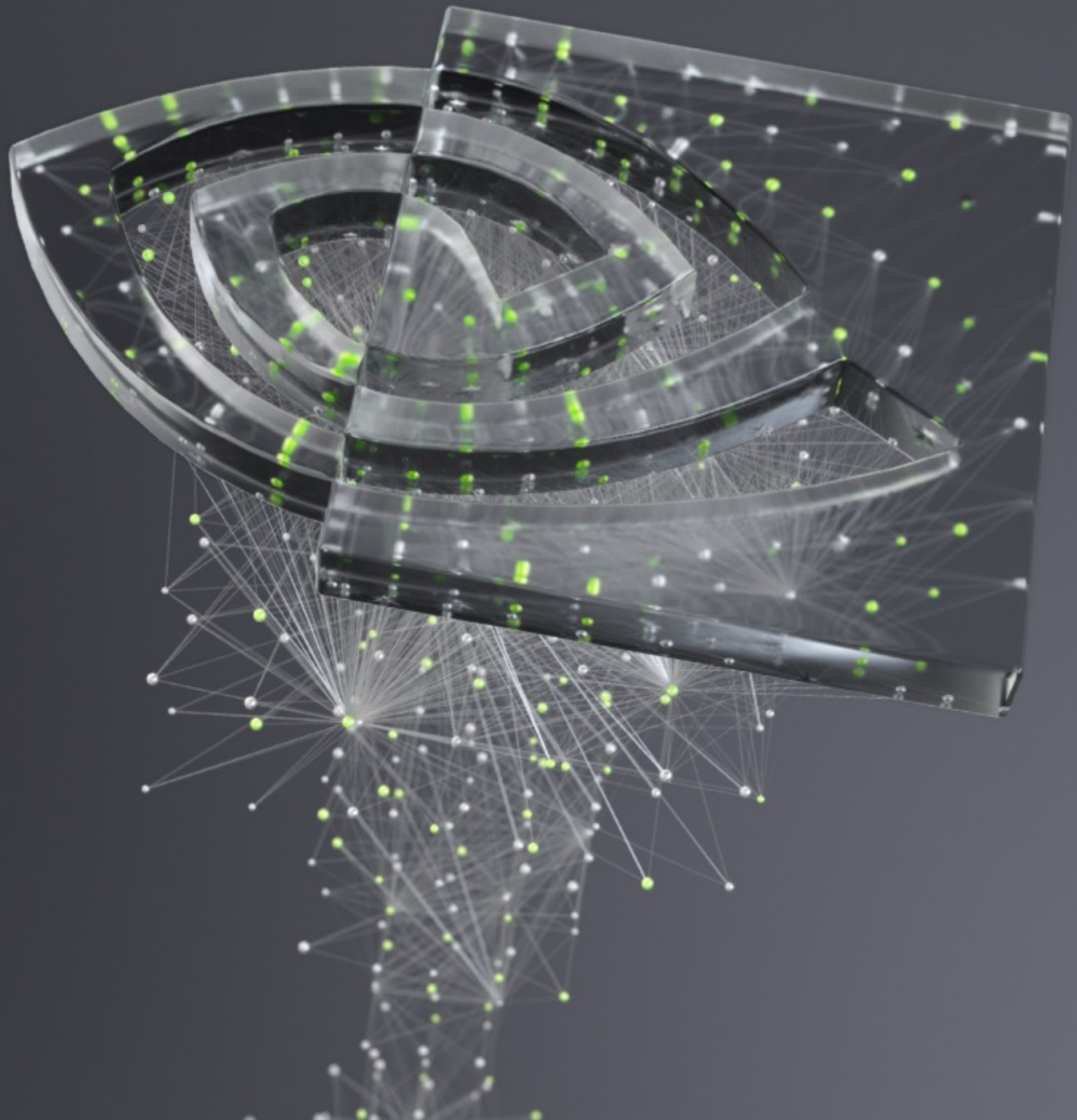


UCC AND SHARP

MANJUNATH GORENTLA VENKATA, NVIDIA

DEVENDAR BUREDDY, NVIDIA

AGENDA

UCC Overview

API, Semantics, and Roadmap

SHARP

Open-source project to provide an API and library implementation of collective (group) communication operations

OUTLINE

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

UCC DESIGN CHALLENGES (1)

- 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 and 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

UCC DESIGN CHALLENGES (2)

- 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 with less overhead
  - One-way execution model – control operations are loosely integrated
    - passive progress, and handle return values (GPU/DPUs)

UCC DESIGN PRINCIPLES: 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

UCC’S SOLUTION

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

- Properties are preferences expressed by the user of the library and what the library actually provides has to be queried
  - Explicit way to request for optional features, semantics, and optimizations (opt-in or opt-out model)
  - Provides an ability to express and request many cross-cutting features

API, ABSTRACTIONS, AND SEMANTICS

CONCEPTS

- Abstractions for Resources
  - Collective Library
  - Communication Context
  - Teams
- Collective Operations
- Triggered Operations

UCC LIBRARY

An object to encapsulate resources related to the group communication operations

Semantics

- All UCC operations should be invoked between the init and finalize operations.
- The library can be tailored to match the user requirements
- The user of the library can be parallel programming models (MPI, PGAS/OpenSHMEM, PyTorch) or applications

Operations

- Routines for initializing and finalizing the resources for the library.

```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 @ref ucc_status_t is returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by @ref 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);
}
```

Library Init C Interface

Properties: Collectives LIBRARY

- Thread Model
- Collective Types
- Reduction Types
- Synchronization Types

```
/**
 *
 *  @ingroup UCC_LIB_INIT_DT
 *
 *  @brief Structure representing the parameters to customize the library
 *
 *  @parblock
 *
 *  Description
 *
 *  @ref ucc_lib_params_t defines the parameters that can be used to customize
 *  the library. The bits in "mask" bit array is defined by @ref
 *  ucc_lib_params_field, which correspond to fields in structure @ref
 *  ucc_lib_params_t. The valid fields of the structure is specified by the
 *  setting the bit to "1" in the bit-array "mask". When bits corresponding to
 *  the fields is not set, the fields are not defined.
 *
 *  @endparblock
 *
 */
typedef struct ucc_lib_params {
    uint64_t                mask;
    ucc_thread_mode_t       thread_mode;
    uint64_t                coll_types;
    uint64_t                reduction_types;
    ucc_coll_sync_type_t    sync_type;
    ucc_reduction_wrapper_t reduction_wrapper;
} ucc_lib_params_t;
```

PROPERTIES OF LIBRARY: THREAD MODEL

- UCC_LIB_THREAD_SINGLE:
  - The user program cannot be multithreaded
- UCC_LIB_THREAD_FUNNELED:
  - The user program may be multithreaded, however, only one thread should invoke the UCC interfaces
- UCC_LIB_THREAD_MULTIPLE:
  - The user program can be multithreaded, and any any thread may invoke the UCC operations.

CONCEPTS

- Abstractions for Resources
  - Collective Library
  - Communication Context
  - Teams
- Collective Operations
- Triggered Operations

COMMUNICATION CONTEXT (1)

An object to encapsulate local resource and express network parallelism

- Context is created by ucc_context_create(), a local operation
- Contexts represents a local resource for group operations - injection queue, and/or network parallelism
  - Example: software injection queues (network endpoints), hardware resources
- Context can be coupled with threads, processes or tasks
  - A single MPI process can have multiple contexts
  - A single thread (pthread or OMP thread) can be coupled with multiple contexts

COMMUNICATION CONTEXT (2)

An object to encapsulate local resource and express network parallelism

- Context can be bound to a specific core, socket, or an accelerator
  - Provides an ability to express affinity
- Context can be used to control resource sharing
- Multiple contexts per team (from same thread) can be supported
  - Software and hardware collectives
  - Optimize for bandwidth utilization

```c
/**
 *  @ingroup UCC_CONTEXT
 *
 *  @brief The @ref ucc_context_create routine creates the context handle.
 *
 *  @param [in]   lib_handle  Library handle
 *  @param [in]   params      Customizations for the communication context
 *  @param [in]   config      Configuration for the communication context to read
 *                            from environment
 *  @param [out]  context     Pointer to the newly created communication context
 *
 *  @parblock
 *
 *  @b Description
 *
 *  The @ref ucc_context_create creates the context and @ref 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
 *  @ref ucc_status_t is returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by @ref 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);
```

Context Create C Interface

PROPERTIES OF CONTEXT : CONTEXT TYPE

Customize for resource sharing and utilization

EXCLUSIVE

- The context participates in a single team
  - So resources are exclusive to a single team
- The libraries can implement it as a lock-free implementation

SHARED

- The context can participate in multiple teams
  - Resources are shared by multiple teams
- The library might be required to protect critical sections

CONCEPTS

- Abstractions for Resources
  - Collective Library
  - Communication Context
  - Teams
- Collective Operations
- Triggered Operations

TEAMS

An object to encapsulate the resources required for group operations such as collective communication operations.

- Created by processes, threads or tasks by calling *ucc_team_create_post()*
  - A collective operation but no explicit synchronization among the processes or threads
- Non-blocking operation and only one active call at any given instance.
- Each process or thread passes local resource object (context)
  - Achieve global agreement during the create operation

```
/**
 *  @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 @ref ucc_status_t is
 *  returned.
 *
 *  @endparblock
 *
 *  @return Error code as defined by @ref 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);
```

Team Create Interface

PROPERTIES: Teams

- Ordering : All team members must invoke collective in the same order?
  - Yes for MPI and No for TensorFlow and Persistent collectives
- Outstanding collectives
  - Can help with resource management
- Should Endpoints be in a contiguous range ?
- Synchronization Model
  - On_Entry, On_Exit, or On_Both – this helps with global resource allocation
- Datatype
  - Can be customized for contiguous, strided, or non-contiguous datatypes

```
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_t     p2p_conn;
    ucc_mem_map_params_t    mem_params;
    ucc_ep_map_t            ep_map;
    uint64_t                id;
} ucc_team_params_t;
```

CONCEPTS

- Abstractions for Resources
  - Collective Library
  - Communication Context
  - Teams
- Collective Operations
- Triggered Operations

COLLECTIVE OPERATIONS: BUILDING BLOCKS

```
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: BUILDING BLOCKS (2)

Semantics

- 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

COLLECTIVE OPERATIONS: BUILDING BLOCKS (3)

- Blocking collectives:
  - Can be implemented with Init_and_post and test+finalize
- Persistent Collectives:
  - Can be implemented using the building blocks - init, post, test, and finalize
- Split-Phase
  - Can be implemented with Init_and_post and test+finalize

CONCEPTS

- Abstractions for Resources
  - Collective Library
  - Communication Context
  - Teams
- Collective Operations
- Triggered Operations

UCC EXECUTION ENGINE, EVENTS, AND TRIGGERED OPERATIONS

Execution Engine

- It is an execution context that supports event-driven network execution on the CUDA streams, CPU threads, and DPU threads.

Events

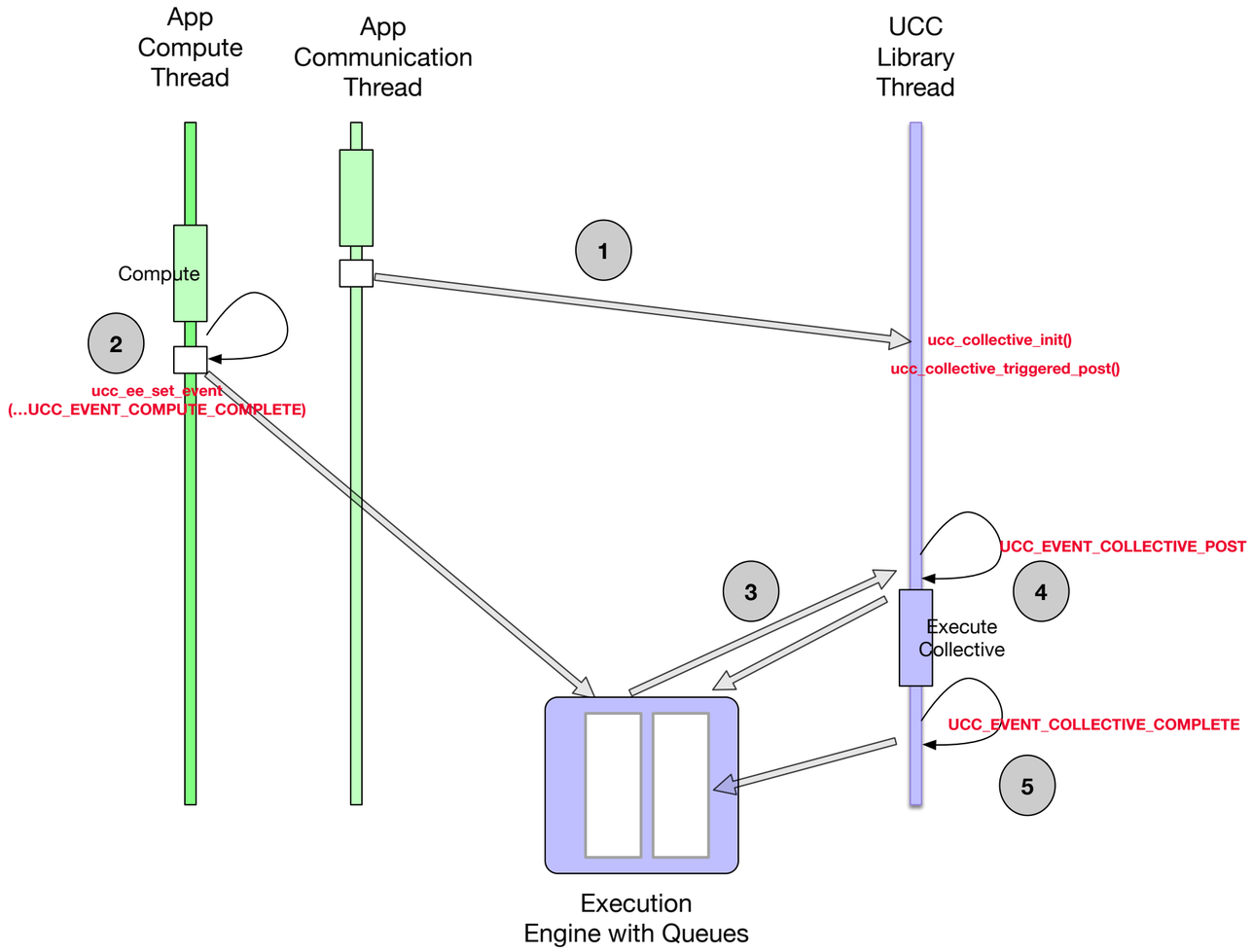
- Library-generated events
  - Examples: Completion of operation, launch of collective
- User-generated events
  - Examples: Compute complete, Data-ready

Triggered Operations

- Triggered operations enable the posting of operations on an event.
  - UCC supports triggering collective operations by library-generated and user-generated events.
- Team-level customization to enable/disable triggered operations

UCC Events: Interaction between a User Thread and Event-driven UCC

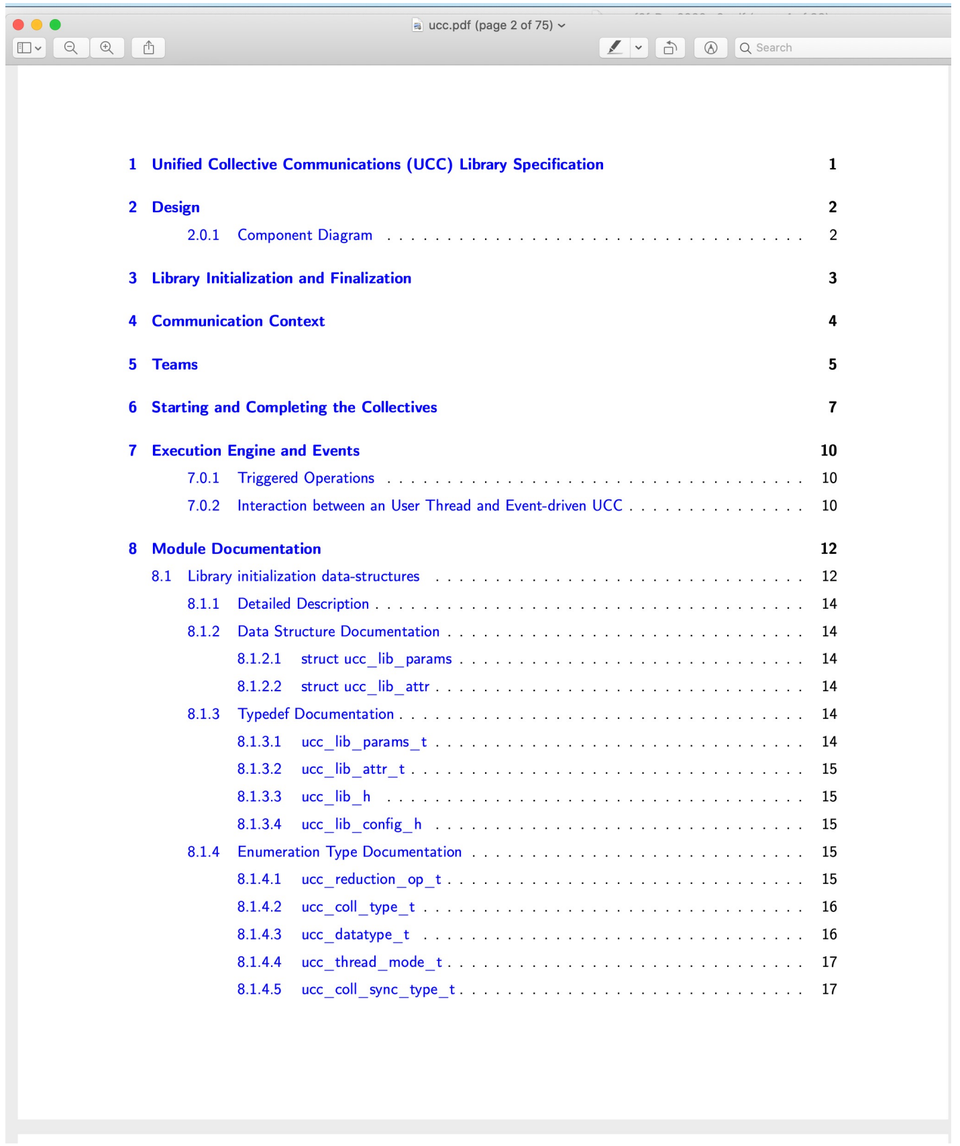
1. Application initializes the collective operation
2. When the application completes the compute, it posts the UCC_EVENT_COMPUTE_COMPLETE event to the execution engine.
3. The library thread polls the event queue and triggers the operations that are related to the compute event.
4. The library posts the UCC_EVENT_POST_COMPLETE event to the event queue.
5. On completion of the collective operation, the library posts UCC_EVENT_COLLECTIVE_COMPLETE event to the completion event queue.

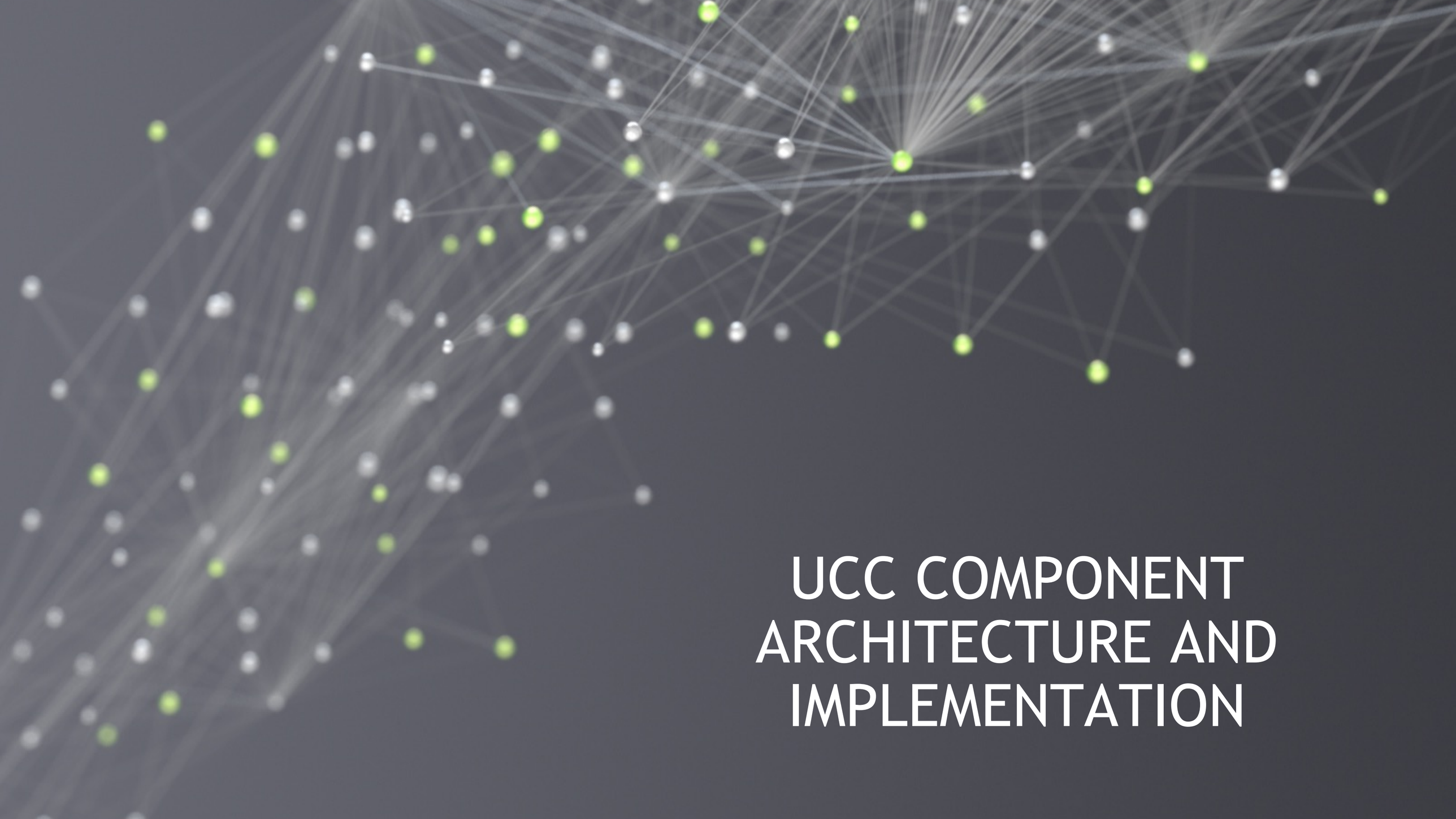


UCC SPECIFICATION

UCC SPECIFICATION: INTERFACES AND SEMANTICS FULLY SPECIFIED

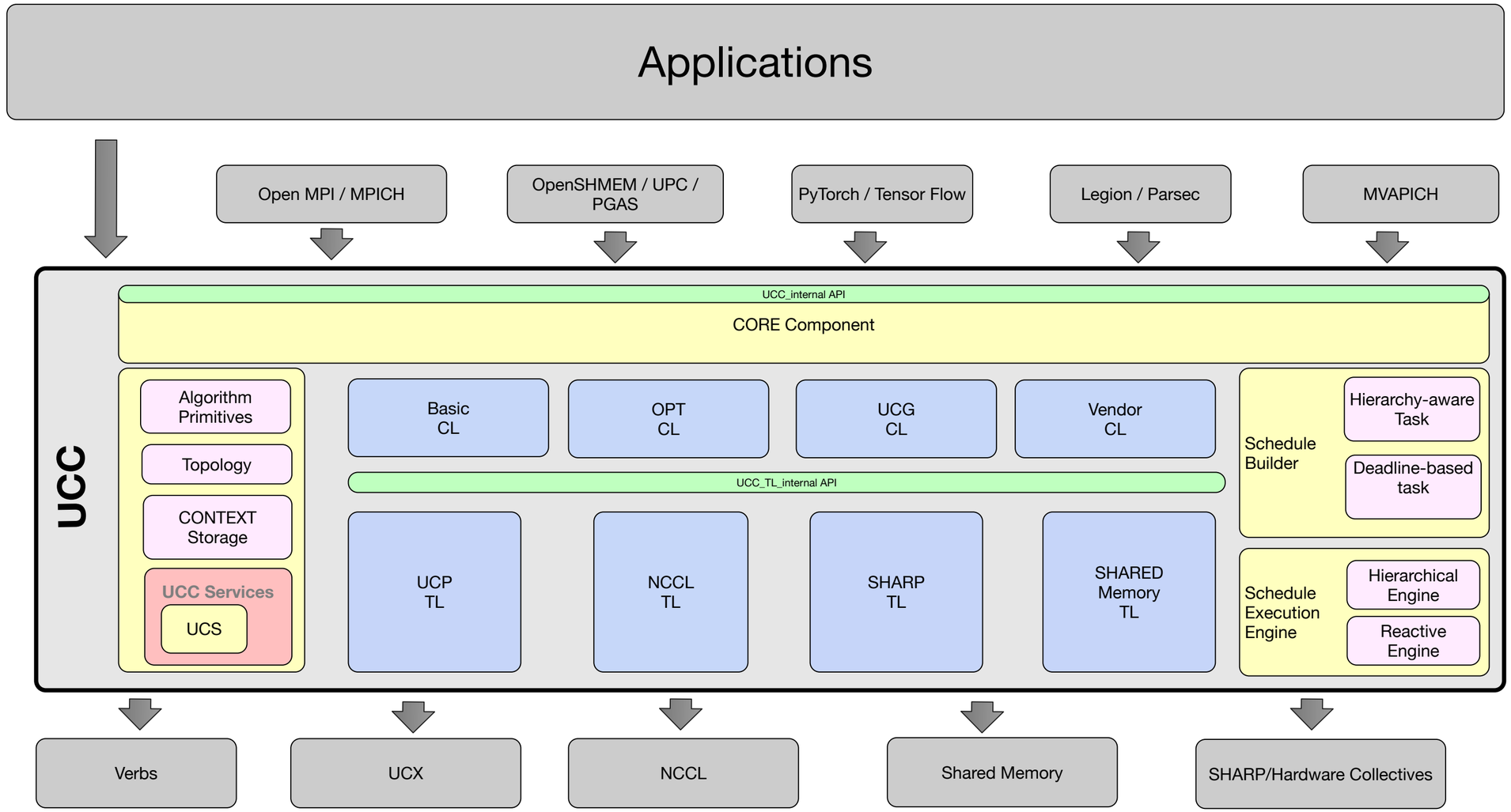
- Specification available on the UCC GH
- Specification is ahead of the code now
- The version 1.0 is agreed by the working group and merged into the master branch
- Over 75 pages of detailed information about the interfaces and semantics
- Doxygen based documentation
- Both pdf and html available



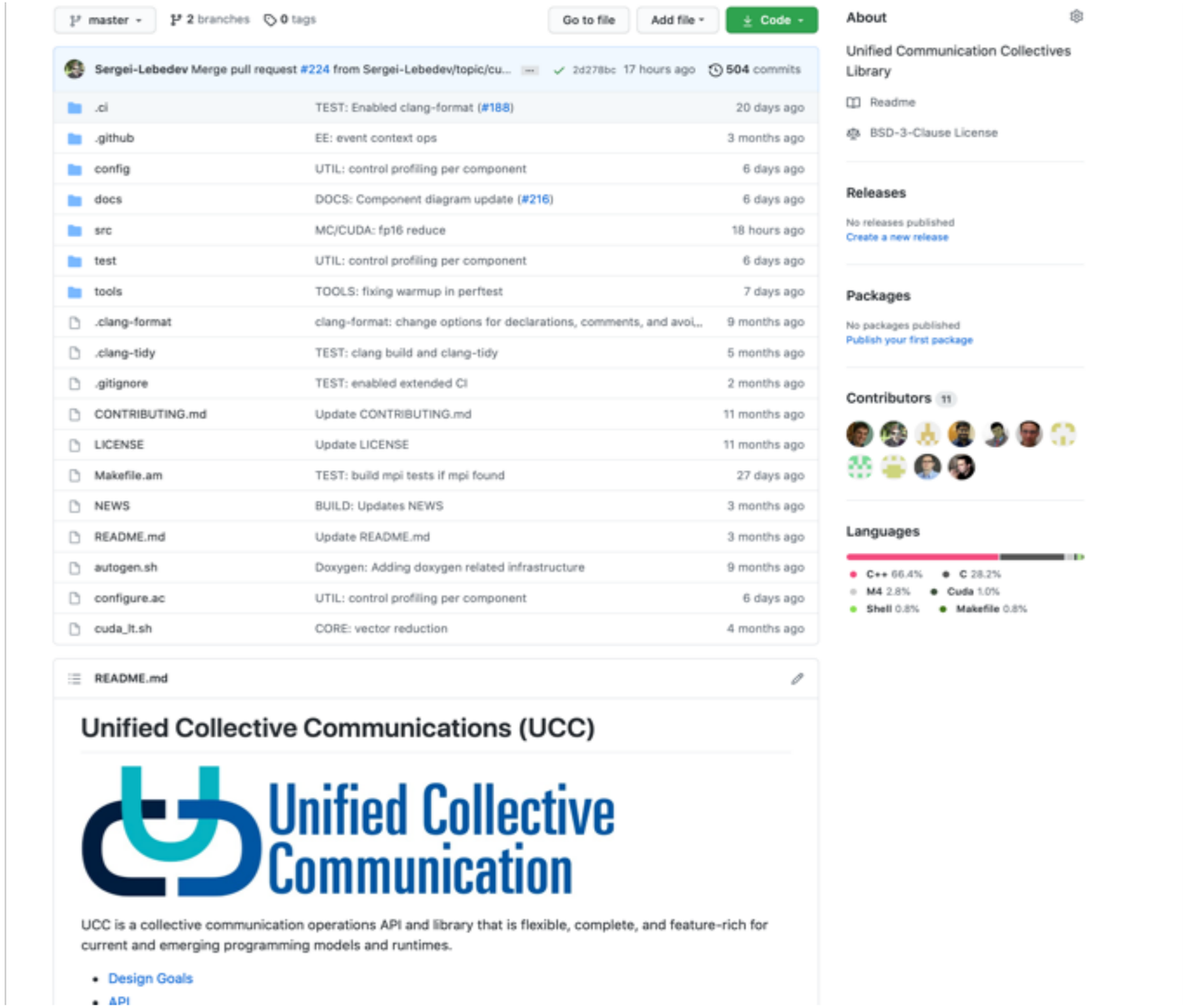


UCC COMPONENT ARCHITECTURE AND IMPLEMENTATION

UCC REFERENCE IMPLEMENTATION: COMPONENT DIAGRAM



UCC: REFERENCE IMPLEMENTATION STATUS



master · 2 branches · 0 tags · Go to file · Add file · Code

Sergei-Lebedev Merge pull request #224 from Sergei-Lebedev/topic/cu... 2d278bc 17 hours ago 504 commits

.ci	TEST: Enabled clang-format (#188)	20 days ago
.github	EE: event context ops	3 months ago
config	UTIL: control profiling per component	6 days ago
docs	DOCS: Component diagram update (#216)	6 days ago
src	MC/CUDA: fp16 reduce	18 hours ago
test	UTIL: control profiling per component	6 days ago
tools	TOOLS: fixing warmup in perftest	7 days ago
.clang-format	clang-format: change options for declarations, comments, and avoi...	9 months ago
.clang-tidy	TEST: clang build and clang-tidy	5 months ago
.gitignore	TEST: enabled extended CI	2 months ago
CONTRIBUTING.md	Update CONTRIBUTING.md	11 months ago
LICENSE	Update LICENSE	11 months ago
Makefile.am	TEST: build mpi tests if mpi found	27 days ago
NEWS	BUILD: Updates NEWS	3 months ago
README.md	Update README.md	3 months ago
autogen.sh	Doxygen: Adding doxygen related infrastructure	9 months ago
configure.ac	UTIL: control profiling per component	6 days ago
cuda_lt.sh	CORE: vector reduction	4 months ago

README.md

Unified Collective Communications (UCC)

Unified Collective Communication

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

- Design Goals
- API

About

Unified Communication Collectives Library

Readme

BSD-3-Clause License

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 11

Languages

C++ 66.4% · C 28.2% · M4 2.8% · Cuda 1.0% · Shell 0.8% · Makefile 0.8%

UCC RELEASE ROADMAP

UCC V1.0 EXPECTED TO RELEASE Q3 2021

v0.1.0 Early Release (Branched Q1 2021)

- Support for most collectives required by parallel programming models
- Many algorithms to support various data sizes, types, and system configurations
- Support for CPU and GPU collectives
- Testing infrastructure
  - Unit tests, profiling, and performance tests
- Support for MPI and PyTorch (via Third-party plugin)

v1.0 Stable Release (Expected SC 2021)

- Incorporate feedback from v0.1.0 release
- Support for OpenSHMEM with one-sided collectives and active sets
- Hardware collectives - support for SHARP
- Support for more optimized collectives (hierarchical/ reactive)
- Infrastructure for pipelining, task management , and customization (algorithm selection)
- Persistent collectives

v1.x Series: Focus on stability, performance and scalability

- Support for DPUs and DPAs
- Partitioned collectives
- OpenSHMEM Teams and nonblocking collectives

CONTRIBUTIONS ARE WELCOME!

- 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

SHARP: IN-NETWORK SCALABLE STREAMING HIERARCHICAL AGGREGATION AND REDUCTION PROTOCOL

IN NETWORK COMPUTING

Offload, Co-design and In-network Computing

- Offload – Have someone else do the work
  - Move functionality from the CPU to the network
- Co-Design – Re-thinking the boundaries between different components
  - Move functionality from SW to HW / end node to switches
- In-Network Computing – Move traditionally compute operations to the network
  - A type of Co-Design

SCALABLE HIERARCHICAL AGGREGATION AND REDUCTION PROTOCOL (SHARP)

In-network Tree based aggregation mechanism

Multiple simultaneous outstanding operations

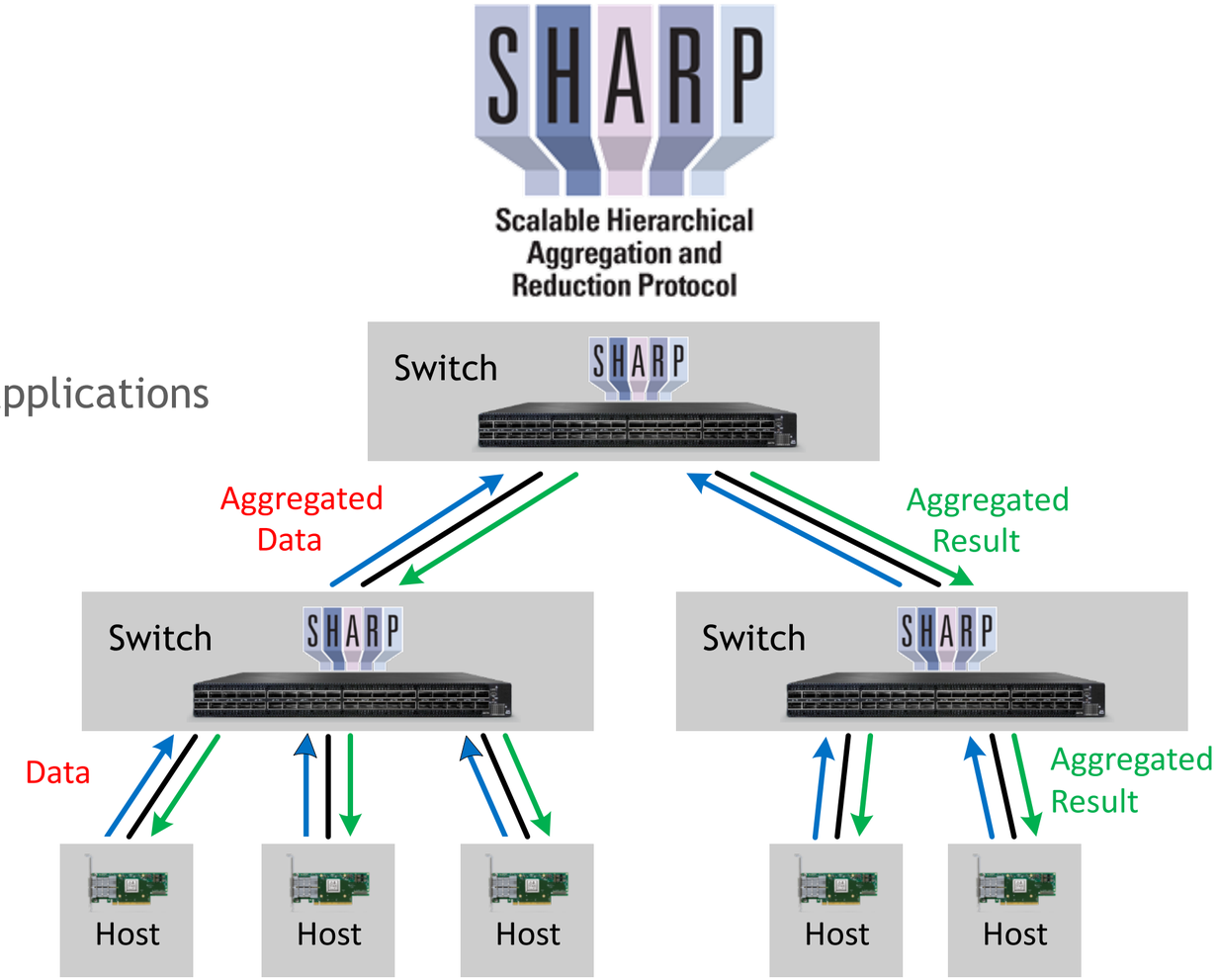
For HPC (MPI / SHMEM) and Distributed Machine Learning applications

Scalable High Performance Collective Offload

Barrier, Reduce, All-Reduce, Broadcast and more

Sum, Min, Max, Min-loc, max-loc, OR, XOR, AND

Integer and Floating-Point, 16/32/64 bits



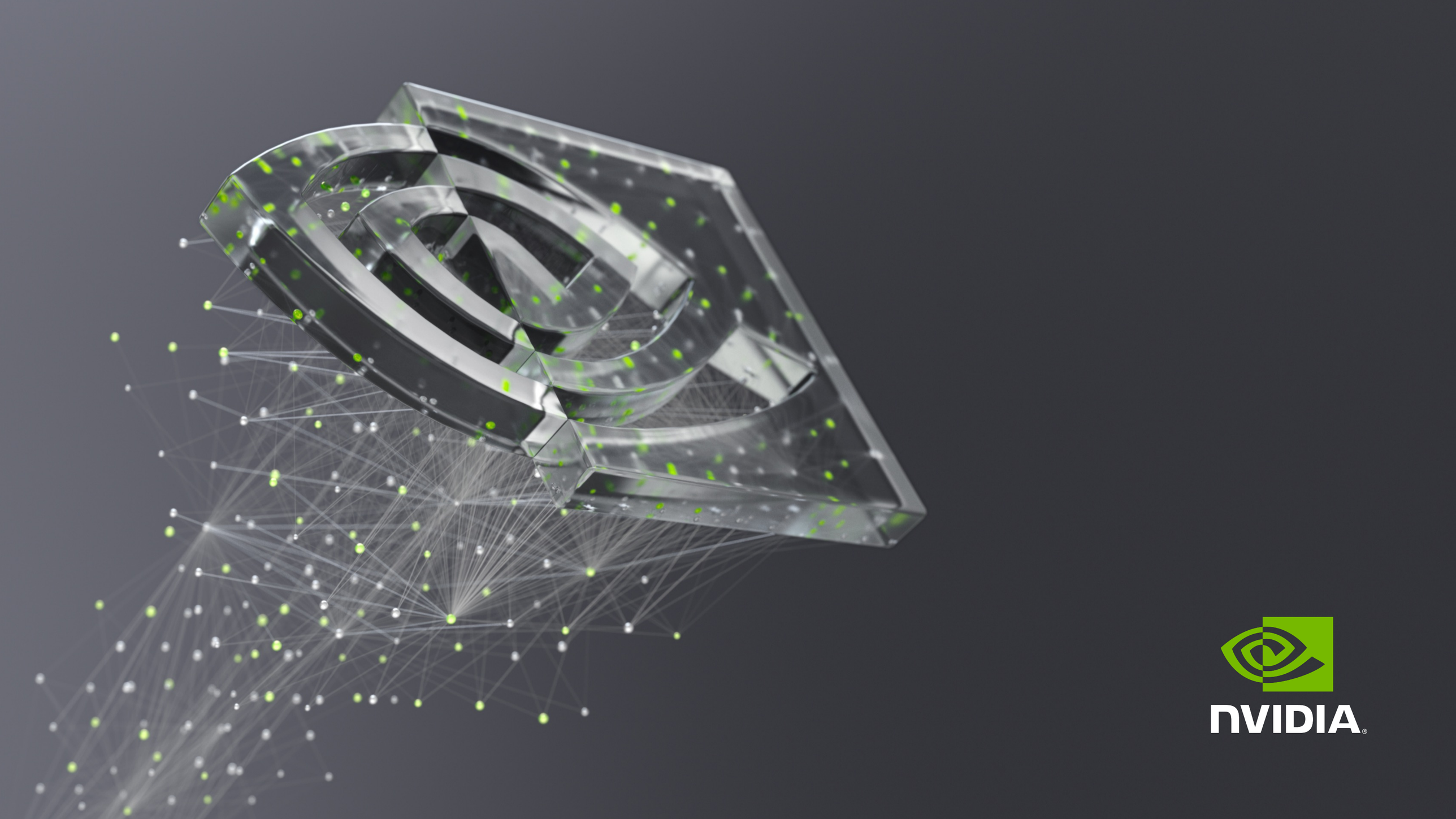
SHARP 2.0

- SHARP v2.0 HDR Quantum switch
  - Support for small vector reductions
  - Improved latency reduction for small vectors (LLT – low latency trees)
  - Support for large vector reductions – perform reductions at line rate (SAT – streaming aggregation trees)
    - Support for two simultaneous streaming operations per switch (limited resource)
  - **Works together with GPUDirect RDMA**
  - **SAT killer app is distributed, synchronous deep learning workloads**
    - Distributed stochastic gradient descent
    - Limiter is large vector allreduce / bandwidth – gradient averaging between nodes
    - Mlperf – v1.0 - Record-Setting NVIDIA Performance with SHARP
      - https://developer.nvidia.com/blog/mlperf-v1-0-training-benchmarks-insights-into-a-record-setting-performance/

SHARP

New Features

- SHARP for Cloud
  - UCX for in-band communication between daemons(sharpd/am)
  - PKEY support
  - GRH support
- Exclusive lock
- Event reporting
- Connection keep alive between Daemons
- Topology API


NVIDIA