UCX-Py: C++ Backend

Peter Entschev (NVIDIA)

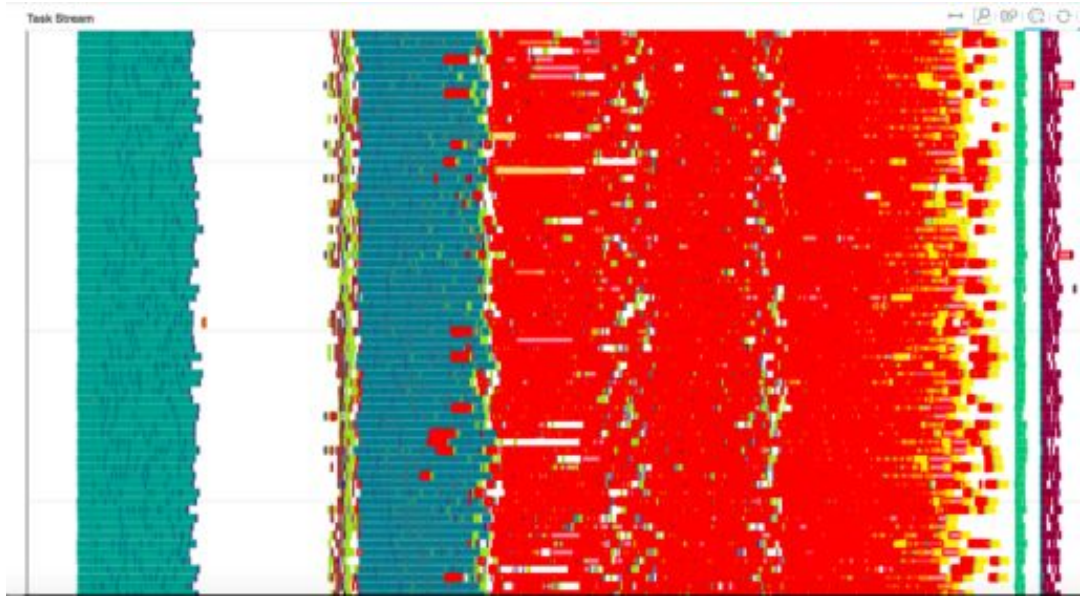
September 20th, 2022

UCX-Py
Introduction

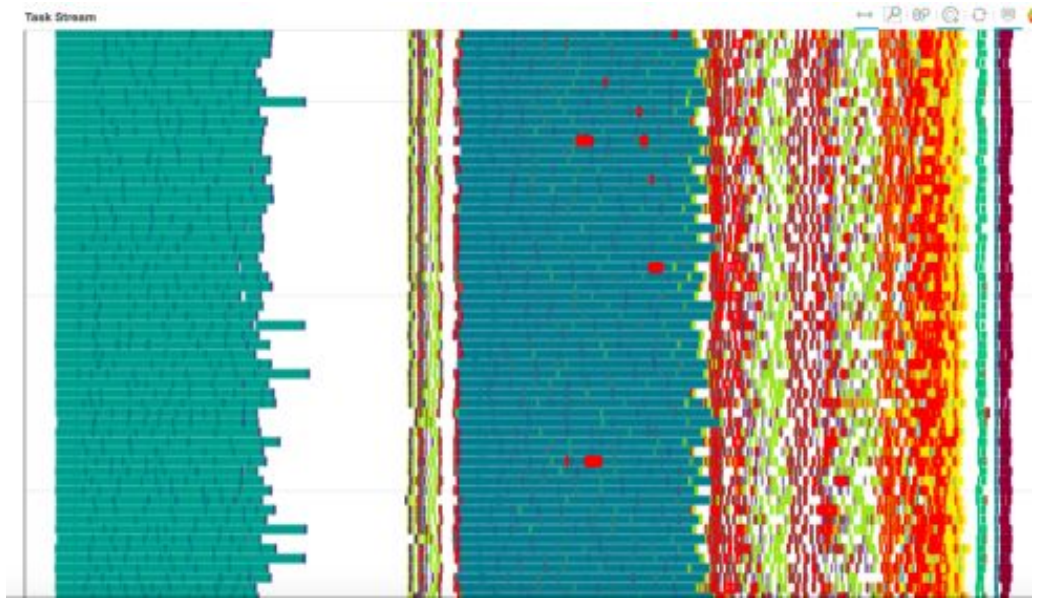
- Python interface for UCX
- Provides sync and async APIs
- Simple replacement for Python communications (e.g., sockets)
- Targeted at library and framework developers
- No low-level communications, UCX or C knowledge required
- Made available to users (e.g., data scientists) via frameworks such as Dask

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)

C++ Backend

Motivation

- Improve performance of small messages
- Existing UCX-Py Cython implementation does not support multithreading
- Cython is great, but has shortcomings
  - Code can be more complex than pure C++
  - Does not expose complete standard library
- Provide seamless integration with object-oriented applications without wrappings
- Attract more developers for object-oriented languages
  - Object-oriented in C++ / shim layer to high-level language
- Currently nicknamed "UCXX" (subject to change)

UCXX

Motivation: Python Asyncio Overhead

```python
import asyncio, time

async def noop():
    pass

async def cpu_task():
    t0 = time.time()
    for _ in range(1000000):
        await asyncio.create_task(noop())
    t1 = time.time()
    print(t1-t0)

await cpu_task()
```

```
11.78263521194458
```

```python
import time

async def noop():
    pass

async def cpu_coro():
    t0 = time.time()
    for _ in range(1000000):
        await noop()
    t1 = time.time()
    print(t1-t0)

await cpu_coro()
```

```
0.14650940895080566
```

11.78263521194458 / 0.14650940895080566 = ~80x!

Source: https://stackoverflow.com/a/55766474

UCXX

Motivation: Python Overhead

- Asyncio is a large source of overhead
  - Up to 170x slower compared to sync code (https://stackoverflow.com/a/55766474)
- Thousands of small tasks (e.g., small message transfers) add to total runtime

- Some of possible solutions:
  - Reducing total number of tasks
  - Reducing overhead of asyncio tasks (is it even possible?)
  - Replacing asyncio by something more efficient (uvloop, Trio, etc.)
  - Potentially others?

UCXX Optimizations

Summary

- UCX worker thread
- Delayed submission of message transfers
- Direct notification of Python futures from C++
- Python multi-buffer transfers

UCXX Optimizations

UCP Worker Progress Thread

- Spawn separate thread to progress the UCP worker
- State of UCXX requests is set by the thread
- May progress the worker continuously or in event-based mode

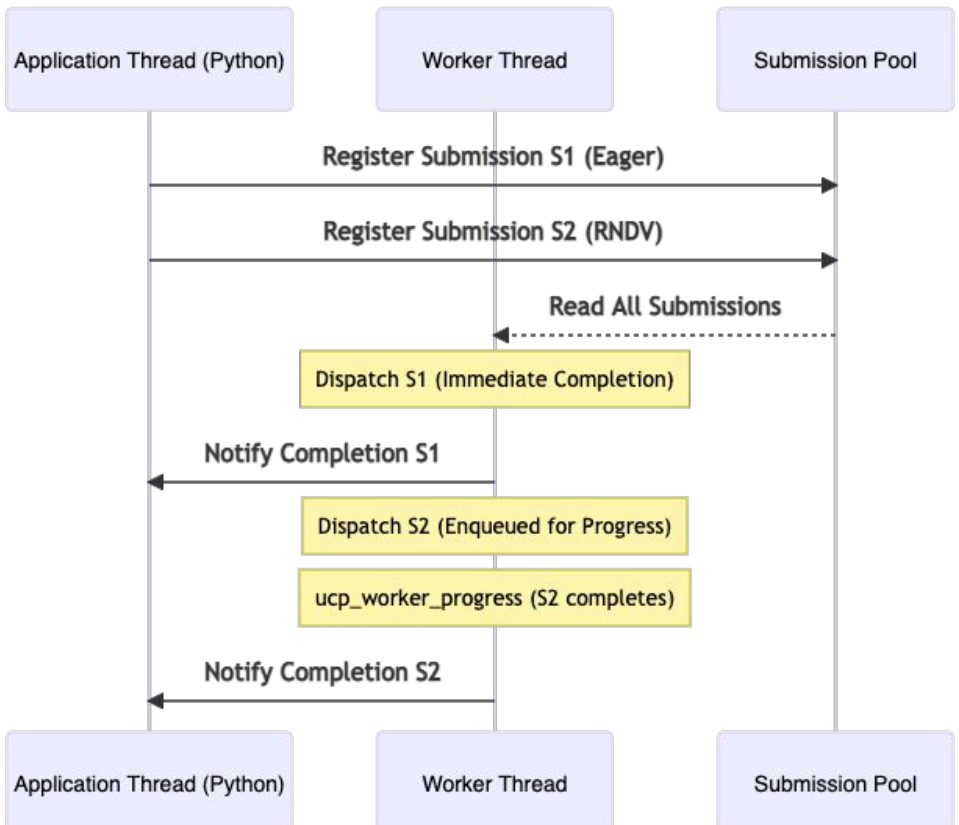
UCXX Optimizations

Delayed Submission

- Postpone all message transfers to the UCP worker progress thread
- Remove all overhead from calling thread

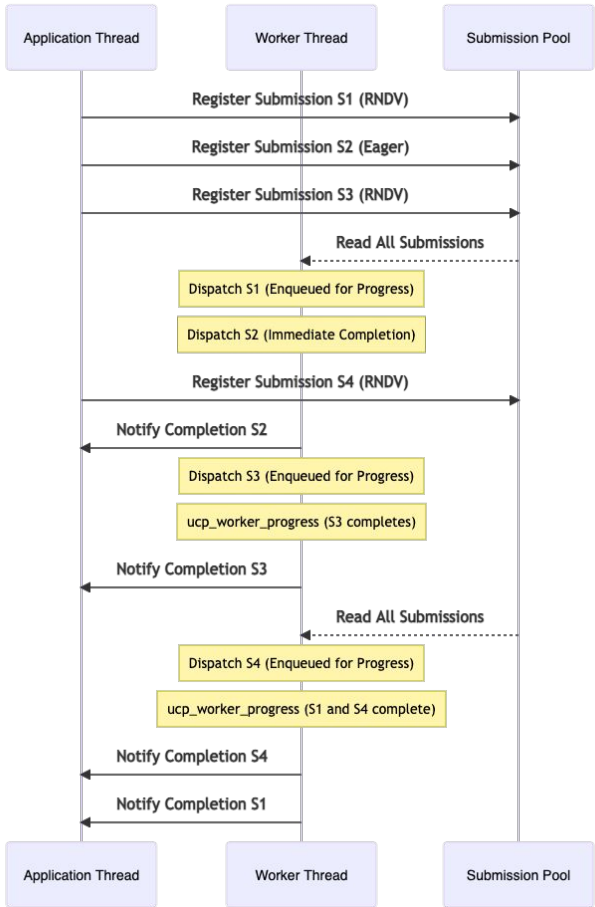
UCXX Optimizations

Delayed Submission - Simple Sequence Diagram



UCXX Optimizations

Delayed Submission - Complex Sequence Diagram



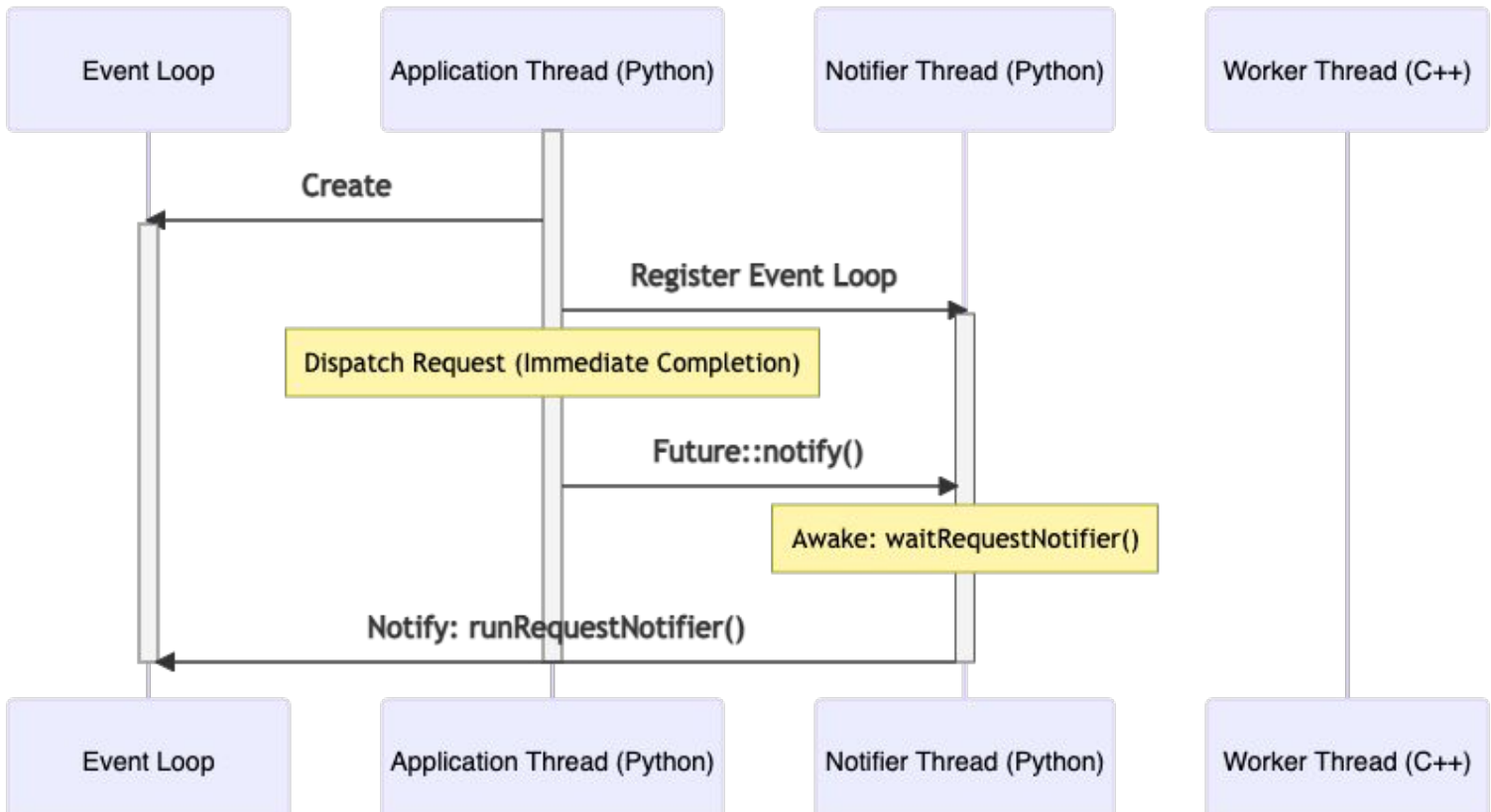
UCXX Optimizations

Python Notifier Thread

- Separate Python thread sharing event loop with application thread
- Awakes when a UCP request completes
- Notifies a Python future
- Avoids Python GIL acquisition from worker progress function
- GIL required only for shortest possible period when notifying future

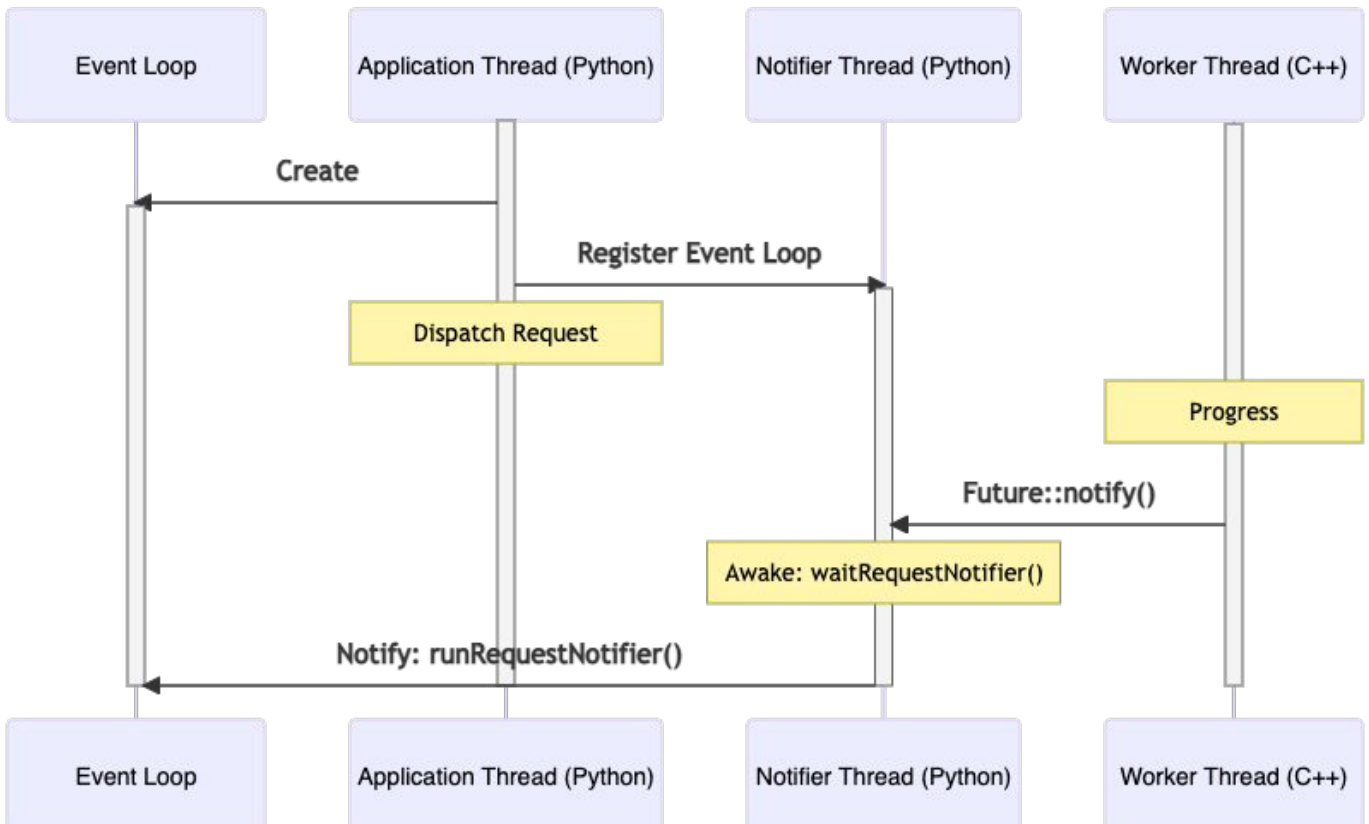
UCXX Optimizations

Python Notifier Thread



UCXX Optimizations

Python Notifier Thread



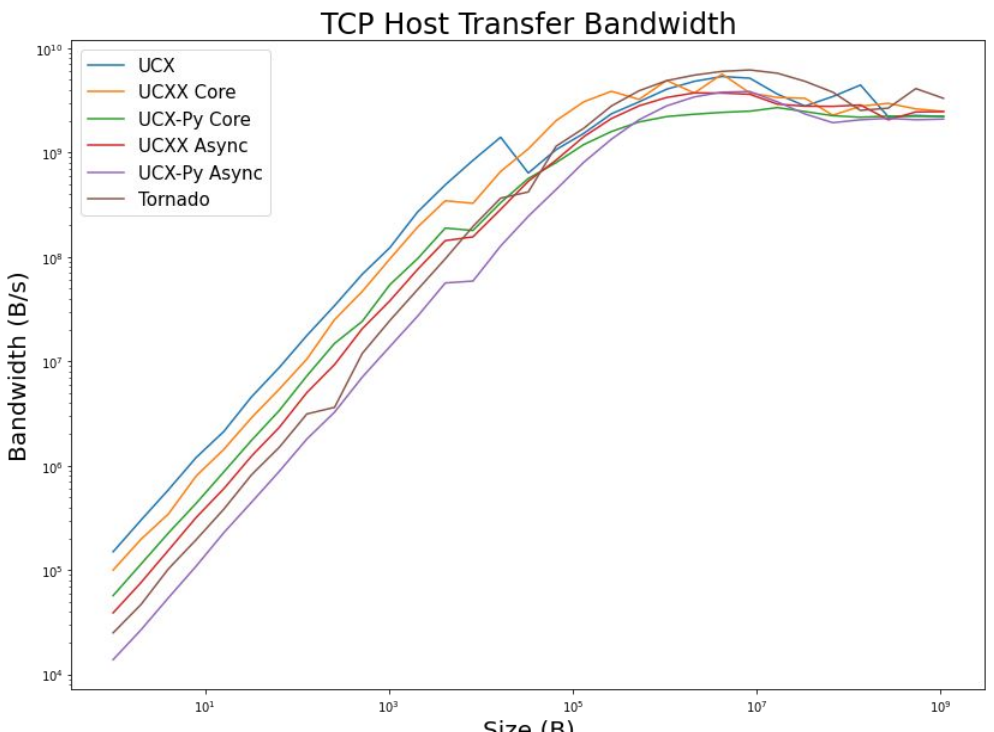
UCXX Optimizations

Python Multi-Buffer Transfers

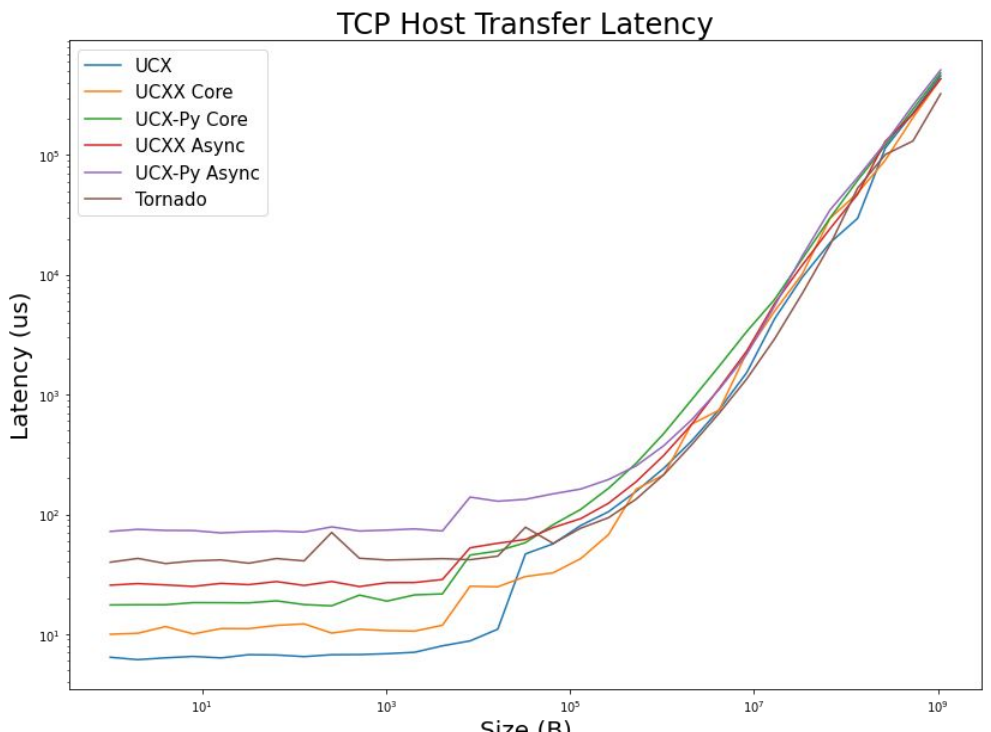
- Only one Python future to be waited per operation, regardless of number of buffers
- Allow sending a list of buffers in a single send operation
- Receive a list of buffers in a single recv operation
  - User doesn't know anything about the content (size, type) beforehand
  - Buffers are allocated by the implementation at receive time
- Memory allocation
  - Host: C `malloc()`/`free()`
  - CUDA: `rmm::device_buffer`

Message Transfer in Python

- Small transfers in Python are slow
- Native Performance gap ~10 (Tornado vs UCX)
- Dask does *a lot* of small transfers (< 1KB), e.g.:
  - Data sizes and number of frames
  - Heartbeart

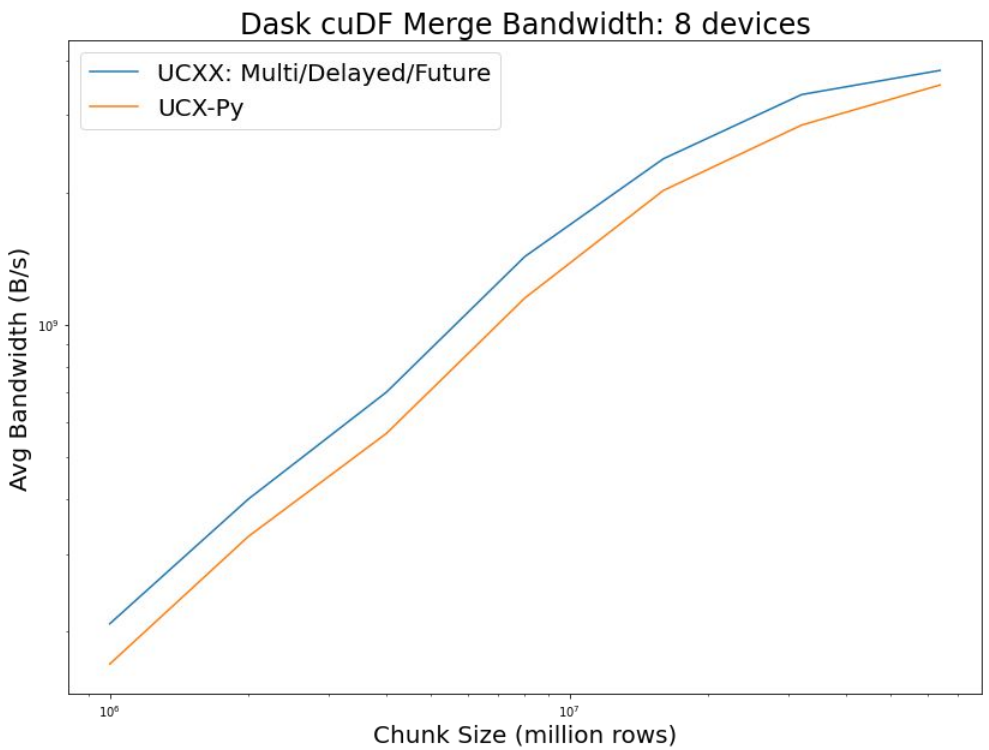


- Flat latency for small transfers
  - Dominated by implementation overhead

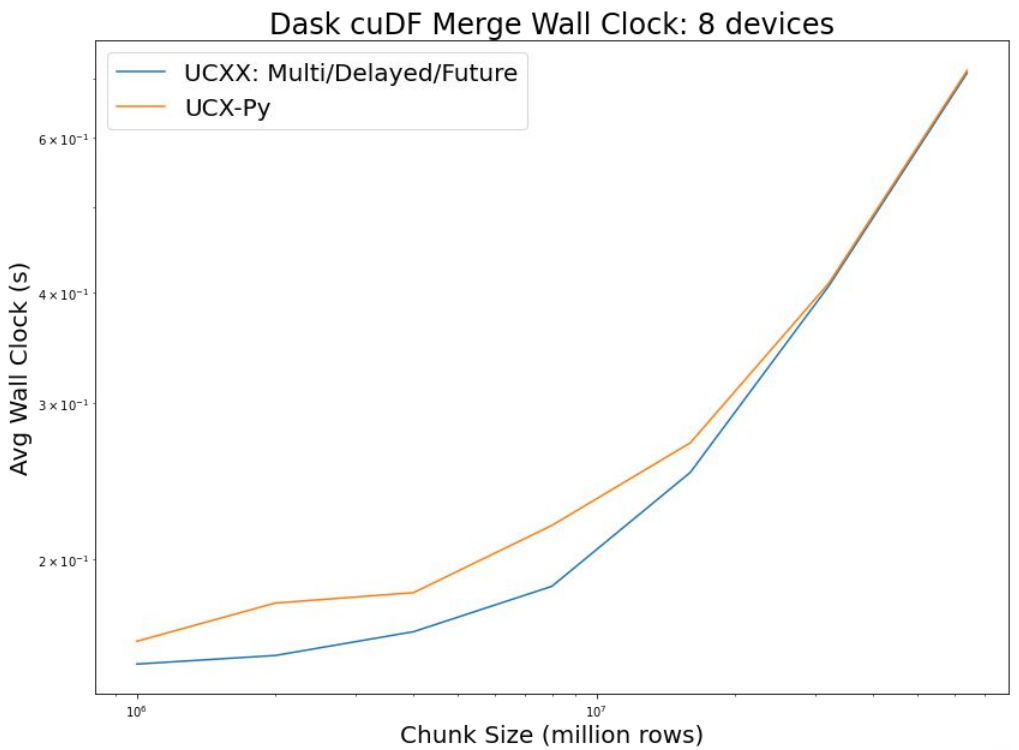


Message Transfer in Dask

- Using new UCXX implementation to reduce overhead by:
  - Decreasing total number of asyncio tasks (multi-buffer transfers)
  - Decreasing blocking tasks in event loop (delayed submission)



- Wall-clock can be reduced by reducing general overhead (when dominated by short-lived tasks, e.g., small transfers)



UCXX API

C++

```
// UCP Context
std::shared_ptr<ucxx::Context> context = ucxx::createContext();

// UCP Worker
std::shared_ptr<ucxx::Worker> worker = context->createWorker();
auto worker = context->createWorker();
std::shared_ptr<ucxx::Worker> worker = ucxx::createWorker(context);
auto worker = ucxx::createWorker(context);

// UCP Listener
std::shared_ptr<ucxx::Listener> listener = worker->createListener(13337, listenerCallback, listenerCallbackArg);

// UCP Endpoint
std::shared_ptr<ucxx::Endpoint> ep = worker->createEndpointFromHostname("127.0.0.1", 13337);
std::shared_ptr<ucxx::Endpoint> ep = worker->createEndpointFromWorkerAddress(worker->getAddress());
std::shared_ptr<ucxx::Endpoint> ep = listener->createEndpointFromConnRequest(connRequest);
```

UCXX API

C++ (Cont.)

```
// Tag transfer
std::shared_ptr<ucxx::Request> tagSendReq = ep->tagSend(sendPtr, sendBytes, 0);
std::shared_ptr<ucxx::Request> tagRecvReq = ep->tagRecv(recvPtr, recvBytes, 0);

// Stream transfer
std::shared_ptr<ucxx::Request> streamSendReq = ep->streamSend(sendPtr, sendBytes);
std::shared_ptr<ucxx::Request> streamRecvReq = ep->streamRecv(recvPtr, recvBytes);

// Request methods
bool isCompleted = transferReq->isCompleted();
transferReq->checkError();
ucs_status_t transferStatus = transferReq->getStatus();
transferReq->cancel();
```

UCXX API

C++ Sample

```
void local_send_recv() {
    std::shared_ptr<ucxx::Context> context = ucxx::createContext();
    std::shared_ptr<ucxx::Worker> worker = context->createWorker();
    worker->startProgressThread();

    auto ep1 = worker->createEndpointFromWorkerAddress(worker->getAddress());
    auto ep2 = worker->createEndpointFromWorkerAddress(worker->getAddress());

    std::vector<int> send{0, 1, 2, 3, 4, 5, 6, 7};
    std::vector<int> recv(send.size());

    std::vector<std::shared_ptr<ucxx::Request>> requests{
        ep1->tagSend(send.data(), send.size() * sizeof(int), 0),
        ep2->tagRecv(recv.data(), recv.size() * sizeof(int), 0)
    }
    waitRequests(worker, requests);
}
```

UCXX API

Python Core Sample

```
def local_send_recv():
    ctx = ucx_api.UCXContext()
    worker = ucx_api.UCXWorker(ctx)
    worker.start_progress_thread()

    ep1 = ucx_api.UCXEndpoint.create_endpoint_from_worker_address(worker, ucx_api.UCXAddress.from_worker(worker))
    ep2 = ucx_api.UCXEndpoint.create_endpoint_from_worker_address(worker, ucx_api.UCXAddress.from_worker(worker))

    send_msg = np.arange(8, dtype=np.int32)
    recv_msg = np.empty(8, dtype=np.int32)
    requests = [ep1.tag_send(send_msg, tag=0), ep2.tag_recv(recv_msg, tag=0)]

    wait_requests(worker, requests)
```

UCXX API

Python Async Sample

```
def local_send_recv_async():
    ep1 = await ucp.create_endpoint_from_worker_address(ucp.get_worker_address())
    ep2 = await ucp.create_endpoint_from_worker_address(ucp.get_worker_address())

    send_msg = np.arange(8, dtype=np.int32)
    recv_msg = np.empty(8, dtype=np.int32)

    await asyncio.gather(ep1.send(send_msg, tag=0, force_tag=True), ep2.recv(recv_msg, tag=0, force_tag=True))
```

UCXX API

Python Async Sample - Multi-Buffer

```
def local_send_recv_multi_async():
    ep1 = await ucp.create_endpoint_from_worker_address(ucp.get_worker_address())
    ep2 = await ucp.create_endpoint_from_worker_address(ucp.get_worker_address())

    send_msg = [np.arange(8, dtype=np.int32) for i in range(8)]

    _, recv_msg = await asyncio.gather(ep1.send_multi(send_msg, tag=0, force_tag=True), ep2.recv_multi(tag=0, force_tag=True))
```

UCXX State and Availability

- Requires C++17 (C++14 possible if needed, but without CUDA Python support)
- Parts of UCX-Py still unimplemented (parts of ucp_address, RMA, AM)
- Missing parts of documentation
- Missing CMake support
- Thorough code review pending
- Still undecided whether this will be merged into UCX-Py repo or entirely new project
- Will be made available late 2022 (schedule permitting)


THANK YOU
Peter Entschev (NVIDIA), pentschev@nvidia.com
NVIDIA