Charm++ with UCX

A focus on inter-GPU communication

Jaemin Choi

University of Illinois Urbana-Champaign

Dec 2, 2020
UCF Virtual Workshop 2020

Charm++ Messaging API

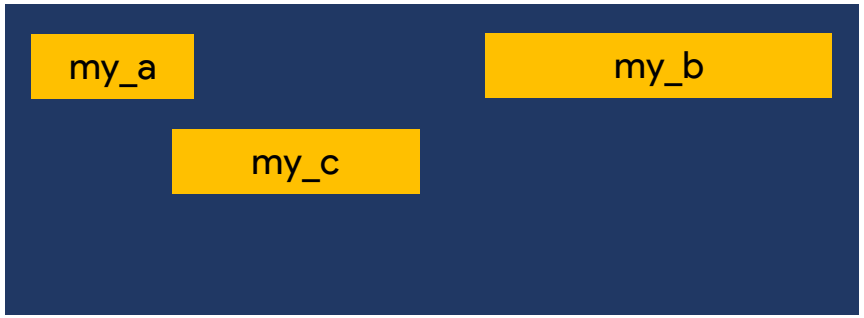
Message-Driven Execution

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, vector<double> b,
         vector<double> c)
```

Source object

Destination object



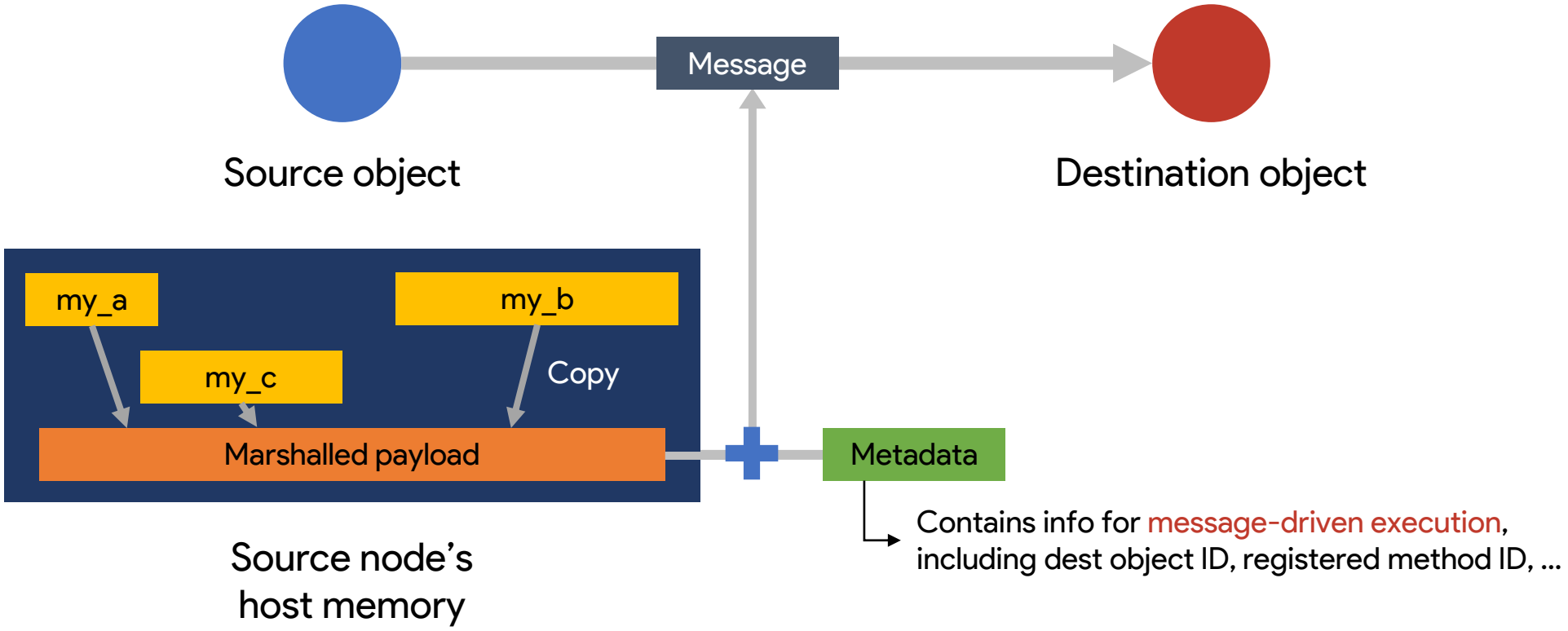
Source node's
host memory

Charm++ Messaging API

Parameter Marshalling

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, vector<double> b,
         vector<double> c)
```



Charm++ Messaging API

Parameter Unmarshalling

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, vector<double> b,
         vector<double> c)
```

- Message is unmarshalled (unpacked) on the receiver
- Metadata is used to invoke the correct method on the destination object, e.g. `foo()`
- Destination object can access the received data (`a, b, c`)

Charm++ GPU Messaging

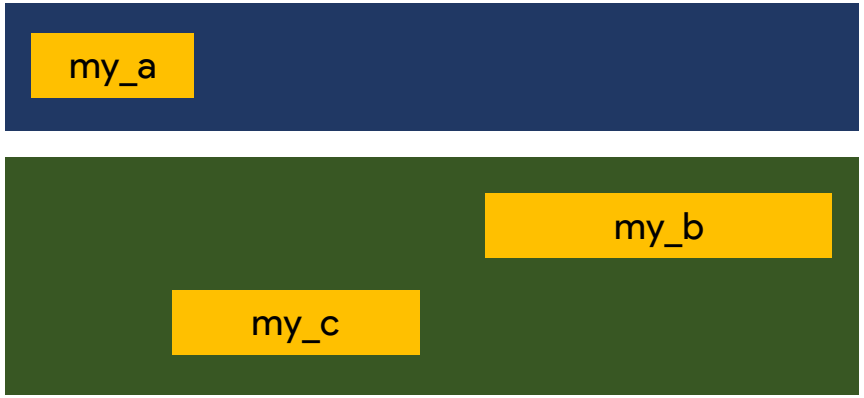
Built on Zero Copy EM Post API[1] & UCX

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, nocopydevice double* b,
         nocopydevice double* c)
```

Source object

Destination object



Source node's GPU memory

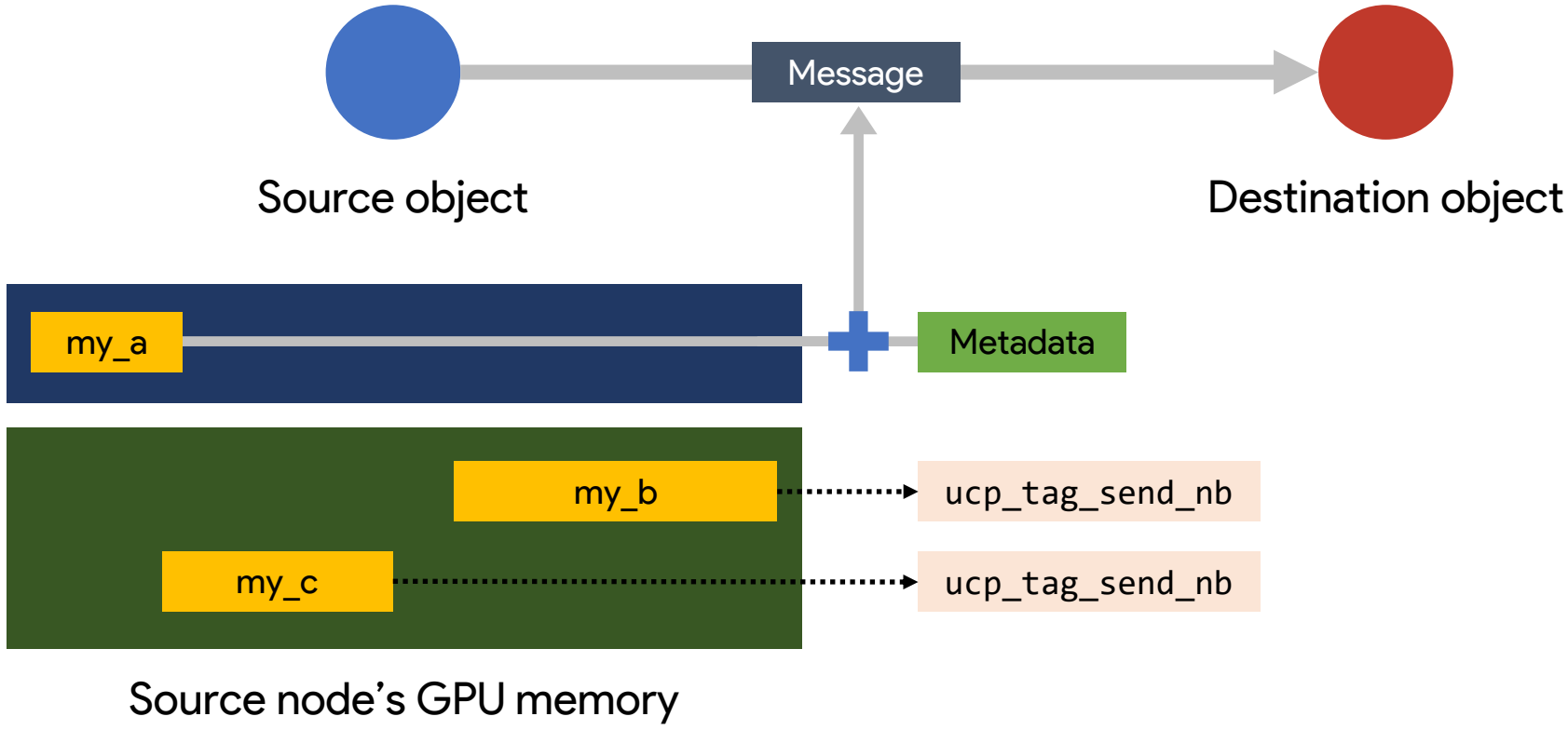
[1] https://charm.readthedocs.io/en/latest/charm++/manual.html#zero-copy-entry-method-post-api

Charm++ GPU Messaging

Parameter Marshalling & UCX Send

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, nocopydevice double* b,
         nocopydevice double* c)
```

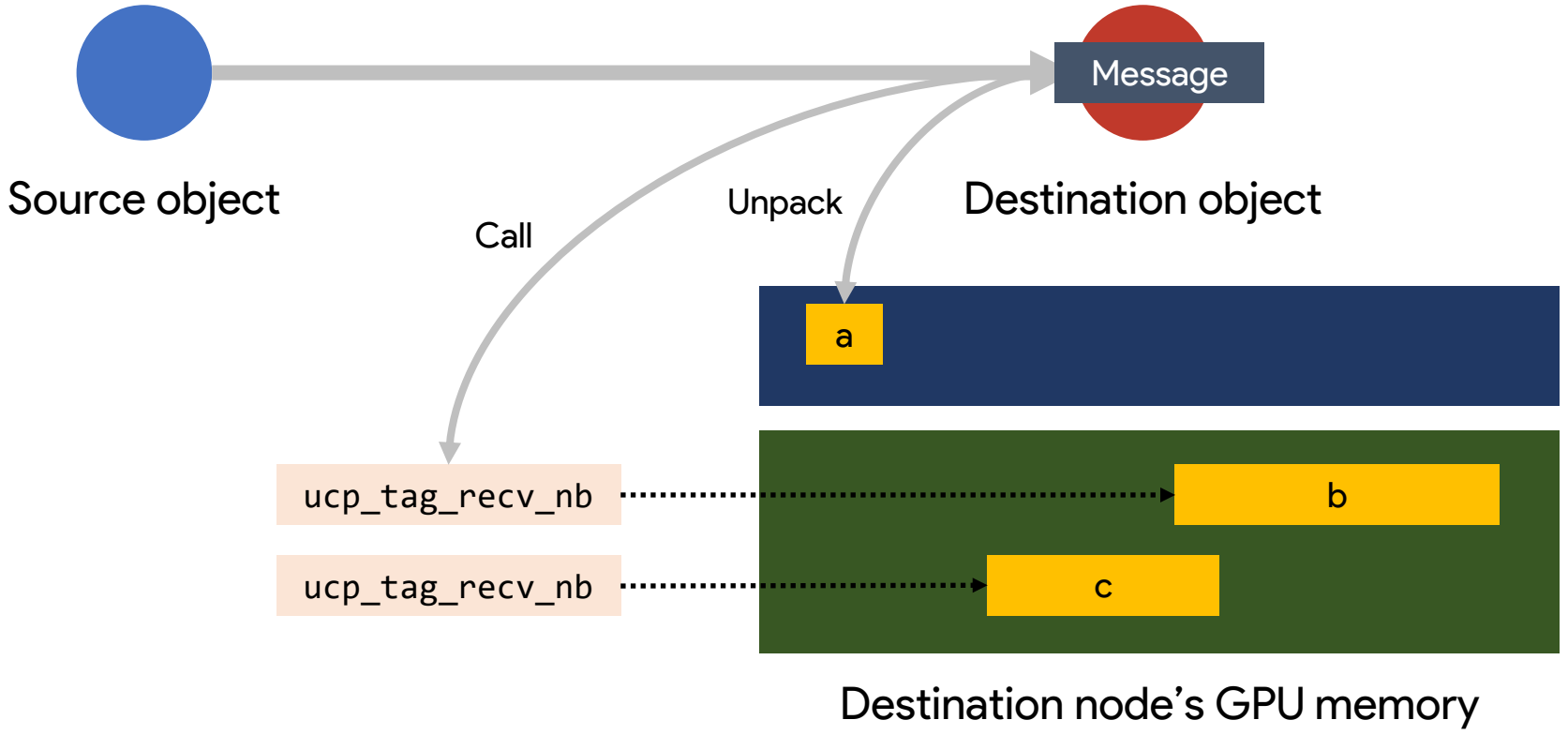


Charm++ GPU Messaging

Parameter Unmarshalling & UCX Recv

```
thisProxy[dest].foo(my_a, my_b, my_c);
```

```
void foo(int a, nocopydevice double* b,
         nocopydevice double* c)
```



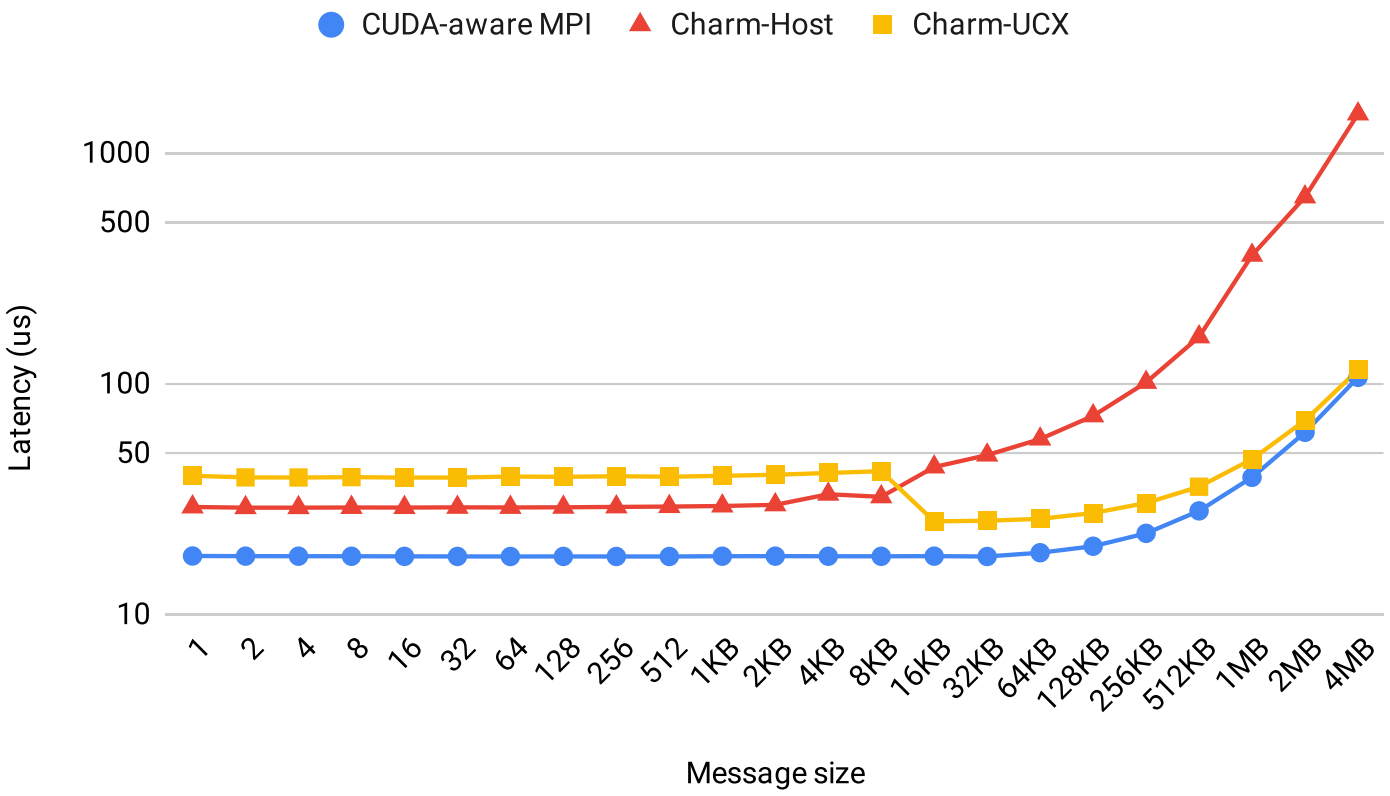
Preliminary Performance

Evaluation Platform/Configurations

- OLCF Summit
- OSU latency benchmark with GPU source/dest buffers (MPI, Charm++ versions)
- IBM Spectrum MPI v10.3.1.2
- UCX v1.9.0
- Inter-process (single-node)
- Inter-node

Preliminary Performance

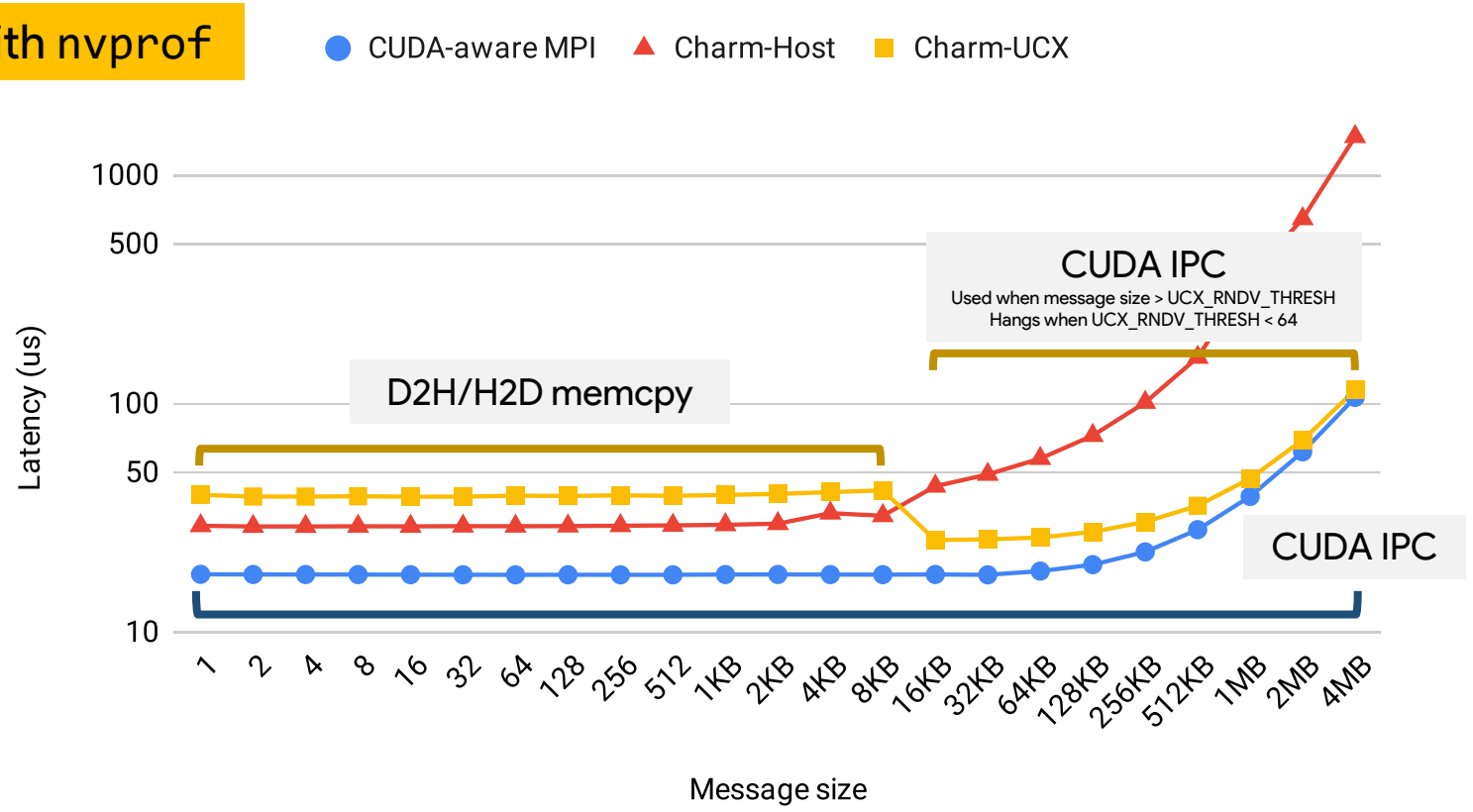
Inter-process OSU Latency Benchmark



Preliminary Performance

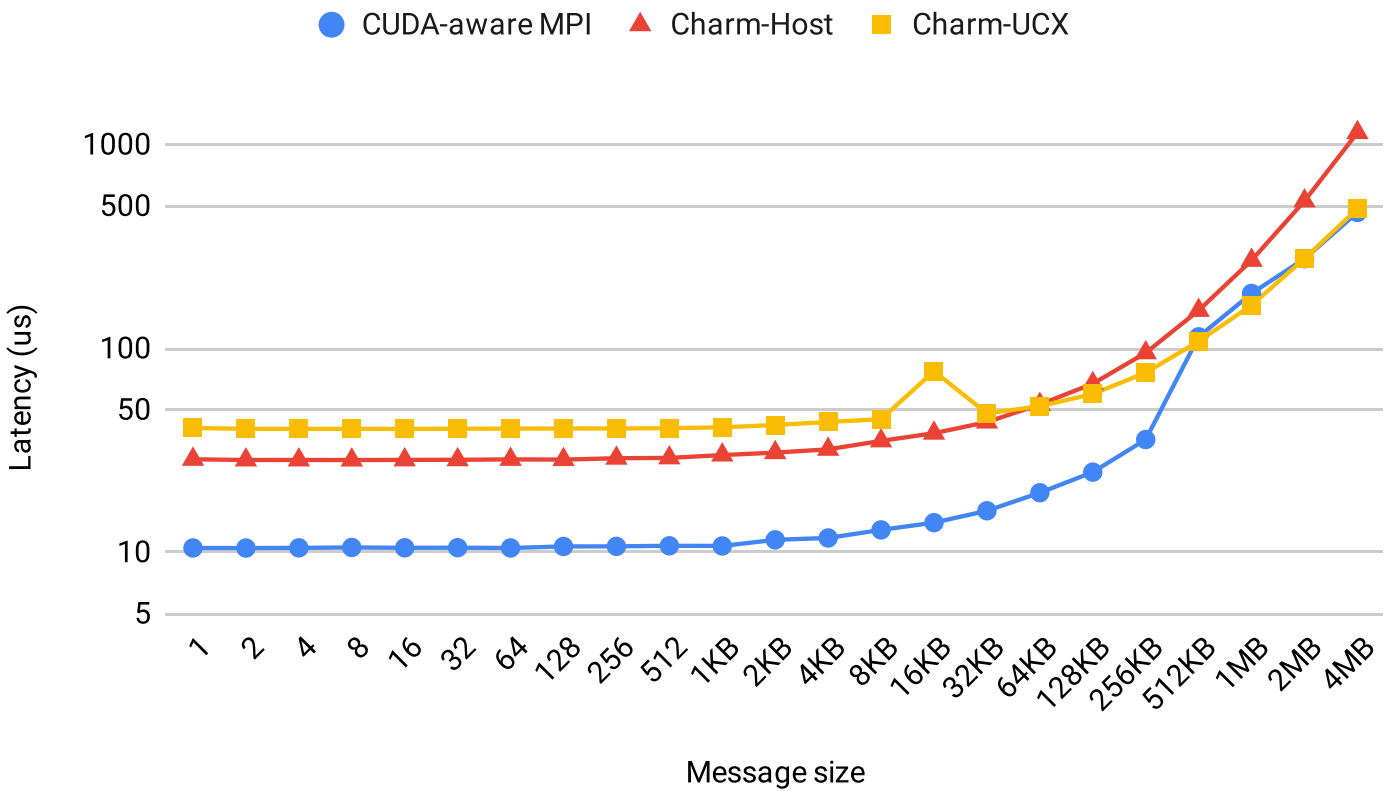
Inter-process OSU Latency Benchmark

Profiled with nvprof



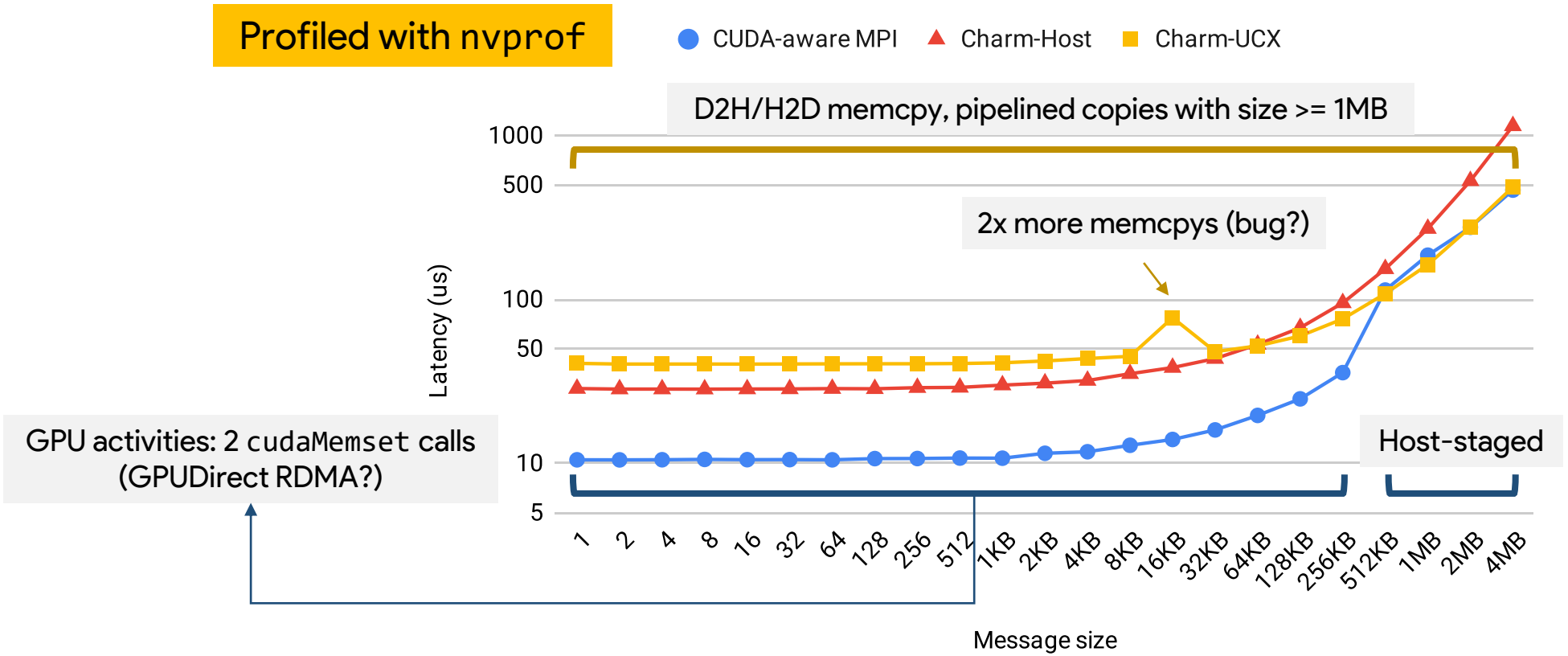
Preliminary Performance

Inter-node OSU Latency Benchmark



Preliminary Performance

Inter-node OSU Latency Benchmark



Conclusion

Charm++ with UCX

- Relatively easy experience to integrate GPU-enabled UCX in Charm++
  - Single implementation for intra-node & inter-node
- Similar designs possible with other task-based runtime systems (esp. those that require metadata in host memory)
- Need to investigate UCX behavior and improve performance
- Plan to be included in next major Charm++ release

Thank you!

Please feel free to contact me at jaemin@acm.org.

Appendix

UCX/Charm++ Build Commands on OLCF Summit

- Build libcheck (https://github.com/libcheck/check)
  - Modify `[libcheck_install_path]/lib64/pkgconfig/check.pc` to use `lib64` instead of `lib`
- Build GDRCopy (https://github.com/NVIDIA/gdrcopy)
  - `git checkout 2.1`
  - `mkdir install`
  - `PKG_CONFIG_PATH=[libcheck_install_path]/lib64/pkgconfig make PREFIX=[gdrcopy_path]/install CUDA=[cuda_install_path] all install`
- Build UCX (https://github.com/openucx/ucx)
  - `git checkout v1.9.0`
  - `./autogen.sh`
  - `mkdir build && mkdir install && cd build`
  - `../contrib/configure-release --prefix=[ucx_path]/install --with-cuda=[cuda_path] --with-gdrcopy=[gdrcopy_install_path]`
  - `make -j && make install`
- Build PMIx (https://github.com/openpmix/openpmix)
- Build Charm++ (https://github.com/UIUC-PPL/charm)
  - `./buildold charm++ ucx-linux-ppc64le smp cuda ompipmix -j -g --with-production -basedir=[pmix_install_path] --basedir=[ucx_install_path]`
  - `export LD_LIBRARY_PATH=[ucx_install_path]/lib:$LD_LIBRARY_PATH`