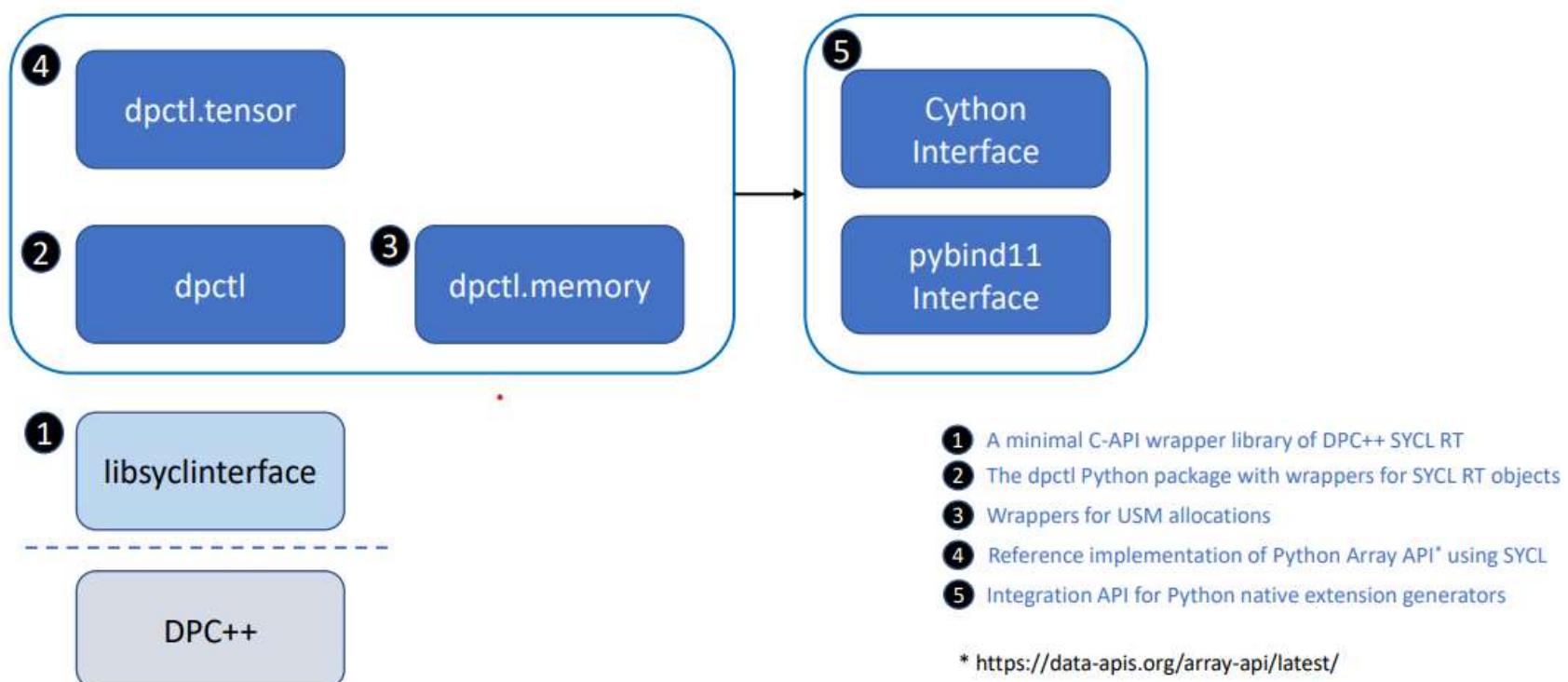
- Python package for supporting and performing parallel compute, following the SYCL spec
- Python interface to SYCL runtime objects provided by DPC++ / Intel LLVM
- Bridges Python applications with device-aware native extensions for high performance compute



# Dpctl (Data Parallel Control) Overview

- Enables writing extensions via pybind11 and Cython interfaces which can integrate with native code and external libraries
- An example use: Python script uses dpctl to allocate memory and create a SYCL queue, then passes those objects to a compiled SYCL/C++ shared library for compute execution, via pybind11 or Cython interfaces

# Dpctl Architecture



# Dpctl Namespace

Python wrappers for SYCL runtime objects and functions

```
In [1]: import dpctl
```

```
In [2]: q = dpctl.SyclQueue()
```

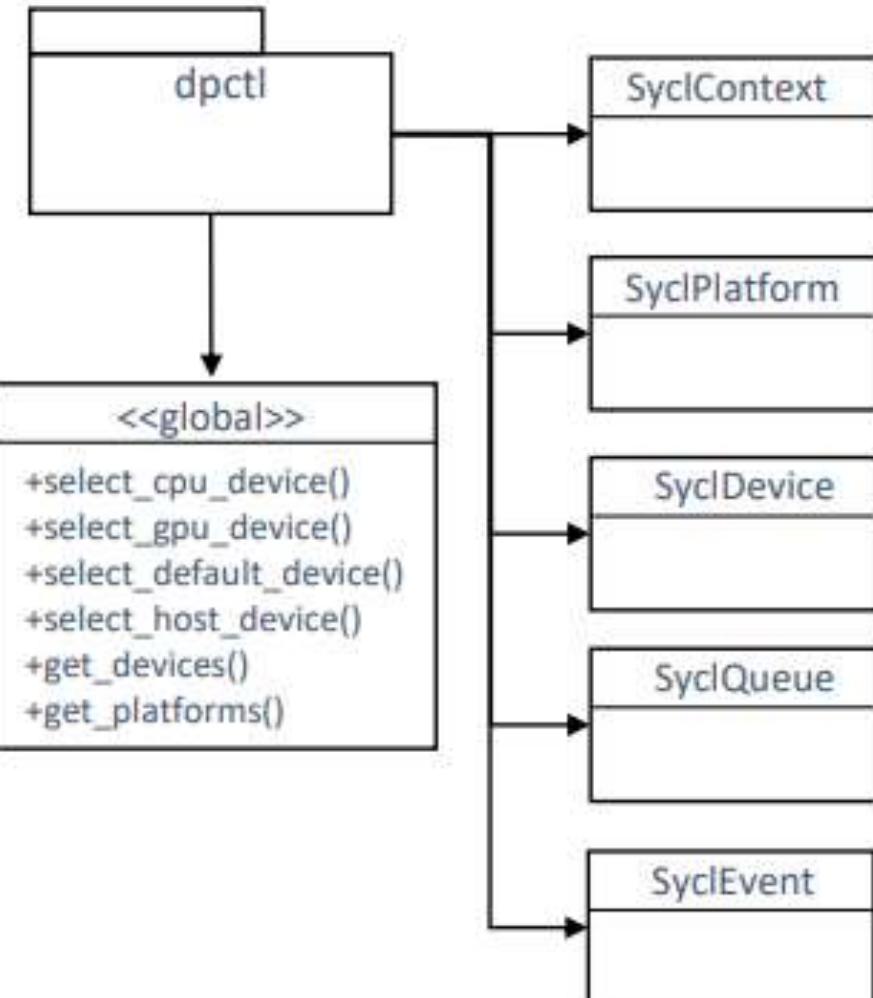
```
In [3]: q.sycl_device
```

```
Out[3]: <dpctl.SyclDevice [backend_type.level_zero,  
device_type.gpu, Intel(R) Graphics [0x9a49]] at 0x7f0ef57a6af0>
```

```
In [4]: q = dpctl.SyclQueue("opencl:cpu",  
property="enable_profiling")
```

```
In [5]: q.has_enable_profiling
```

```
Out[5]: True
```



# Dpctl Submodules

dpctl.tensor

```
In [1]: import dpctl
```

```
In [2]: import dpctl.tensor as dpt
```

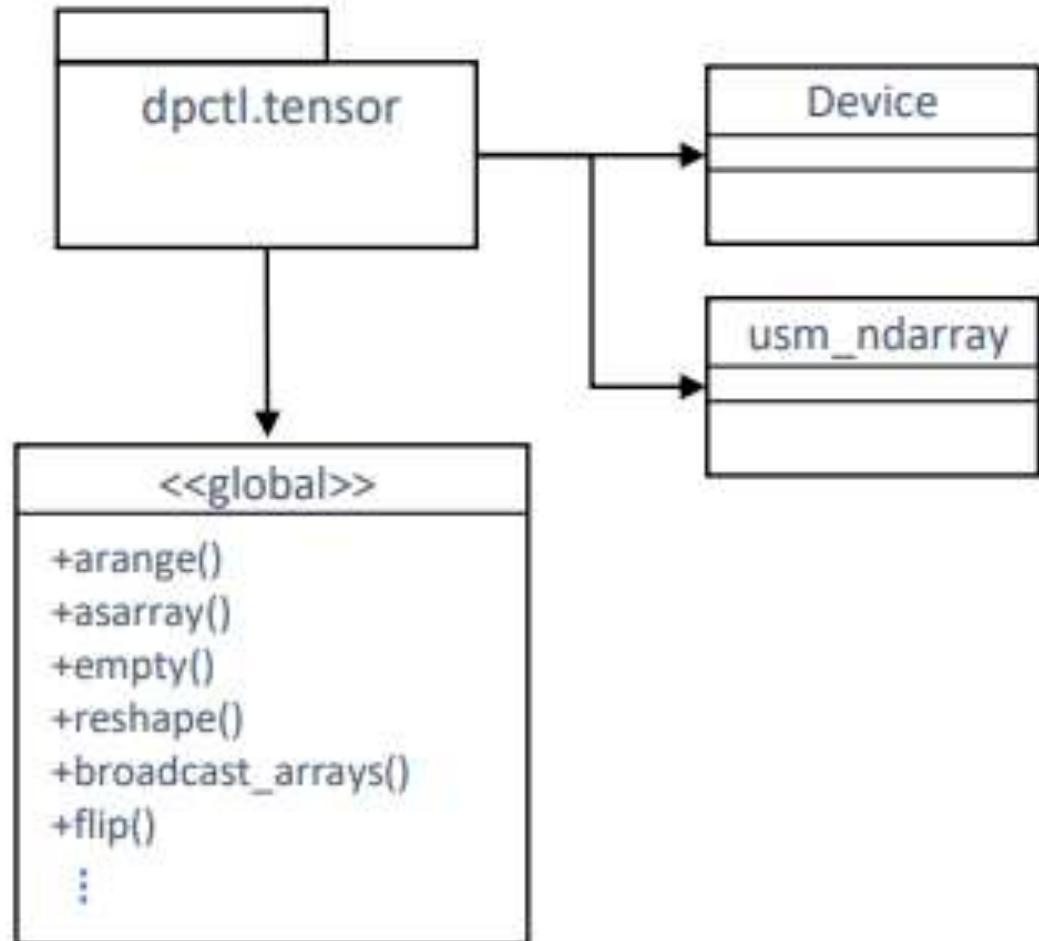
```
In [3]: q = dpctl.SyclQueue()
```

```
In [4]: x = dpt.arange(10)
```

```
In [5]: x = dpt.arange(10, usm_type="device", sycl_queue=q)
```

```
In [6]: y = dpt.arange(10, usm_type="host", device="cpu")
```

```
In [7]: x + y # fails, different devices/queues
```



# Dpctl Submodules

## dpctl.memory

```
In [1]: import dpctl
```

```
In [2]: import dpctl.memory as dpm
```

```
In [3]: q = dpctl.SyclQueue()
```

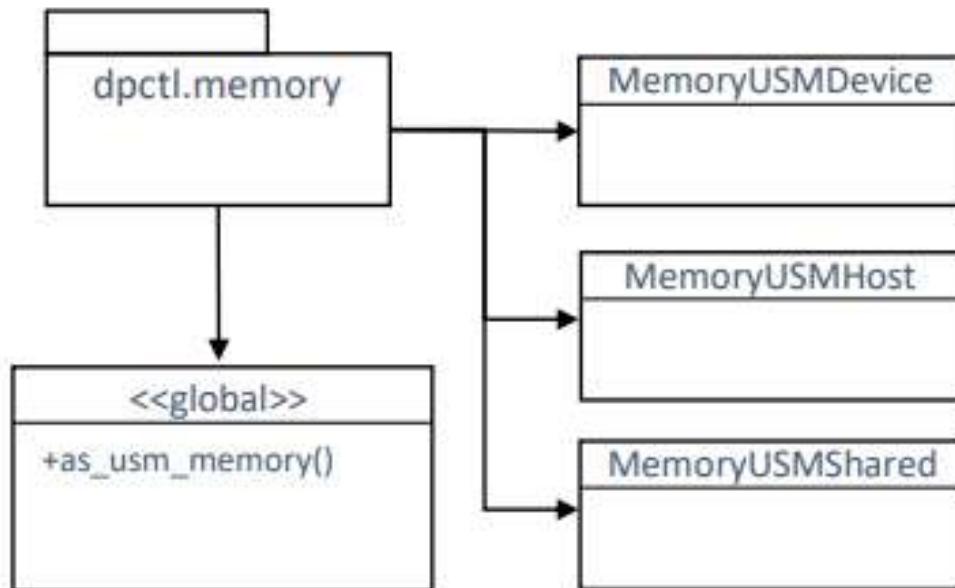
```
In [4]: x = dpm.MemoryUSMHost(256, alignment=32, queue=q)
```

```
In [5]: y = dpm.MemoryUSMDevice(256, alignment=32, queue=q)
```

```
In [6]: memoryview(x)
```

```
Out[6]: <memory at 0x7854a5d29480>
```

```
In [7]: memoryview(y) # fails, y is device-accessible only
```



# QR Extension Example

- Using Intel's oneMath library and dpctl, we can write a platform-portable, data-parallel extension for computing a QR decomposition
- As oneMath is also platform-portable, we can run this on CPU, Intel GPU, CUDA GPU, or AMD GPU
- See [portable data parallel extensions presented at EuroPython 2025](#)

```
In [1]: import dpctl, dpctl.tensor as dpt

In [2]: # use oneMath extension

In [3]: import onemath_ext as me

In [4]: # show available devices

In [5]: dpctl.lsplatform(verbosity=0)
Intel(R) FPGA Emulation Platform for OpenCL(TM) OpenCL 1
Intel(R) FPGA SDK for OpenCL(TM), Version 20.3
Intel(R) OpenCL OpenCL 3.0 LINUX
NVIDIA CUDA BACKEND CUDA 12.2

In [6]: # choose with an appropriate selector string

In [7]: x = dpt.eye(400, dtype=dpt.float32,
device="cuda:gpu")

In [8]: q, r = me.qr(x)

In [9]: # show devices of output arrays

In [10]: q.device, r.device
Out[10]: (Device(cuda:gpu:0), Device(cuda:gpu:0))
```

# Submit Custom Kernel Example

From dpctl example [use\\_dpctl\\_sycl\\_kernel](#), for an example of calling an arbitrary kernel (in this case, for doubling a vector)

```
void submit_custom_kernel(sycl::queue &q,           // from dpctl.SyclQueue
                         sycl::kernel &krn,        // from dpctl.program.SyclKernel
                         dpctl::tensor::usm_ndarray x, // from dpctl.tensor
                         dpctl::tensor::usm_ndarray y, // from dpctl.tensor
                         const std::vector<sycl::event> &depends = {}) // from list[dpctl.SyclEvent]
```

Then from Python we can call the Python binding for this function

```
submit_custom_kernel(
    q,      # dpctl.SyclQueue
    krn, # dpctl.program.SyclKernel
    x, # dpt.usm_ndarray
    y, # dpt.usm_ndarray
    [], # empty list
)
```

# Limitations and Ongoing Work

- No support for free-threaded Python
- No support for `sycl::buffer`, just USM
- No support for non-DPC++ compilers (AdaptiveCpp)
- Moving tensor to sister project, dpnp

Work and research is ongoing for all of the above

# Install dpctl for yourself!

- Dpctl is available on Intel's conda channel

```
conda install -c https://software.repos.intel.com/python/conda/ -c conda-forge --override-channels dpctl
```

- Available on PyPi as well

```
python -m pip install dpctl
```

Ongoing work to bring dpctl to conda-forge

# Links

[Dpctl Github](#)

[use\\_sycl\\_kernel Example](#)

[KDE/oneMath Example](#)