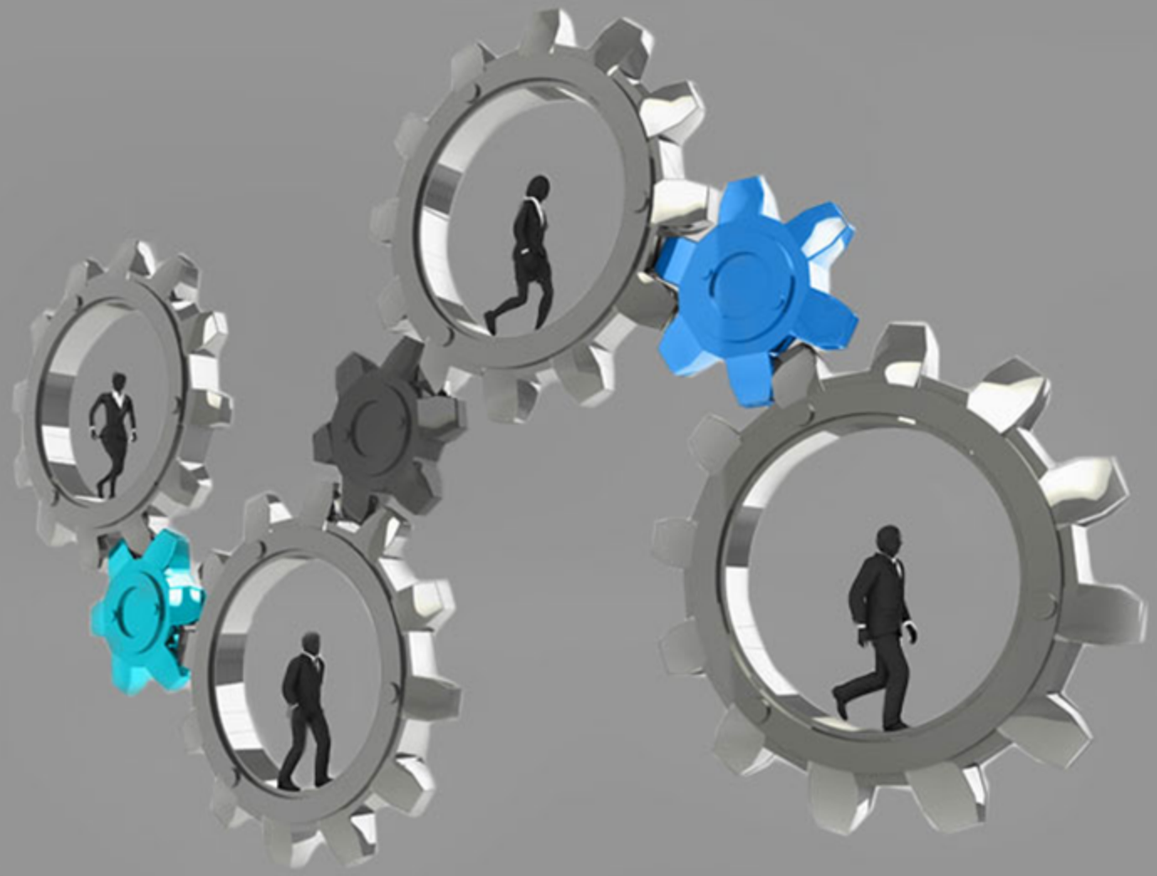
ENABLER OF CO-DESIGN

UCF



Unified Collective Communications (UCC):

Designing and Implementing Next Generation Collectives Library

Manjunath Gorentla Venkata on behalf of Collectives WG, UCF Workshop, Dec 3rd, 2020

UCC is a collective communication operations API and library that is flexible, complete, and feature-rich for current and emerging programming models and runtimes.

Outline

- Design challenges
- Properties of the solution
- API Overview
- Reference implementation and project status
- Roadmap

UCC Design Challenges

- Unified collective stack for HPC and DL/ML workloads
  - Need to support a wide variety of semantics
  - Need to optimize for different performance sensitives - latency, bandwidth, throughput
  - Need for flexible resource scheduling model
  - Need for flexible ordering model
- Unified collective stack for software and hardware transports
  - Need for complex resource management - scheduling, sharing, and exhaustion
  - Need to support multiple semantic differences – reliability, completion
- Unify parallelism and concurrency
  - Concurrency – progress of a collective and the computation
  - Parallelism – progress of many independent collectives
- Unify execution models for CPU, GPU, and DPU collectives
  - Two-way execution model – control operations are tightly integrated
    - Do active progress, returns values, errors, and callbacks will less overhead
  - One-way execution model – control operations are loosely integrated
    - passive progress, and handle return values (GPU/DPUs)

UCC Design: Properties we want

- Scalability and performance for key use-cases
  - Enable efficient implementation for common cases in MPI, OpenSHMEM and AI/ML
- Extensible
  - We cannot possibly cover all the options and features for all use cases
  - We need the API and semantics that is modular
- Opt in-and-out
  - If for a certain path some semantic is not applicable, we need a way to opt-out
- Explicit API and semantics over implicit
  - Explicit -> implicit is easier than implicit -> explicit
- Minimal API surface area
  - Lessen the mental load
  - A few set of abstractions to understand and go into details when required
- Other properties such as the ability to override functionality, composability, programmability, and many more are important.

UCC's Solution: Key Concepts

- Abstractions
  - Abstract the resources required for collective operations
  - Local: Library, Context, Endpoints
  - Global: Teams
- Operations
  - Create/modify/destroy the resources
  - Build, launch and finalize collectives
- Properties
  - Explicit way to request for optional features, semantics, and optimizations
  - Provides an ability to express and request many cross-cutting features
  - Properties are preferences expressed by the user of the library and what the library provides must be queried
    - In the future, we might extend the properties to be "required" in addition to the "preferred" and "query" model
  - Examples: Collective types, ordering, synchronization, thread model

- Challenge is to map a broad range of requirements to these concepts
- Overall, this approach has worked
  - Minimizes the API surface area,
  - Extendible
  - Scalable and efficient from the prototype implementations

Concepts

1. Abstractions for Resources
   - Collective Library
   - Communication Context
   - Teams
   - Endpoints
2. Operations
   - Collective Operations

UCC Library Handle

ucc_lib_h encapsulates all resources related to the group communication operations

Semantics

- All UCC operations should be invoked between the init and finalize operations.

- Properties
  - Collective types
  - Thread model
  - Synchronization model

- Operations
  - Routines for initializing and finalizing the library handle.
  - Query the properties

UCC Library Handle

```c
/**
 *  @ingroup UCC_LIB
 *
 *  @brief The @ref ucc_init initializes the UCC library.
 *
 *
 *  @param [in]  params    user provided parameters to customize the library functionality
 *  @param [in]  config    UCC configuration descriptor allocated through
 *                         @ref ucc_lib_config_read "ucc_config_read()" routine.
 *  @param [out] lib_p     UCC library handle
 *
 *  @parblock
 *
 *  @b Description
 *
 *  A local operation to initialize and allocate the resources for the UCC
 *  operations. The parameters passed using the ucc_lib_params_t and
 *  @ref ucc_lib_config_h structures will customize and select the functionality of the
 *  UCC library. The library can be customized for its interaction with the user
 *  threads, types of collective operations, and reductions supported.
 *  On success, the library object will be created and ucc_status_t will return
 *  UCC_OK. On error, the library object will not be created and corresponding
 *  error code as defined by ucc_status_t is returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by ucc_status_t
 */
static inline ucc_status_t ucc_init(const ucc_lib_params_t *params,
                                    const ucc_lib_config_h config,
                                    ucc_lib_h *lib_p)
{
    return ucc_init_version(UCC_API_MAJOR, UCC_API_MINOR, params, config,
                            lib_p);
}
```

UCC Context Handle

ucc_context_h local resources required for expressing network parallelism

Usage

- Encapsulate local network resources such as IB QPs, SHARP trees, or UCX worker
- To express affinity between network resource and thread invoking the collective
- Resource sharing between multiple collectives

- Properties
  - Shared or exclusive
  - Thread model
  - Synchronization model

- Operations
  - Routines for creating and destroying the context
  - Query the properties

UCC Context

```
/**
 *  @ingroup UCC_CONTEXT
 *
 *  @brief The @ref ucc_context_create routine creates the context handle.
 *
 *  @param [in]   lib_handle  Library handle
 *  @param [out]  params      Customizations for the communication context
 *  @param [out]  config      Configuration for the communication context to read
 *                            from environment
 *  @param [out]  context     Pointer to the newly created communication context
 *
 *  @parblock
 *
 *  @b Description
 *
 *  The ucc_context_create creates the context and ucc_context_destroy
 *  releases the resources and destroys the context state. The creation of context
 *  does not necessarily indicate its readiness to be used for collective or other
 *  group operations. On success, the context handle will be created and ucc_status_t will return
 *  UCC_OK. On error, the library object will not be created and corresponding
 *  error code as defined by ucc_status_t is returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by ucc_status_t
 */


ucc_status_t ucc_context_create(ucc_lib_h lib_handle,
                                const ucc_context_params_t *params,
                                const ucc_context_config_h  config,
                                ucc_context_h *context);
```

UCC Team handle

ucc_team_h encapsulates the global resources required for collective communication operations.

Usage

- Map MPI communicator/”group” abstractions to UCC teams
- Negotiate and converge on the semantics of how local resources are used during collective operations

- Properties
  - Shared or exclusive
  - Thread model
  - Synchronization model

- Operations
  - Routines for creating and destroying the context
  - Query the properties

Team Create Interface

```
/**
 *  @ingroup UCC_TEAM
 *
 *  @brief The routine is a method to create the team.
 *
 *  @param  [in]  contexts          Communication contexts abstracting the resources
 *  @param  [in]  num_contexts      Number of contexts passed for the create operation
 *  @param  [in]  team_params       User defined configurations for the team
 *  @param  [out] new_team          Team handle
 *
 *  @parblock
 *
 *  @b Description
 *
 *  @ref ucc_team_create_post is a nonblocking collective operation to create
 *  the team handle. It takes in parameters ucc_context_h and ucc_team_params_t.
 *  The ucc_team_params_t provides user configuration to customize the team and,
 *  ucc_context_h provides the resources for the team and collectives.
 *  The routine returns immediately after posting the operation with the
 *  new team handle. However, the team handle is not ready for posting
 *  the collective operation. ucc_team_create_test operation is used to learn
 *  the status of the new team handle. On error, the team handle will not
 *  be created and corresponding error code as defined by ucc_status_t is
 *  returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by ucc_status_t
 */
ucc_status_t ucc_team_create_post(ucc_context_h *contexts,
                                  uint32_t num_contexts,
                                  const ucc_team_params_t *team_params,
                                  ucc_team_h *new_team);
```

Properties: Example with Teams abstraction

```
typedef struct ucc_team_params {
    uint64_t                mask;
    ucc_post_ordering_t     ordering;
    uint64_t                outstanding_colls;
    uint64_t                ep;
    uint64_t                *ep_list;
    ucc_ep_range_type_t     ep_range;
    uint64_t                team_size;
    ucc_coll_sync_type_t    sync_type;
    ucc_team_oob_coll_t     oob;
    ucc_team_p2p_conn       p2p_conn;
    ucc_mem_map_params_t    mem_params;
    ucc_ep_map_t            ep_map;
} ucc_team_params_t;
```

```
typedef struct ucc_team_attr {
    uint64_t                mask;
    ucc_post_ordering_t     ordering;
    uint64_t                outstanding_colls;
    uint64_t                ep;
    ucc_ep_range_type_t     ep_range;
    ucc_coll_sync_type_t    sync_type;
    ucc_mem_map_params_t    mem_params;
} ucc_team_attr_t;
```

Collective Operations: Building Blocks

UCF Unified Communication Framework

```
ucc_status_t ucc_collective_init(ucc_coll_op_args_t* coll_args,
                                 ucc_coll_req_h* request, ucc_team_h team);

ucc_status_t ucc_collective_post(ucc_coll_req_h request);

ucc_status_t ucc_collective_init_and_post(ucc_coll_op_args_t* coll_args,
                                          ucc_coll_req_h* request,
                                          ucc_team_h team);

ucc_status_t ucc_collective_finalize(ucc_coll_req_h request);
```

Collective Operations

- Collective operations : ucc_collective_init(…) and ucc_collective_init_and_post(…)
- Local operations: ucc_collective_post, test, and finalize
- Initialize with ***ucc_collective_init(…)***
  - Initializes the resources required for a particular collective operation, but does not post the operation
- Completion
  - The ***test*** routine provides the status
- Finalize
  - Releases the resources for the collective operation represented by the request
  - The post and wait operations are invalid after finalize

UCC Spec: Interfaces and semantics fully specified

- Download from the UCC github and build it.
- Specification is ahead of the code now
- The version 1.0 is agreed by the working group and merged into the master branch
  - Changes are allowed but requires high-bar for integration.
- Over 60 pages of detailed information about the interfaces and semantics
- Doxygen based documentation
  - Both pdf and html available

Contents

Experimental Implementations

UCC : A convergence of multiple implementations

UCF Unified Communication Framework

- UCC API has emerged as this convergence …
- Now working towards converged implementation

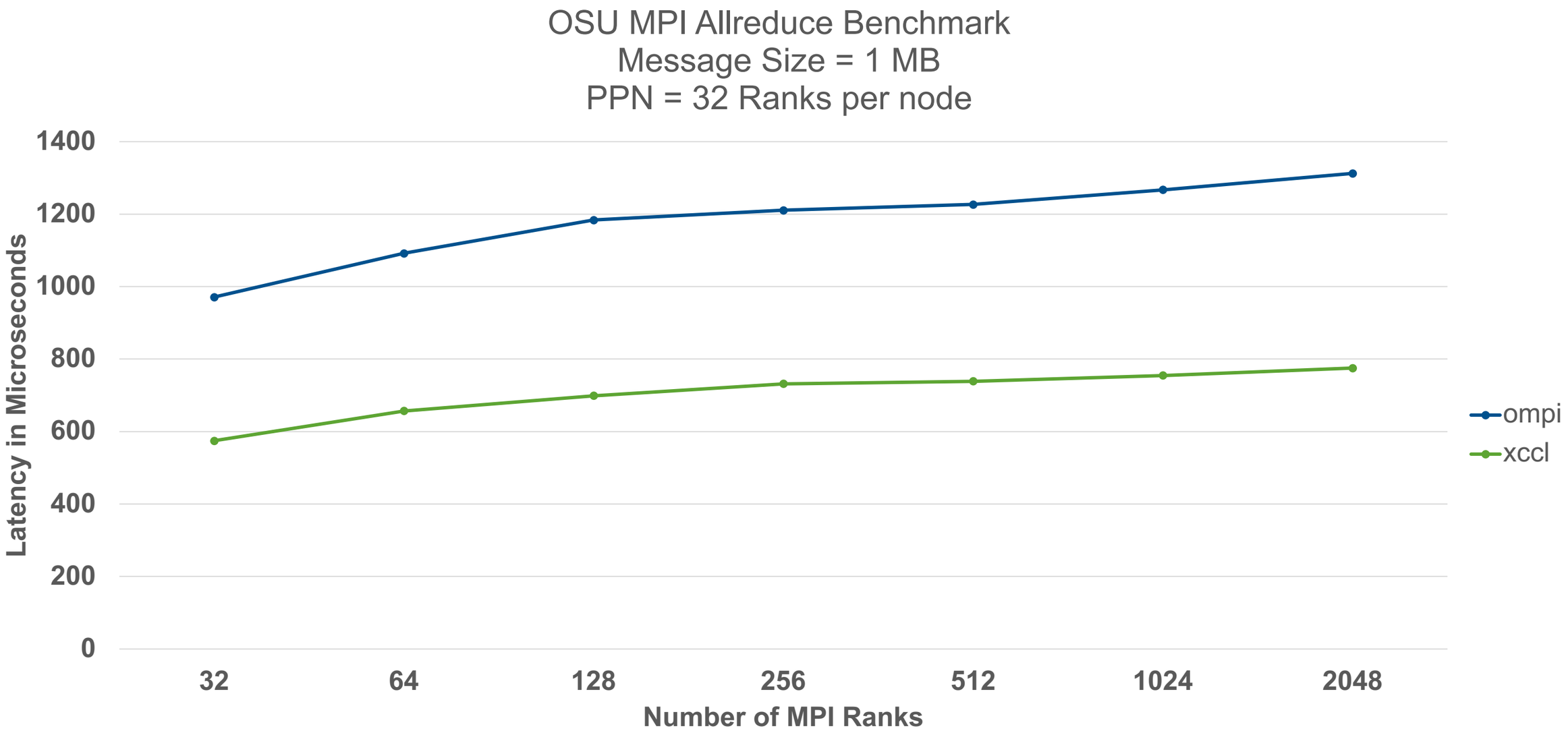
- Particularly XUCG and XCCL
- XCCL
  - Driven by NVIDIA/Mellanox and hierarchical based design
  - https://github.com/openucx/xccl
- XUCG
  - Driven by Huawei and reactive based design
  - https://github.com/openucx/xucg

- HCOLL, PAMI and other collectives design and implementation

XCCL: Collective implementation using UCC API and semantics

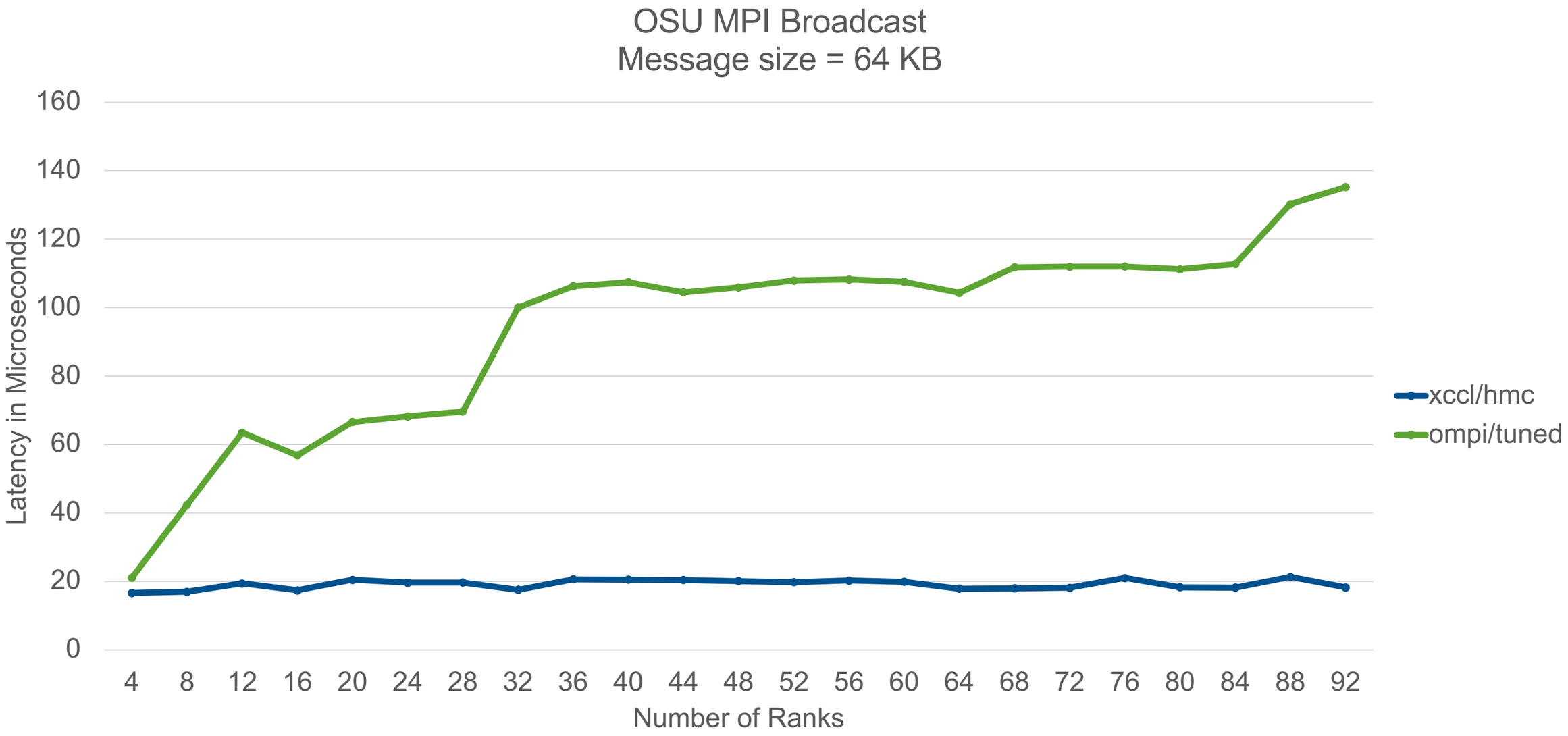
- Developed to experiment with UCC API, design, and semantics
  - Uses XCCL namespace instead of UCC
  - Implements a subset of UCC API
  - Code-base evolving along with the design discussions in the WG
- Hierarchical-based implementation
  - Supports composition of shared memory, software and hardware collectives
- Supports both software and hardware transports
  - UCX based implementation for general network transport
  - Leverages SHARP collectives when appropriate hardware is available
  - Leverages hardware multicast support for broadcast collective operation
  - Specialized shared memory collectives for systems with high core count
  - Offloaded collectives for DPUs
  - Supports using GPU buffers for collective operations
- Supports HPC and AI/ML semantics
  - Currently integrated with Open MPI and PYTorch
  - Production-ready and used with real workloads

XCCL based MPI Allreduce latency



Slide courtesy: Valentine Petrov (NVIDIA)

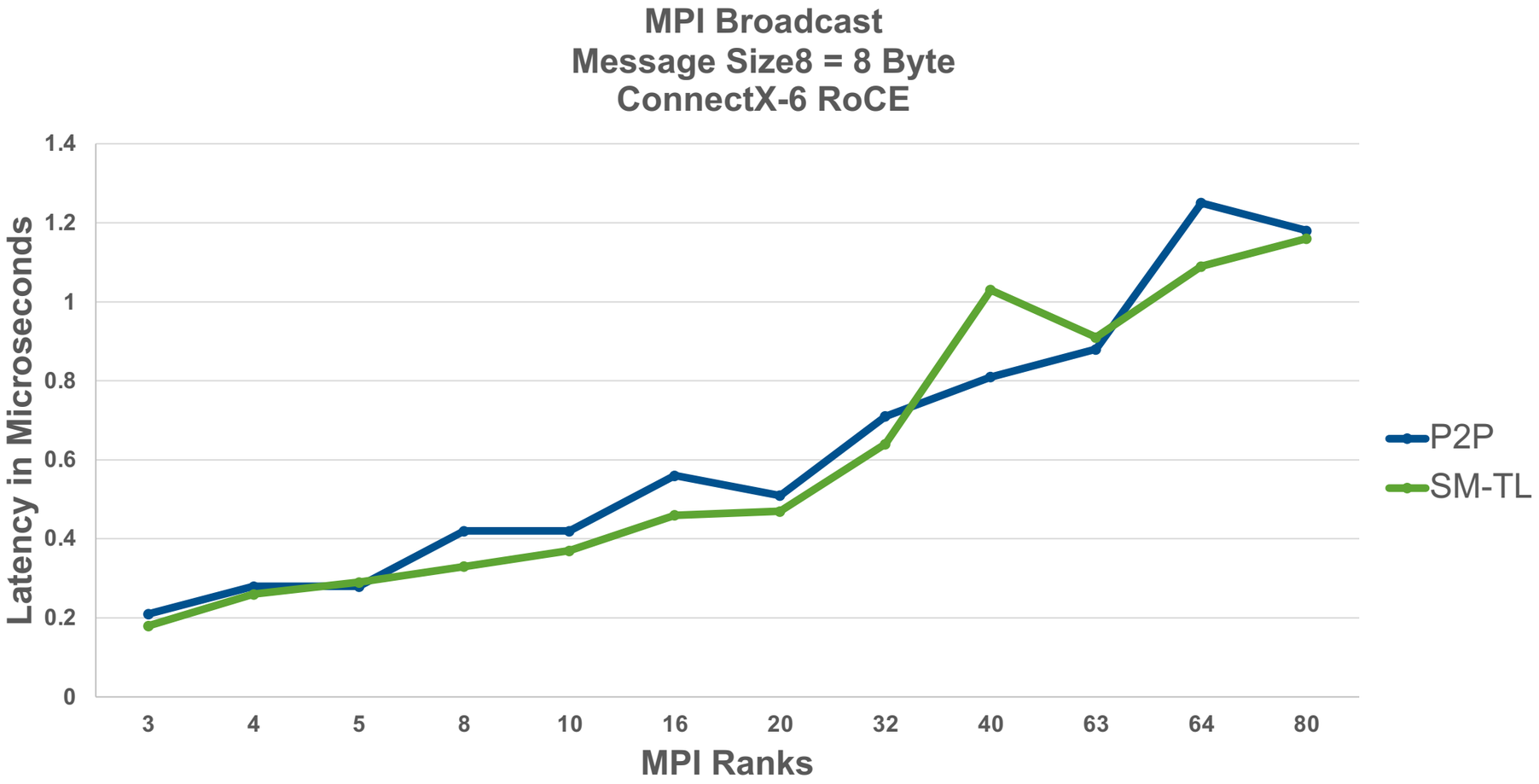
XCCL based MPI Broadcast latency



Slide courtesy: Valentine Petrov (NVIDIA)

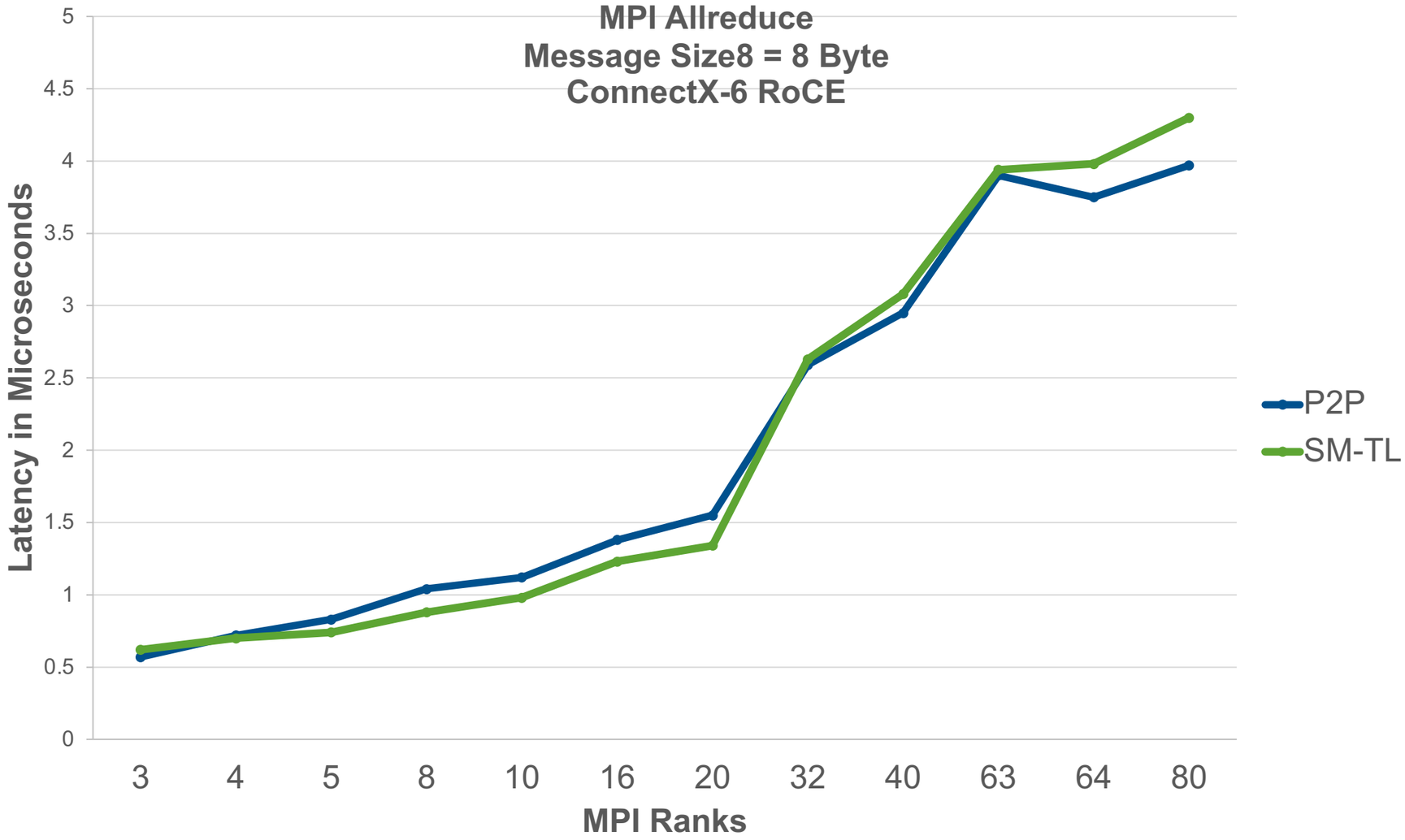
XUCG Experimental Results

XUCG based Broadcast (Preliminary Results)



Data courtesy: Alex Margolin (HUAWEI)

XUCG based Allreduce (Preliminary Results)



Slide courtesy: Alex Margolin (HUAWEI)

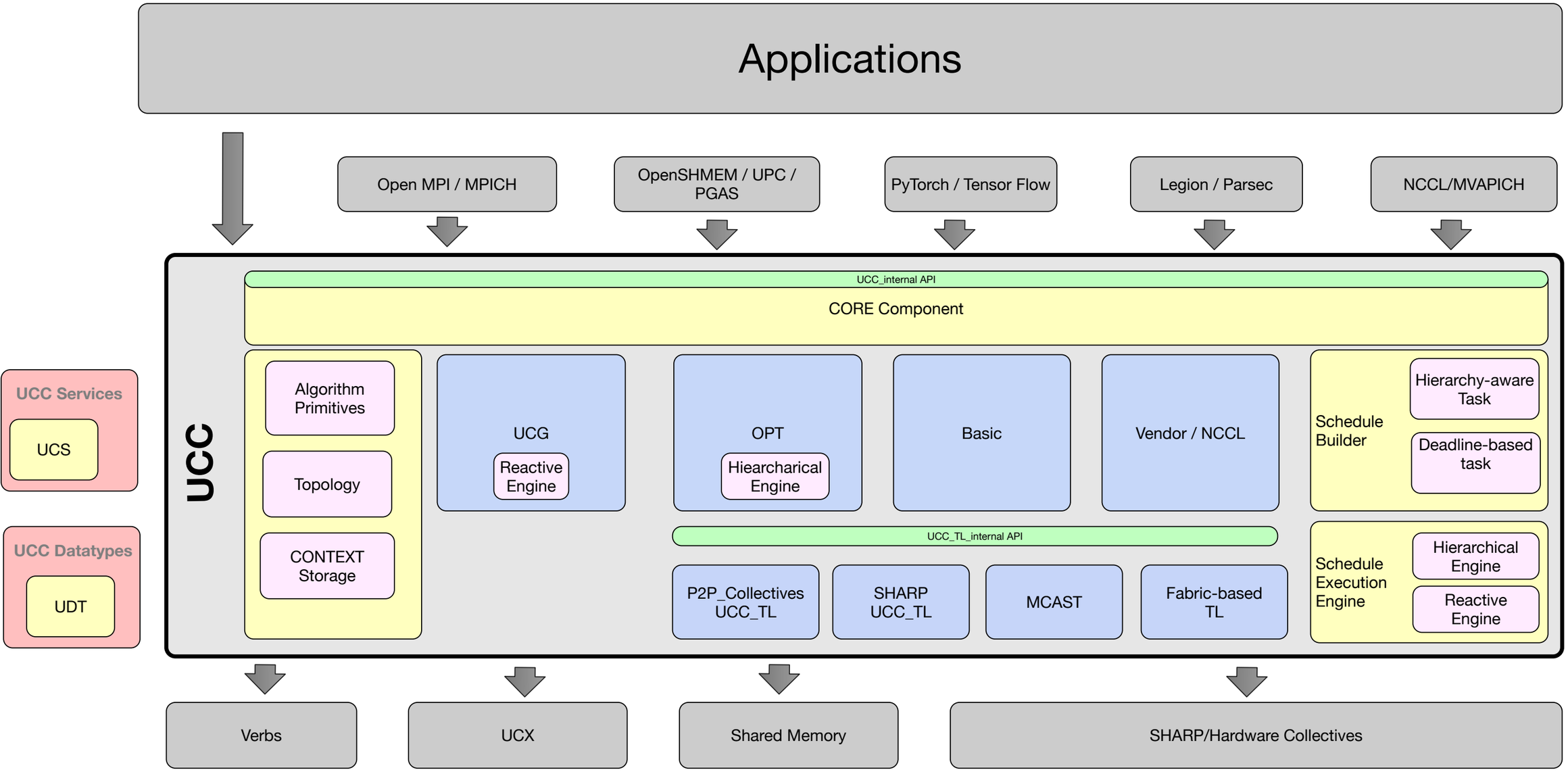
Talks on XCCL and XUCG

- One-to-many UCT transports, part I: Shared-memory (Alex/HUAWEI)
- One-to-many UCT transports, part II: Multicast (Morad/HUAWEI)
- Until UCC is available - UCG status update (Alex/HUAWEI)
- Scaling Facebook's Deep Learning Recommender Model (DLRM) with UCC/XCCL

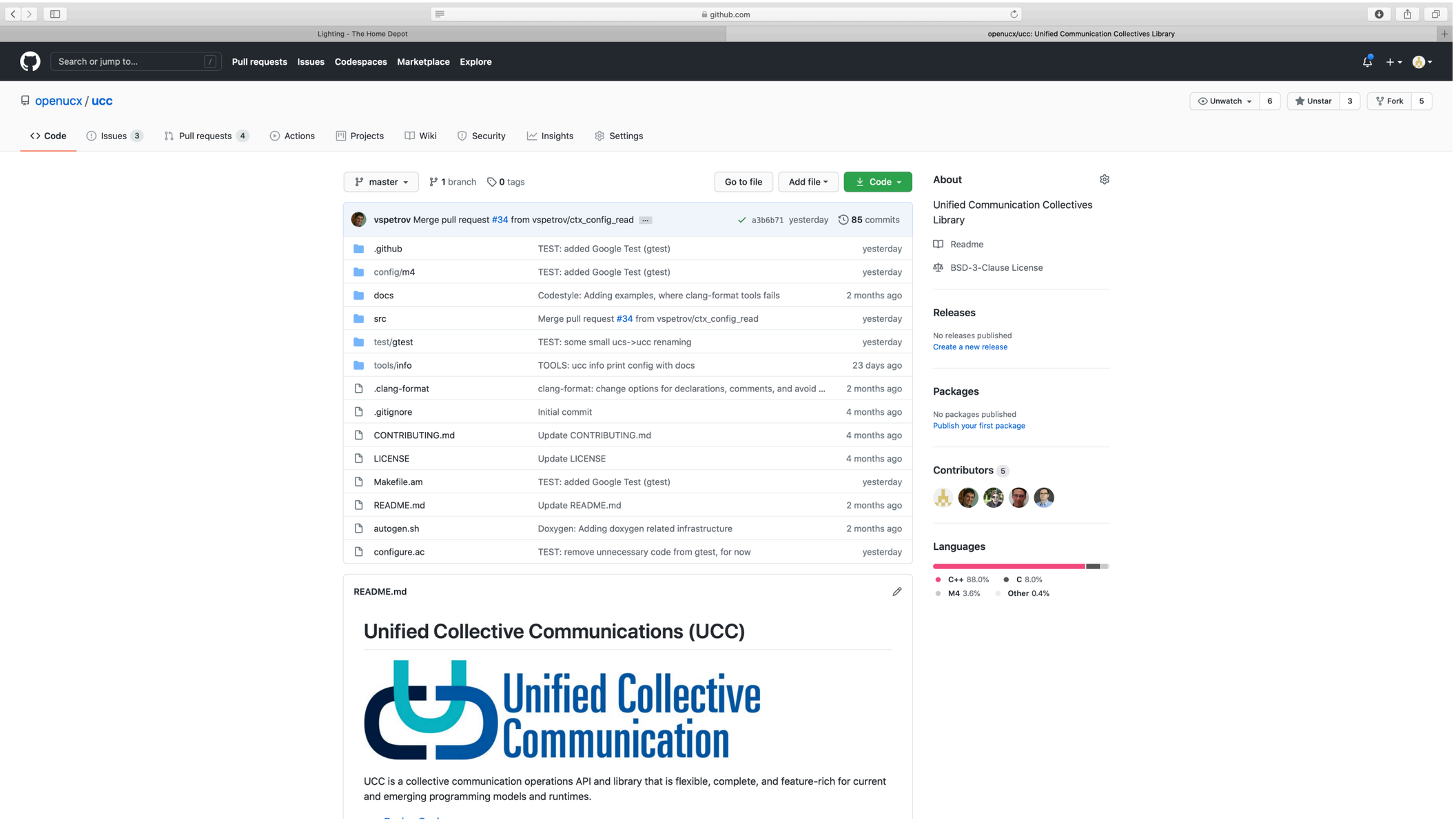
  (Josh/NVIDIA, Srinivas/FaceBook)

UCC Reference Implementation

UCC Reference Implementation: Component Diagram



UCC: Reference Implementation Status



UCC Release Roadmap

UCC v1.0 Expected to Release Q1 2021

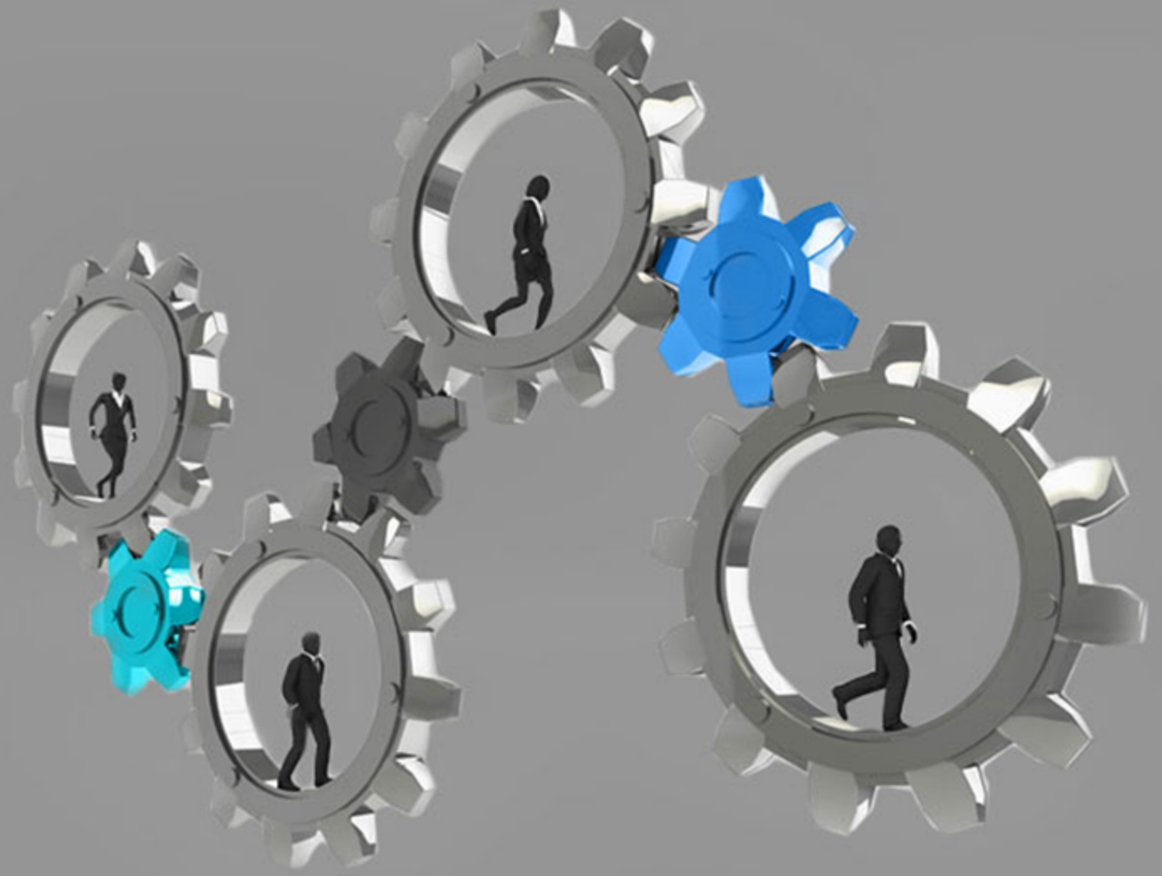
- v1.0 Early Release
  - Specification document: Well defined API and semantics
  - Reference implementation
    - Support with important MPI collectives and fallback for rest
      - Barrier, Broadcast, Allreduce, and Alltoall
    - Multithreading support
  - Support for OpenMPI
  - Support for PyTorch
  - Infrastructure
    - Unit test infrastructure
- v1.0 Stable Release (Target: Q2 2021)
  - Incorporate feedback from the early release
  - MTT for performance and functional testing
  - Performance tests
- v1 Series focusses on performance and stability
- v2.0 release : Advance features and more programming models
  - Task management
  - Algorithm selection
  - Complete GPU support
  - Support for DPUs
  - Support for PGAS collectives
  - Advanced topology support

Plenty of work : Contributions are welcome!

- Acknowledgements
  - Contributions came from many working group members who participate weekly
- What contributions are welcomed ?
  - Everything from design, documentation, code, testing infrastructure, code reviews …
- How to participate ?
  - WG Meetings : https://github.com/openucx/ucc/wiki/UCF-Collectives-Working-Group
  - GitHUB: https://github.com/openucx/ucc
  - Slack channel: Ask for an invite
  - Mailing list: ucx-group@elist.ornl.gov

ENABLER OF CO-DESIGN

UCF



Thank You

The UCF Consortium is a collaboration between industry, laboratories, and academia to create production grade communication frameworks and open standards for data centric and high-performance applications.