UCX

Unified Communication - X Framework

Presented by Pavel Shamis

UCX Framework Mission

- Collaboration between industry, laboratories, and academia
- To create open-source production grade communication framework for data centric and HPC applications
- To enable the highest performance through co-design of software-hardware interfaces

Background

MXM

- Developed by Mellanox Technologies
- HPC communication library for InfiniBand devices and shared memory
- Primary focus: MPI, PGAS

UCCS

- Developed by ORNL, UH, UTK
- Originally based on Open MPI BTL and OPAL layers
- HPC communication library for InfiniBand, Cray Gemini/Aries, and shared memory
- Primary focus: OpenSHMEM, PGAS
- Also supports: MPI

PAMI

- Developed by IBM on BG/Q, PERCS, IB VERBS
- Network devices and shared memory
- MPI, OpenSHMEM, PGAS, CHARM++, X10
- C++ components
- Aggressive multi-threading with contexts
- Active Messages
- Non-blocking collectives with hw accleration support

UCX is an Integration of Industry and Users Design Efforts

UCX Goals

API

Exposes broad semantics that target data centric and HPC programming models and applications

Performance oriented

Optimization for low-software overheads in communication path allows near native-level performance

Production quality

Developed, maintained, tested, and used by industry and researcher community

Community driven

Collaboration between industry, laboratories, and academia

Research

The framework concepts and ideas are driven by research in academia, laboratories, and industry

Cross platform

Support for Infiniband, Cray, various shared memory (x86-64 and Power), GPUs

Co-design of Exascale Network APIs

Members

Mellanox co-designs network interface and contributes MXM technology

- Infrastructure, UD, RC, DCT, shared memory, protocols, integration with OpenMPI/ SHMEM, MPICH

ORNL co-designs network interface and contributes UCCS project

- IB optimizations, support for Cray devices, shared memory

NVIDIA co-designs high-quality support for GPU devices

- GPU-Direct, GDR copy, etc.

IBM co-designs network interface and contributes ideas and concepts from PAMI

UH/UTK focus on integration with their research platforms

What’s new about UCX?

- **Simple and consistent API**
- Choosing between low-level and high-level API allows easy integration with a wide range of applications and middleware.
- Protocols and transports are selected by capabilities and performance estimations, rather than hard-coded definitions.
- Support thread contexts and dedicated resources, as well as fine-grained and coarse-grained locking.
- Accelerators are represented as a transport, driven by a generic “glue” layer, which will work with all communication networks.

The UCX Framework

UC-S for Services

This framework provides basic infrastructure for component based programming, memory management, and useful system utilities

Functionality:
Platform abstractions, data structures, debug facilities.

UC-T for Transport

Low-level API that expose basic network operations supported by underlying hardware. Reliable, out-of-order delivery.

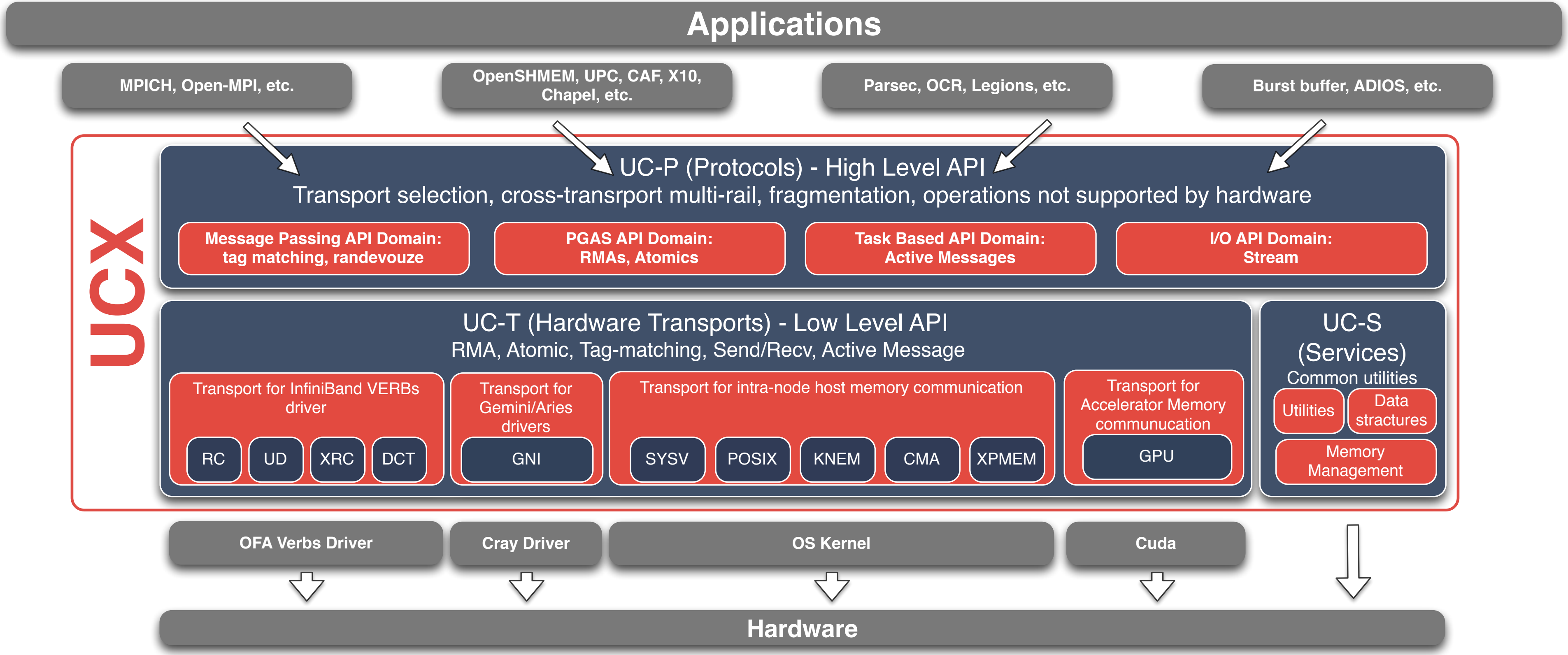
Functionality:
Setup and instantiation of communication operations.

UC-P for Protocols

High-level API uses UCT framework to construct protocols commonly found in applications

Functionality:
Multi-rail, device selection, pending queue, rendezvous, tag-matching, software-atomics, etc.

High-level Overview



Clarifications

- UCX is not a device driver
- UCX is a framework for communications
    - Close-to-hardware API layer
    - Providing an access to hardware's capabilities
- UCX relies on drivers supplied by vendors

Project Management

- API definitions and changes are discussed within developers (mail-list, github, conf call)
- PRs with API change have to be approved by ALL maintainers
- PR within maintainer “domain” has to be reviewed by the maintainer or team member (Example: Mellanox reviews all IB changes)

Licensing

- BSD 3 Clause license
- Contributor License Agreement – BSD 3 based

UCX Advisory Board

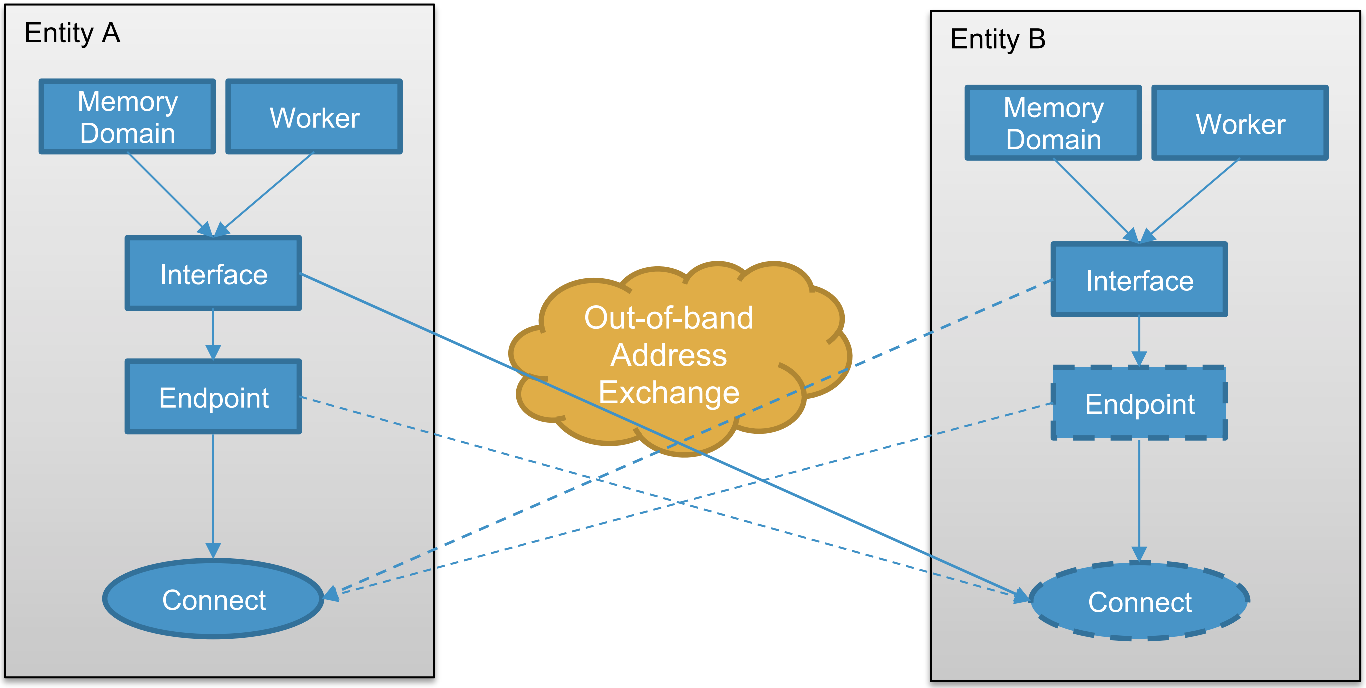
- Arthur Barney Maccabe (ORNL)
- Bronis R. de Supinski (LLNL)
- Donald Becker (NVIDIA)
- George Bosilca (UTK)
- Pavan Balaji (ANL)
- Richard Graham (Mellanox Technologies)
- Sameer Kumar (IBM)
- Stephen Poole (Open Source Software Solutions)
- Gilad Shainer (Mellanox Technologies)
- Sameh Sharkawi (IBM)

API Overview

UCT (Transport layer) objects

- **uct_worker_h**
  A context for separate progress engine and communication resources. Can be either thread-dedicated or shared.
- **uct_pd_h** (will be renamed to uct_md_h)
  Memory registration domain. Can register user buffers and/or allocate registered memory.
- **uct_iface_h**
  Communication interface, created on a specific memory domain and worker. Handles incoming active messages and spawns connections to remote interfaces.
- **uct_ep_h**
  Connection to a remote interface. Used to initiate communications.

UCT initialization



UCT memory primitives

- Register memory within the domain
  - Can potentially use a cache to speedup registration
- Allocate registered memory.
- Pack memory region handle to a remote-key-buffer
  - Can be sent to another entity.
- Unpack a remote-key-buffer into a remote-key
  - Can be used for remote memory access.

UCT communication primitives

- Not everything has to be supported.
  - Interface reports the set of supported primitives.
  - UCP uses this info to construct protocols.
- Send active message (active message id)
- Put data to remote memory (virtual address, remote key)
- Get data from remote memory (virtual address, remote key)
- Perform an atomic operation on remote memory:
  - Add
  - Fetch-and-add
  - Swap
  - Compare-and-swap
- Insert a fence
- Flush pending communications

UCT data types

- UCT communications have a size limit
  - Interface reports max. allowed size for every operation.
  - Fragmentation, if required, should be handled by user / UCP.

- Several data “classes” are supported:
  - “short” – small buffer.
  - “bcopy” – a user callback which generates data (in many cases, “memcpy” can be used as the callback).
  - “zcopy” – a buffer and it’s memory region handle. Usually large buffers are supported.

- Atomic operations use a 32 or 64 bit immediate values.

UCT completion semantics

- All operations are non-blocking
- Return value indicates the status:
  - OK – operation is completed.
  - INPROGRESS – operation has started, but not completed yet.
  - NO_RESOURCE – cannot initiate the operation right now. The user might want to put this on a pending queue, or retry in a tight loop.
  - ERR_xx – other errors.
- Operations which may return INPROGRESS (get/atomics/zcopy) can get a completion handle.
  - User initializes the completion handle with a counter and a callback.
  - Each completion decrements the counter by 1, when it reaches 0 – the callback is called.

UCT API

```c
typedef ucs_status_t (*uct_am_callback_t)(void *arg, void *data, size_t length,
                                          void *desc);

typedef void (*uct_pack_callback_t)(void *dest, void *arg, size_t length);

typedef void (*uct_completion_callback_t)(uct_completion_t *self);

typedef struct uct_completion uct_completion_t;
struct uct_completion {
    uct_completion_callback_t func;
    int                       count;
};

typedef uintptr_t uct_rkey_t;
typedef void * uct_mem_h;
```

```c
ucs_status_t uct_ep_put_short(uct_ep_h ep, const void *buffer, unsigned length,
                              uint64_t remote_addr, uct_rkey_t rkey);

ucs_status_t uct_ep_put_bcopy(uct_ep_h ep, uct_pack_callback_t pack_cb,
                              void *arg, size_t length, uint64_t remote_addr,
                              uct_rkey_t rkey);

ucs_status_t uct_ep_put_zcopy(uct_ep_h ep, const void *buffer, size_t length,
                              uct_mem_h memh, uint64_t remote_addr,
                              uct_rkey_t rkey, uct_completion_t *comp);

ucs_status_t uct_ep_am_short(uct_ep_h ep, uint8_t id, uint64_t header,
                             const void *payload, unsigned length);

ucs_status_t uct_ep_atomic_cswap64(uct_ep_h ep, uint64_t compare, uint64_t swap,
                                   uint64_t remote_addr, uct_rkey_t rkey,
                                   uint64_t *result, uct_completion_t *comp);
```

Currently supported transports

- Shared memory
  - SystemV
  - CMA
  - KNEM
- uGNI (RMA API and ATOMICS)
- IB
  - RC
  - UD (still WIP)
  - CM (for wireup only)

UCP (protocol layer)

- Mix-and-match transports, devices, and operations, for optimal performance.
    - Based on UCT capabilities and performance estimations.

- Enforce ordering when required (e.g tag matching)

- Work around UCT limitations:
    - Fragmentation
    - Emulate unsupported operations
    - Expose one-sided connection establishment

UCP objects

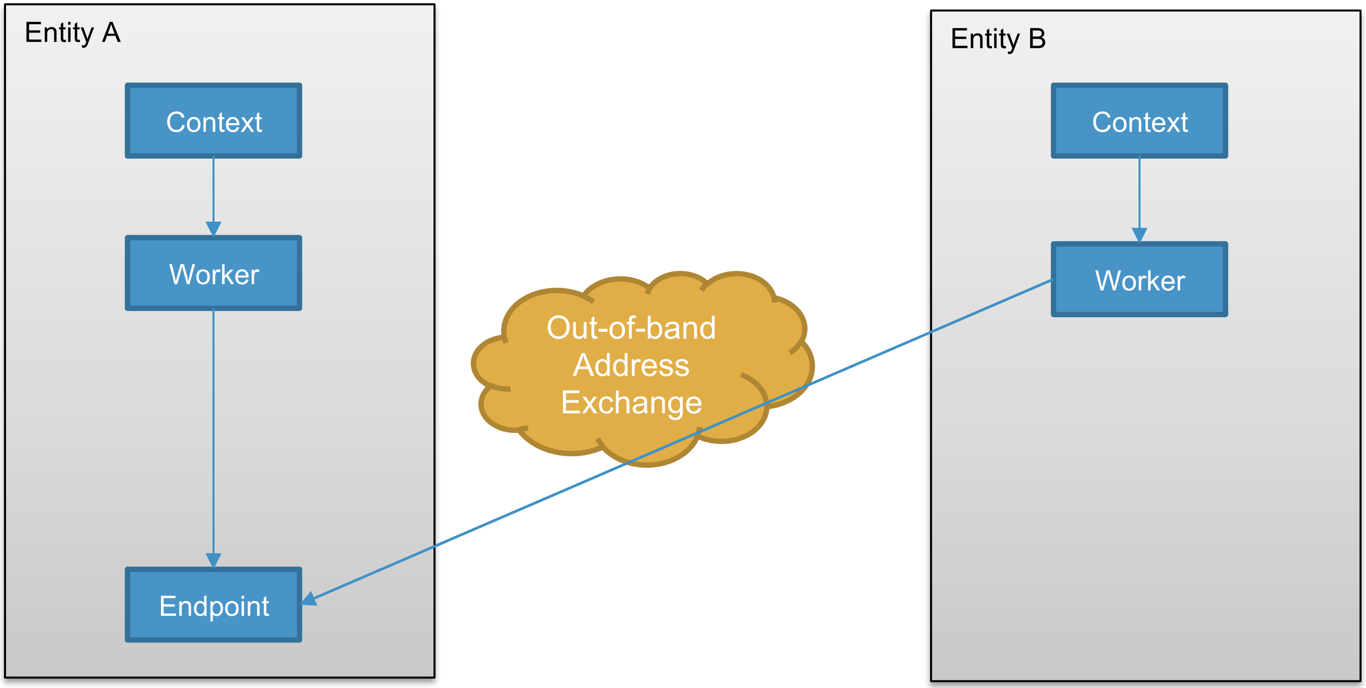
- ucp_context_h

  A global context for the application. For example, hybrid MPI/SHMEM library may create on context for MPI, and another for SHMEM.
- ucp_worker_h

  Communication resources and progress engine context. One possible usage is to create one worker per thread.
- ucp_ep_h

  Connection to a remote worker. Used to initiate communications.

UCP initialization



UCP communications

- Tag-matched send/receive
  - Blocking / Non-blocking
  - Standard / Synchronous / Buffered

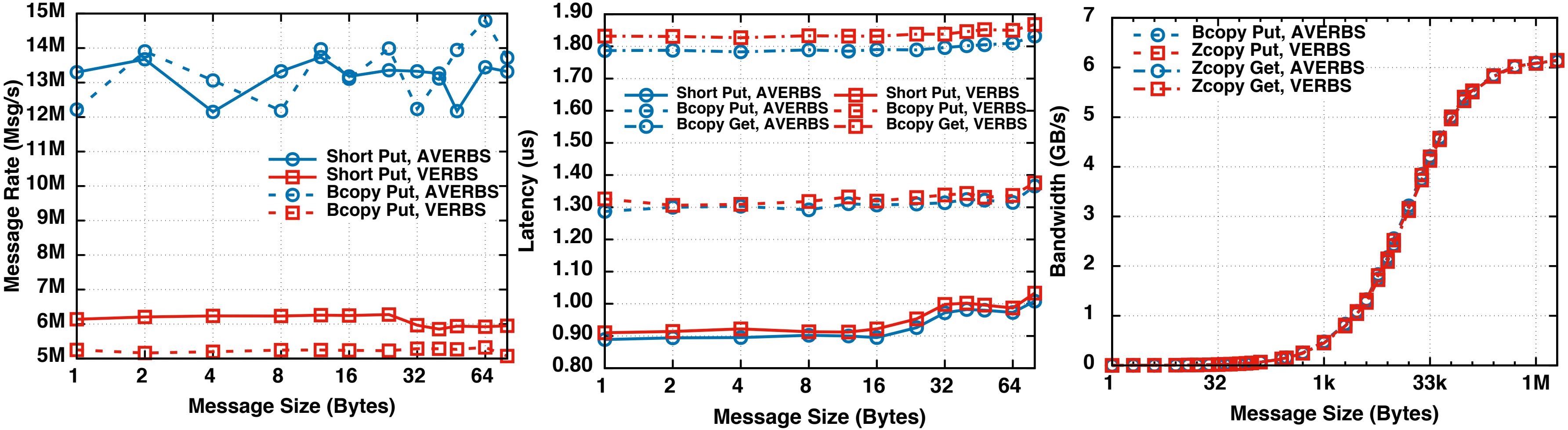
- Remote memory operations
  - Blocking put, get, atomics
  - Non-blocking – TBD

- Data is specified as buffer and length
  - No size limit
  - May register the buffer and use zero copy

UCP API

```
typedef uint64_t ucp_tag_t;



ucs_status_t ucp_worker_create(ucp_context_h context, ucs_thread_mode_t thread_mode,
                               ucp_worker_h *worker_p);

ucs_status_t ucp_worker_get_address(ucp_worker_h worker, ucp_address_t **address_p,
                                    size_t *address_length_p);

ucs_status_t ucp_ep_create(ucp_worker_h worker, ucp_address_t *address,
                           ucp_ep_h *ep_p);



ucs_status_t ucp_put(ucp_ep_h ep, const void *buffer, size_t length,
                     uint64_t remote_addr, ucp_rkey_h rkey);

ucs_status_t ucp_get(ucp_ep_h ep, void *buffer, size_t length,
                     uint64_t remote_addr, ucp_rkey_h rkey);
```

Preliminary Evaluation (UCT)



Pavel Shamis, Manjunath Gorentla Venkata, M. Graham Lopez, Matthew B. Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L. Graham, Liran Liss, Yiftah Shahar, Sreeram Potluri, Davide Rossetti, Donald Becker, Duncan Poole, Christopher Lamb, Sameer Kumar, Craig Stunkel, George Bosilca, Aurelien Bouteiller, "UCX: An Open Source Framework for HPC Network APIs and Beyond", HOTI 2015

OPENFABRICS ALLIANCE

Acknowledgments

Department of Defense — United States of America

Oak Ridge National Laboratory

UCX

Unified Communication - X
Framework

WEB:
www.openucx.org
Mailing List:
https://elist.ornl.gov/mailman/listinfo/ucx-group
ucx-group@elist.ornl.gov