UCX-Py — Progress Summary 2021

Peter Entschev (NVIDIA)

November 30th, 2021

UCX-Py

Introduction

- Python interface for UCX
- Provides sync and async APIs
- Simple replacement for Python communications (e.g., sockets)
- Targeted at library and framework developers
- No low-level communications, UCX or C knowledge required
- Made available to users (e.g., data scientists) via frameworks such as Dask

Using UCX-Py

Send/Receive with CuPy

Server

```python
async def server(ep):
    # allocate buffer and receive tag message
    arr = cupy.empty(10000, dtype='u1')
    await ep.recv(arr)

    # send data back via tag message
    await ep.send(arr)

    await ep.close()
    lf.close()

async def main():
    global lf
    lf = ucp.create_listener(server, port=12345)

    while not lf.closed():
        await asyncio.sleep(0.1)
```

Client

```python
async def client():
    host = ucp.get_address(ifname='eth0')  # get address for eth0
    ep = await ucp.create_endpoint(host, port=12345)

    msg = cupy.zeros(10000, dtype='u1')  # data to send
    await ep.send(msg)  # send tag message

    # receive tag response
    resp = cupy.empty_like(msg)
    await ep.recv(resp)  # receive the echo
    cupy.testing.assert_array_equal(msg, resp)
    await ep.close()
```

Using UCX-Py

Array Protocols

Server

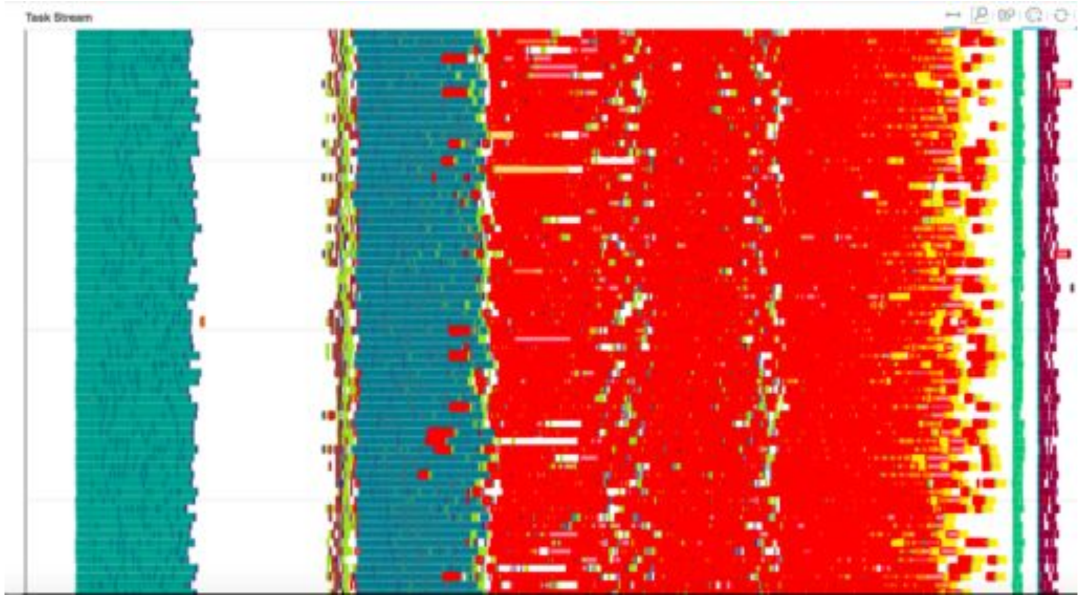
```python
async def server(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    msg = numpy.zeros(10000, dtype='f8')

    # send tag message
    await ep.send(msg)
```

Client

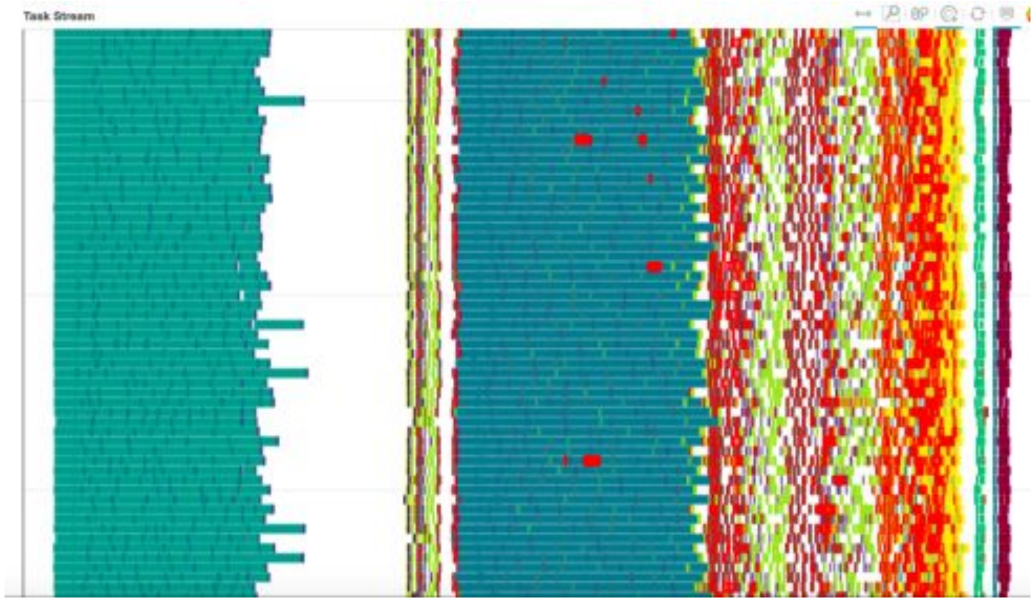
```python
async def client(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    tag_msg = numpy.empty(10000, dtype='f8')  # Must match sender

    # receive tag message into tag_msg
    await ep.recv(tag_msg)
```

UCX-Py in Dask

RAPIDS GPU-BDB



Dask task stream with Python sockets
(Red is communication)



Dask task stream with UCX-Py
(Red is communication)

UCX-Py

Summary of 2021 Progress

- TCP_SOCKCM/UCX 1.11+ support
- Improved endpoint error handling
- Improved stability for InfiniBand devices
- Support for Active Messages
- Support for RMA operations
- Support for IP-less setups (without a UCX Listener)
- Separate UCX-Py Core and Asynchronous APIs — Code cleanup
- Improved Documentation
- Simplified Configuration for InfiniBand (dask-cuda)

UCX-Py — Progress 2021

TCP SOCKCM

- Introduced UCX 1.10
- Many bugs still existed prior to 1.11
- Some features were missing, compared to old implementation
- Common use case: NVLink (intranode) + TCP (internode)
    - Dask
    - NVIDIA Morpheus
- TCP: very important for UCX-Py users

UCX-Py — Progress 2021

Endpoint Error Handling

- Error callbacks can be registered for each endpoint
- Prior to UCX 1.11 some transports didn't support them (e.g., cuda_ipc)
- May be used to track errors as well as closed connections
- Previously UCX-Py implemented its own connection closing mechanism
- Error callbacks are more versatile — no hangs due to disconnected endpoint
- Default in UCX-Py (when UCX 1.11+ is used)

UCX-Py — Progress 2021

Active Messages

- Provide more familiar interface for Python users
- Avoid unnecessary tag-matching overhead
- Metadata (size, memory type) can be packed into the message
- User can register allocator for received messages
- Zero-copy conversion possible for allocators supporting
    - `__array_interface__`
    - `__cuda_array_interface__`
    - Python buffer protocol

UCX-Py — Progress 2021

Tag Messages Example

Server

```
async def server(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    msg = numpy.zeros(10000, dtype='f8')

    # send tag message
    await ep.send(msg)
```

Client

```
async def client(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    tag_msg = numpy.empty(10000, dtype='f8')  # Must match sender

    # receive tag message into tag_msg
    await ep.recv(tag_msg)
```

UCX-Py — Progress 2021

Active Messages Example

Server

```
async def server(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    msg = numpy.zeros(10000, dtype='f8')

    # send tag message
    await ep.send(msg)

    # send active message
    await ep.am_send(msg)
```

Client

```
async def client(ep):
    # buffer -> __array_inteface__ / __cuda_array_interface__
    tag_msg = numpy.empty(10000, dtype='f8')  # Must match sender

    # receive tag message into tag_msg
    await ep.recv(tag_msg)

    ucp.register_am_allocator(
        lambda n: numpy.empty(n, dtype='u1'), 'host'
    )
    ucp.register_am_allocator(
        lambda n: cupy.empty(n, dtype='u1'), 'cuda'
    )

    # receive active message, no pre-allocation or prior knowledge
    # of size/memory type required
    am_msg = await ep.am_recv()
```

UCX-Py — Progress 2021

Remote Memory Access Operations

- Enables direct access to local Python memory by a remote peer
- Extended by UCXIO, a class simulating Python streams over UCX RMA
- Only supported in core API at the moment
- Contributed by Matt Baker, ORNL

UCX-Py — Progress 2021

IP-Less Setups

- UCX worker address can be queried
- Address can be serialized as a byte-string
- Byte-string can be distributed (via DNS SRV record, shared filesystem, etc.)
- Remote processes can establish connections using that address
- No need for an IP address to be assigned
  - When no IP address is assigned, requires InfiniBand or another capable interconnect
- No listener is needed

UCX-Py — Progress 2021

Separate Core and Asynchronous APIs

- Separation of Core and Asynchronous APIs has been consolidated
- No more Core→Asynchronous API dependency
- Better Cython code organization into multiple files
- Cython code still has a "main" file including all other files to keep binary to a single shared library file

UCX-Py — Progress 2021

Improved Documentation

- Added docker-specific documentation on NVLink
- More information on common OS limits, such as maximum file descriptors and connections open
- Moved Dask docs with more complete examples to https://docs.rapids.ai/api/dask-cuda/nightly/ucx.html

UCX-Py — Progress 2021

Reference Docker Container

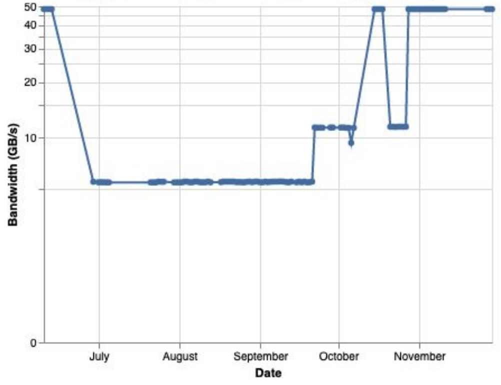
- Included reference Docker container
- Helps introduce all UCX-Py requirements for new users
- Used as well as reference for application deployment
- Allows running all tests and benchmarks
- Currently limited to basic memory and transports only (no CUDA or MOFED)

UCX-Py — Progress 2021

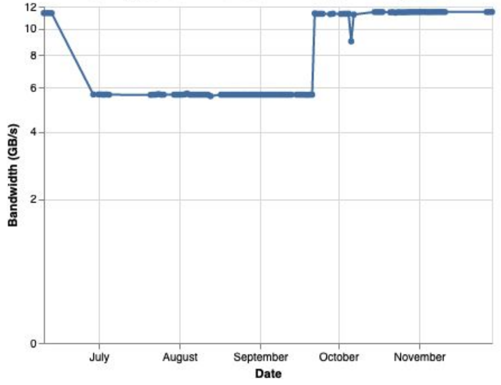
UCX-Py Nightly CI

- Simplified CI capabilities running nightly
- Multiple UCX versions tests
- Running on an NVIDIA DGX-1
    - ○ Supports CUDA, NVLink, InfiniBand
- Allowed us to quickly notice functionality and performance regressions

UCX-Py Core - AM Transfer API - cupy - Transport CUDA_IPC_NV2



UCX-Py Core - AM Transfer API - cupy - Transport RC



Dask-CUDA

Simplified Configuration for InfiniBand

- Mapping of CUDA↔InfiniBand devices is totally done by UCX
- Considerable code simplification — no more libhwloc required in UCX-Py
- Allows proper identification in non-baremetal systems (e.g., cloud)
- Does not require users to indicate InfiniBand auto/manual device mapping
- RDMACM working — required for large-scale clusters

UCX-Py — Progress 2021

Upstreaming Code to Mainline OpenUCX

- Got delayed (again) in 2021
- Effort to add UCX-Py tests into OpenUCX CI started (PR #7412)
    - CI currently lacks CUDA support
- Once CI limitations are resolved, upstreaming will be done in order:
    - Core library (Cython + C code)
    - Asynchronous (high-level) library
    - Packaging (PyPI, conda)

UCX-Py — 2022

Planned Improvements

- Multi-threading support
- Improvements on small message transfers (< 1MB), up to 5x slower today
- Remove the obligation to specify UCX_TLS for applications (e.g., Dask)
  - Today a CUDA context is necessary prior to UCX initialization for CUDA↔IB GPUDirectRDMA
- Please reach out to us if you have any requests
  - https://github.com/rapidsai/ucx-py
- Contributors welcome and encouraged!

THANK YOU

Peter Entschev (NVIDIA), pentschev@nvidia.com