Interfacing Python and SYCL for XPU Programming

Diptorup Deb (presenter)
Oleksandr Pavlyk
Why SYCL matters for Python developers... ... and vice versa

https://spectrum.ieee.org/top-programming-languages-2021
So basically, our goal is to empower the Python ninja to chop through the overgrown landscape of device programming!
Interfacing SYCL and Python

• **Black Box** – Call a SYCL library like any other native library using Python C API.

• **Holistic** – Allow SYCL device and queue management, memory management, synchronization directly from Python.
**Technical Design Goals**

<table>
<thead>
<tr>
<th>Bring the SYCL programming model to Python</th>
<th>Simplify using SYCL API-based programming in Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Dynamic device selection</td>
<td>• Infrastructure to build Python native extensions wrapping SYCL libraries</td>
</tr>
<tr>
<td>• Sub-devices</td>
<td>• Navigate the language boundary between SYCL and Python</td>
</tr>
<tr>
<td>• Async kernel execution</td>
<td></td>
</tr>
<tr>
<td>• Device memory management</td>
<td></td>
</tr>
</tbody>
</table>
Data Parallel Control (dpctl)

1. A minimal C-API wrapper library of DPC++ SYCL RT
2. The dpctl Python package with wrappers for SYCL RT objects
3. Wrappers for USM allocations
4. Reference implementation of Python Array API* using SYCL
5. Integration API for Python native extension generators

* https://data-apis.org/array-api/latest/
dpctl: SYCL Bindings

- **SyclContext**
- **SyclPlatform**
- **SyclDevice**
- **SyclQueue**
- **SyclEvent**

**Custom Device Selection**

```
import dpctl

def select_device():
    """
    Programatically try to select a CUDA GPU device
    or else select a CPU device.
    """
    D = None
cuda_gpus = dpctl.get_devices(backend="cuda")
    if not cuda_gpus:
        D = select_cpu_device()
    else:
        D = cuda_gpus[0]
    return D
```

**Support for SYCL’s Filter Selector Extension**

```
import dpctl;

def subdivide_by_affinity(affinity=" numa"):
    """
    Create sub-devices partitioning by affinity. NUMA affinity
    is selected by default.
    """
    cpu_d = dpctl.SyclDevice("cpu")
    try:
        sub_devs = cpu_d.create_sub_devices(partition=affinity)
        print("Partitioned device using ", affinity) 
        print("(8) sub-devices were created with respective 
        " + "EU being "){format( 
            len(sub_devs), 
            [d.max_compute_units for d in sub_devs] 
        )}
    except Exception:
        print("Partitioning device by affinity not possible.")
```

**Create Sub-devices**
dpctl.memory: USM Bindings

Support for Python buffers for host-accessible USM types

Full control over allocation

```
import dpctl
import dpctl.memory as dpmem

q = dpctl.SyclQueue()
dev = dpmem.MemryUSMDevice(256, alignment=32, queue=q)
py_obj = bytes(b'abcd' * (m_dev.nbytes // 4))
m_dev.copy_from_host(pyobj)
m_host = dpmem.MemryUSMHost(m_dev.nbytes)
m_host.copy_from_device(m_dev)
bytearray(m_host)
```
dpctl.tensor: Array API

- Reference implementation of Python Array API standard using SYCL
  - A subset of NumPy-like API (~200 functions)
  - Built on top of dpctl, dpctl.memory
  - Under heavy development

```python
import dpctl.tensor as dpt
X = dpt.ones((5, 3), dtype='d', usm_type='device',
            device='opencl:gpu')

# execute on the device where data was allocated
Y = 3 * X

# modify subset of memory
Y[1:4] = dpt.fill(3, 2.7, device=X.device)
```

https://data-apis.org/array-api/latest
Extension Interfaces

- Create a Python ext. to call onemkl::gemv in < 40 loc (fits on a slide)
- Invoke it seamless from Python using dpctl, dpctl.tensor

```python
#include "dpctl4pybind11.hpp"
#include <CL/sycl.hpp>
#include <oneapi/ml.hpp>
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>

void gemv(sycl::queue q,
            dpt::usm_ndarray m,
            dpt::usm_ndarray v,
            dpt::usm_ndarray r,
            const std::vector<sycl::event> &deps = {})
{
    auto n = m.get_shape(0);
    auto m = m.get_shape(1);
    int mat_type_num = m.get_type_num();
    /* various legality checks omitted */
    sycl::event res_ev;
    if (mat_type_num == VAR_DOUBLE) {
        auto *v_ptr = v.get_data<double>();
        auto *m_ptr = m.get_data<double>();
        auto *r_ptr = r.get_data<double>();
        res_ev = oneapi::ml::blas::row_major::gemv(
            q, oneapi::ml::transpose::trans, n, m, 1,
            mat_ptr, m, v_ptr, 1, 0, r_ptr, 1, depends);
    } else
        throw std::runtime_error("unsupported");
    // submit the host task keeping arguments alive
    ht_ev = keep_ args_alive(q, (m, v, r), {res_ev});
    // return the pair of host task event and gemv event
    return std::make_pair(ht_ev, res_ev);
}
PYBIND11_MODULE(oneMKL, m)
{
    // Import the dpctl extensions
    import dpctl;
    m.def("gemv", &gemv, "oneMKL gemv wrapper");
}
```

```python
import dpctl;
import numpy as np
import dpctl.tensor as dpt
import onemkl4py

q = dpctl.SyclQueue("level_zero:gpu")
# Allocate matrices and vectors objects using NumPy
Mnp, vnp = np.random.randn(5, 3), np.random.randn(3)

# Copy data to a USM allocation
M = dpt.asarray(Mnp, sycl_queue=q)
v = dpt.asarray(vnp, sycl_queue=q)
r = dpt.empty((5,), dtype="d", sycl_queue=q)

# Invoke a binding for the oneMKL gemv kernel.
he, cc = onemkl4py.gemv_nonblocking(M, v, r)
# ... other computation may be overlapped with kernel execution

# synchronize to finalize the script
q.wait()
```
A Full Example

oneMKL gemv bindings and solvers implementations using gemv


![Graph showing Native v/s dpctl pybind11 CG solver](image)

- Time per iteration (ms)
- Problem Size (# elements)
Current Limitations

• DPC++ only: Supporting other SYCL compilers is a TBD for future
  • Depends on DPC++ extensions: `filter_selector`, `enqueue_barrier`, `default_context`

• Only USM: Buffers are not supported. Technical questions need to be sorted out first
Status and Ongoing Work

• Fully open source and currently under heavy development
  • https://github.com/IntelPython/dpctl (Source)
  • https://intelpython.github.io/dpctl/latest/index.html (Docs)

• Install from conda or pip

  pip3 install dpctl
  conda install dpctl -c intel
**Wider Ecosystem**

1. **dpctl**
   - SYCL Wrapper classes
   - USM allocators
   - Cython, Pybind11 iface

2. **dpctl.tensor**
   - Math
   - Relational
   - Stats

3. **dpnp**
   - Numba-dpex (JIT Compiler for NumPy, Kernel programming)
   - Drop in NumPy replacement

4. **XGBoost Extension**
   - Scikit-learn extension for XPU

**User-level libraries**
- Python Data API compliant array library based on USM
- Python bindings for subset of SYCL

**Data Parallel Extensions for Python**
- Wider ecosystem
Thanks!