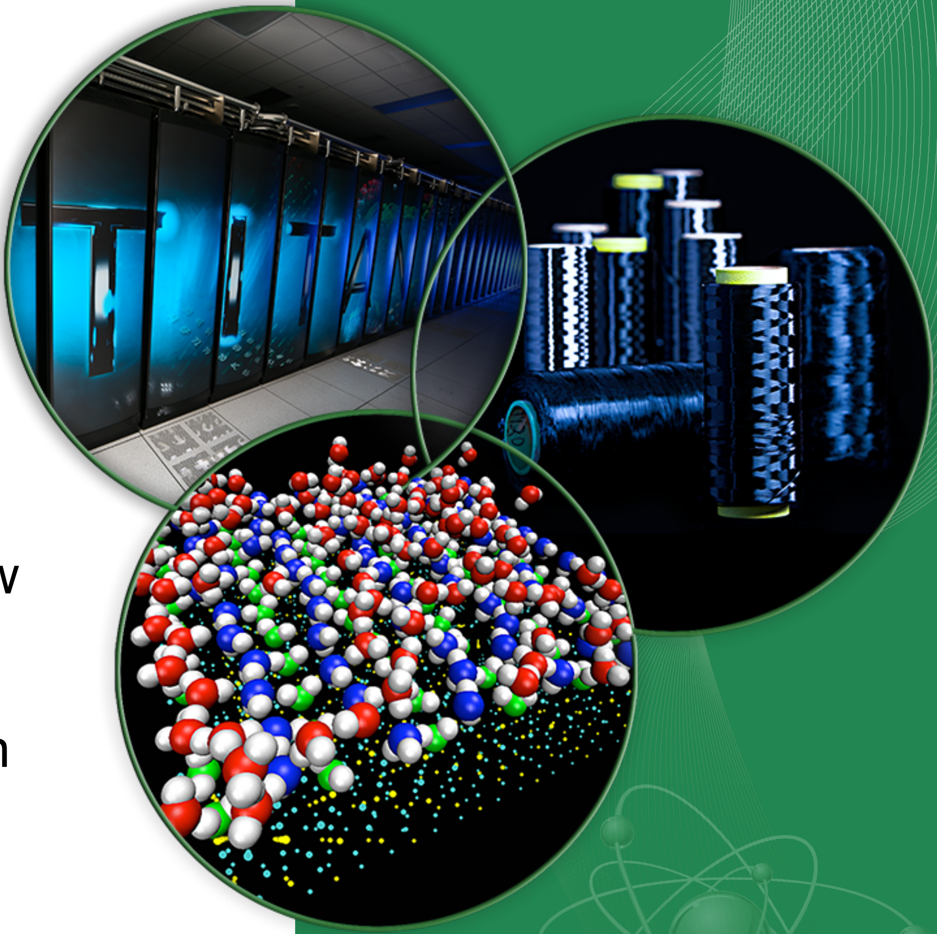
UCX: An Open Source Framework for HPC Network APIs and Beyond

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

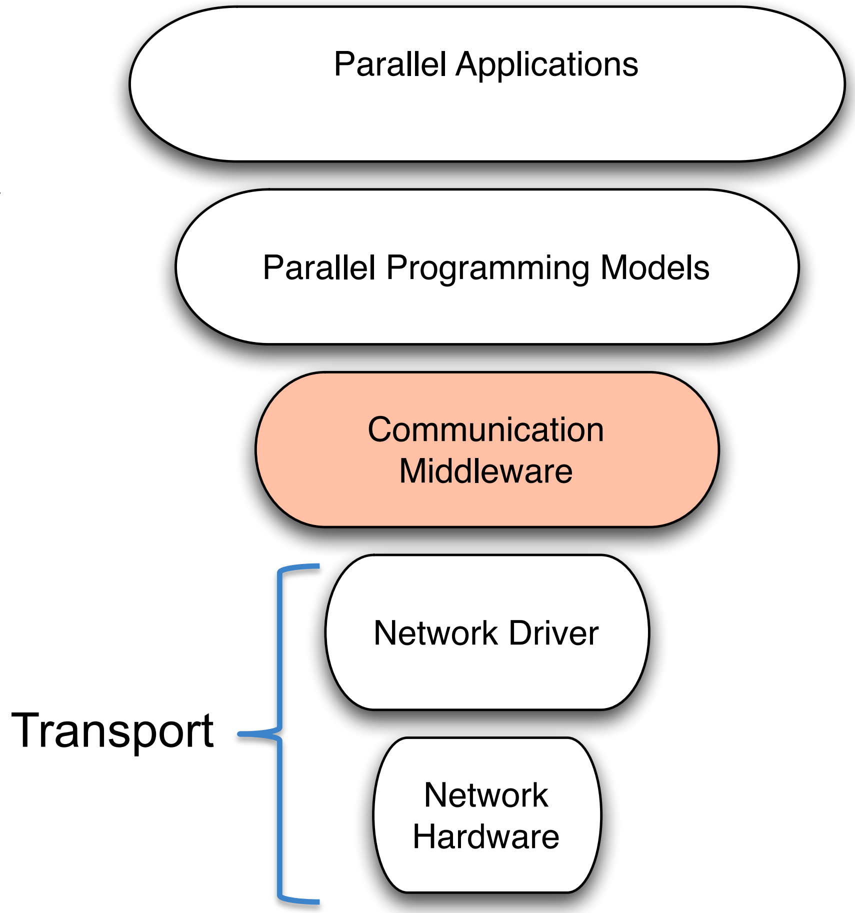
Presented by: **Pavel Shamis / Pasha**



OAK RIDGE National Laboratory

Outline

- Background
- UCX Framework Overview
  - Goals
  - Architecture
  - Preliminary Evaluation



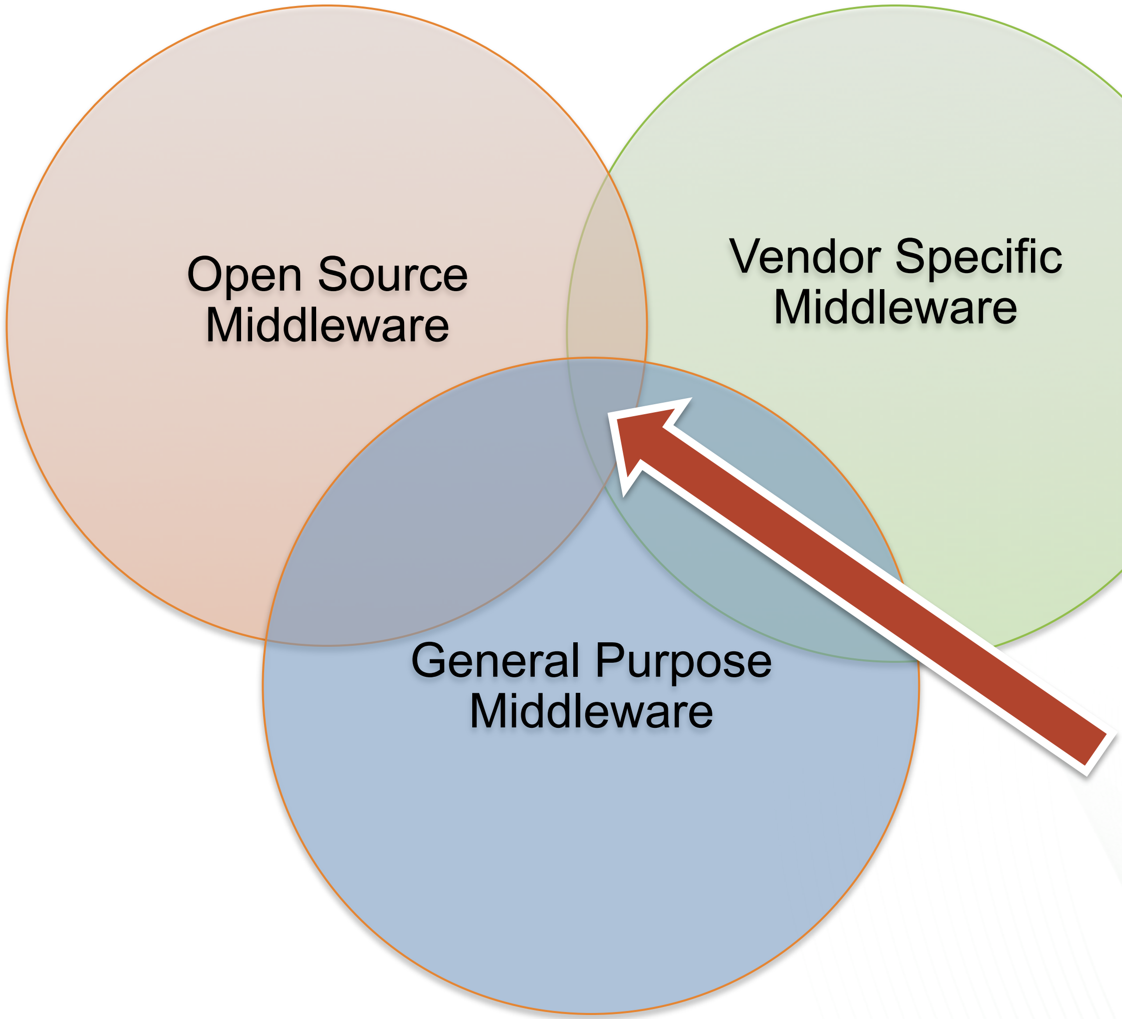
Background

- Special purpose: GASNet, ARMCI, CCI, etc.
- Vendors specific: MXM, PSM, PAMI, uGNI / DMAPP
- Network Architectures: Portals
- Low Level: Verbs, Libfabrics, DPDK, UCCS

The fact is that MPIs, PGAS languages, I/O libraries have to maintain and re-implement thousands of code-lines for various network technologies and programming languages

- Actually, the classification is very tricky…

Background


Open Source Middleware
Vendor Specific Middleware
General Purpose Middleware

Challenges

- Performance Portability
  - Collaboration between industry and research institutions
    - …but mostly industry (because they built the hardware)
- Maintenance
  - Maintaining a network stack is time consuming and expensive
  - Industry have resources and strategic interest for this
- Extendibility
  - MPI+X+Y ?
  - Exascale programming environment is an ongoing debate

Challenges (CORAL)

Feature	Summit	Titan
Application Performance	5-10x Titan	Baseline
Number of Nodes	~3,400	18,688
Node performance	> 40 TF	1.4 TF
Memory per Node	>512 GB (HBM + DDR4)	38GB (GDDR5+DDR3)
NVRAM per Node	800 GB	0
Node Interconnect	NVLink (5-12x PCIe 3)	PCIe 2
System Interconnect (node injection bandwidth)	Dual Rail EDR-IB (23 GB/s)	Gemini (6.4 GB/s)
Interconnect Topology	Non-blocking Fat Tree	3D Torus
Processors	IBM POWER9 NVIDIA Volta™	AMD Opteron™ NVIDIA Kepler™
File System	120 PB, 1 TB/s, GPFS™	32 PB, 1 TB/s, Lustre®
Peak power consumption	10 MW	9 MW

OAK RIDGE National Laboratory

UCX – Unified Communication X Framework

- Unified
  - Multiple Application and Programing Modes (HPC focused)
  - Multiple network architectures
- Communication
  - How to move data from location in memory A to location in memory B considering multiple types of memories
- Framework
  - A collection of libraries and utilities for HPC network programmers

History

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

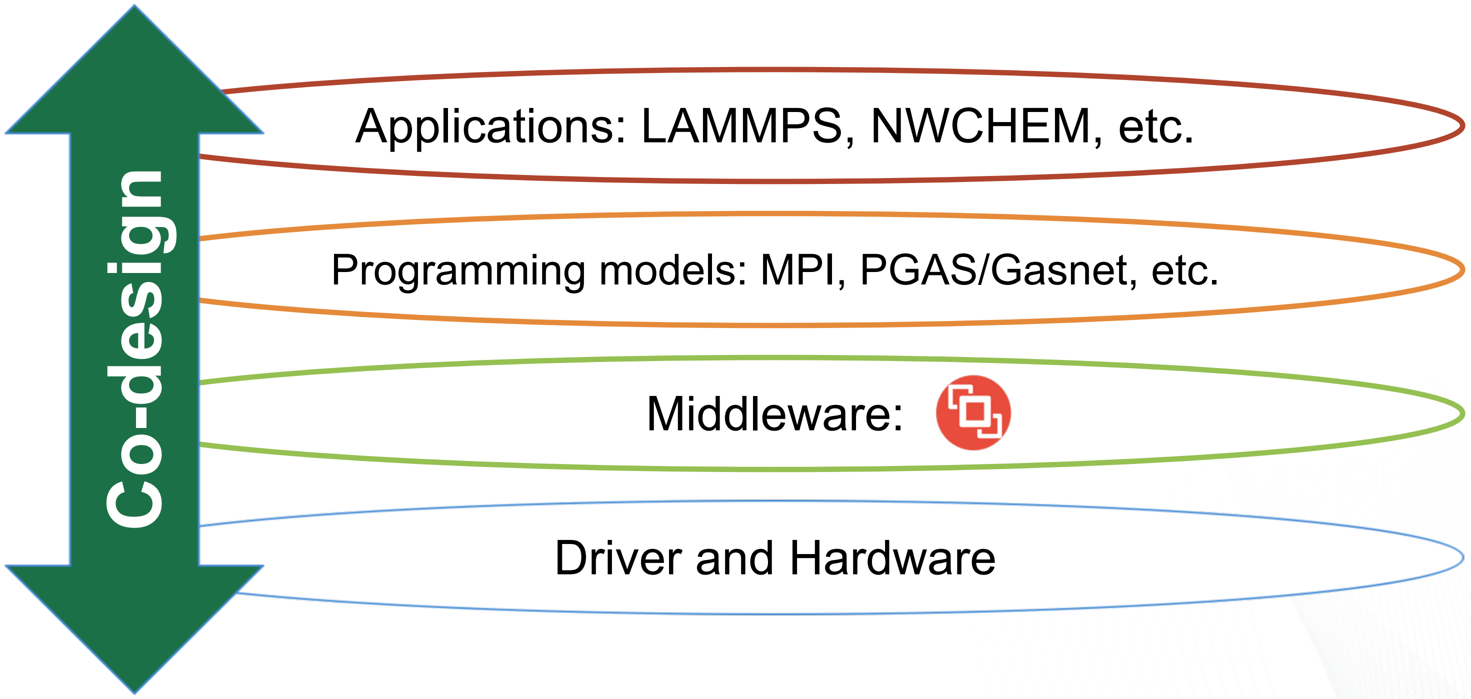
Decades of community and industry experience in development of HPC software

What we are doing differently…

- UCX consolidates multiple industry and academic efforts
  - Mellanox MXM, IBM PAMI, ORNL/UTK/UH UCCS, etc.
- Supported and maintained by industry
  - IBM, Mellanox, NVIDIA, Pathscale

What we are doing differently...

- Co-design effort between national laboratories, academia, and industry


Co-design
Applications: LAMMPS, NWCHEM, etc.
Programming models: MPI, PGAS/Gasnet, etc.
Middleware:
Driver and Hardware

What's new about UCX?

- A collection (framework) of APIs, which enables construction of customized protocols
- Enables choosing between a low-level and high-level API, thus allows easy integration with a wide range of applications and middleware
- Customizes network stack to hardware by selecting the protocols and transports based on the capabilities and performance characteristics
- Thread is a first class citizen, not an after thought
- Mutli-rail cross transport transport (devices) capabilities
- Accelerators are represented as a transport, driven by a generic "glue" layer, which will work with all communication networks

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

Architecture

UCX Framework

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.

UC-T for Transport

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

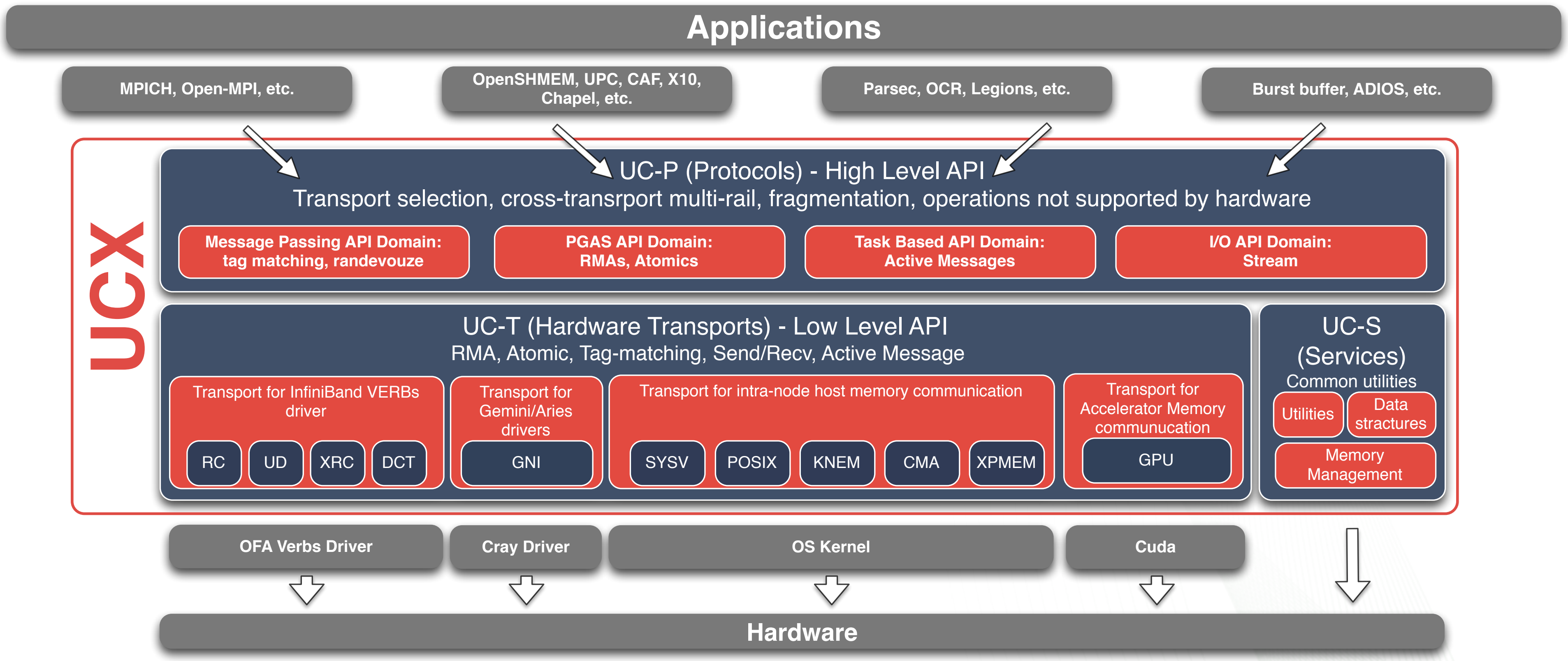
Functionality:
Setup and instantiation of communication operations.

UC-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.

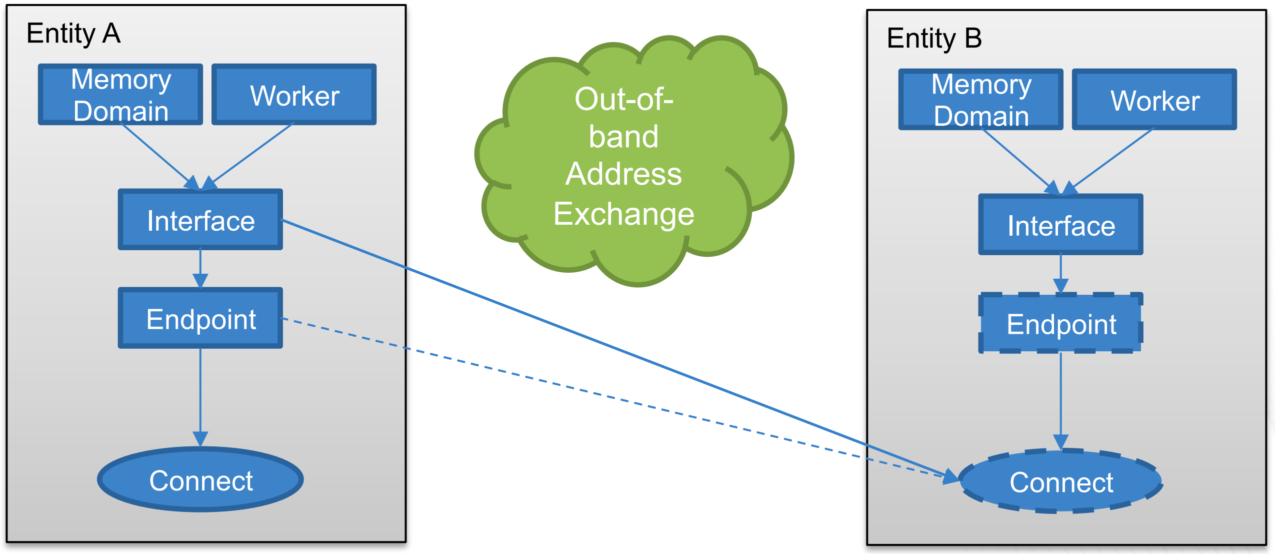
A High-level Overview



Applications
MPICH, Open-MPI, etc.
OpenSHMEM, UPC, CAF, X10, Chapel, etc.
Parsec, OCR, Legions, etc.
Burst buffer, ADIOS, etc.
UCX
UC-P (Protocols) - High Level API
Transport selection, cross-transrport multi-rail, fragmentation, operations not supported by hardware
Message Passing API Domain: tag matching, randevouze
PGAS API Domain: RMAs, Atomics
Task Based API Domain: Active Messages
I/O API Domain: Stream
UC-T (Hardware Transports) - Low Level API
RMA, Atomic, Tag-matching, Send/Recv, Active Message
Transport for InfiniBand VERBs driver
RC
UD
XRC
DCT
Transport for Gemini/Aries drivers
GNI
Transport for intra-node host memory communication
SYSV
POSIX
KNEM
CMA
XPMEM
Transport for Accelerator Memory communication
GPU
UC-S (Services)
Common utilities
Utilities
Data structures
Memory Management
OFA Verbs Driver
Cray Driver
OS Kernel
Cuda
Hardware

UCT (Transport Layer) Objects

- uct_worker_h
  - A context for separate progress engine and communication resources. Can be either thread-dedicated or shared.
- 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/Address to a remote interface. Used to initiate communications



OAK RIDGE National Laboratory

UCT Primitives

Memory:

- Register memory within the domain
- Allocate memory
- Pack memory region handle to a remote-key-buffer
- Unpack a remote-key-buffer into a remote-key

Operations

- 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” – “buffered” copy and 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

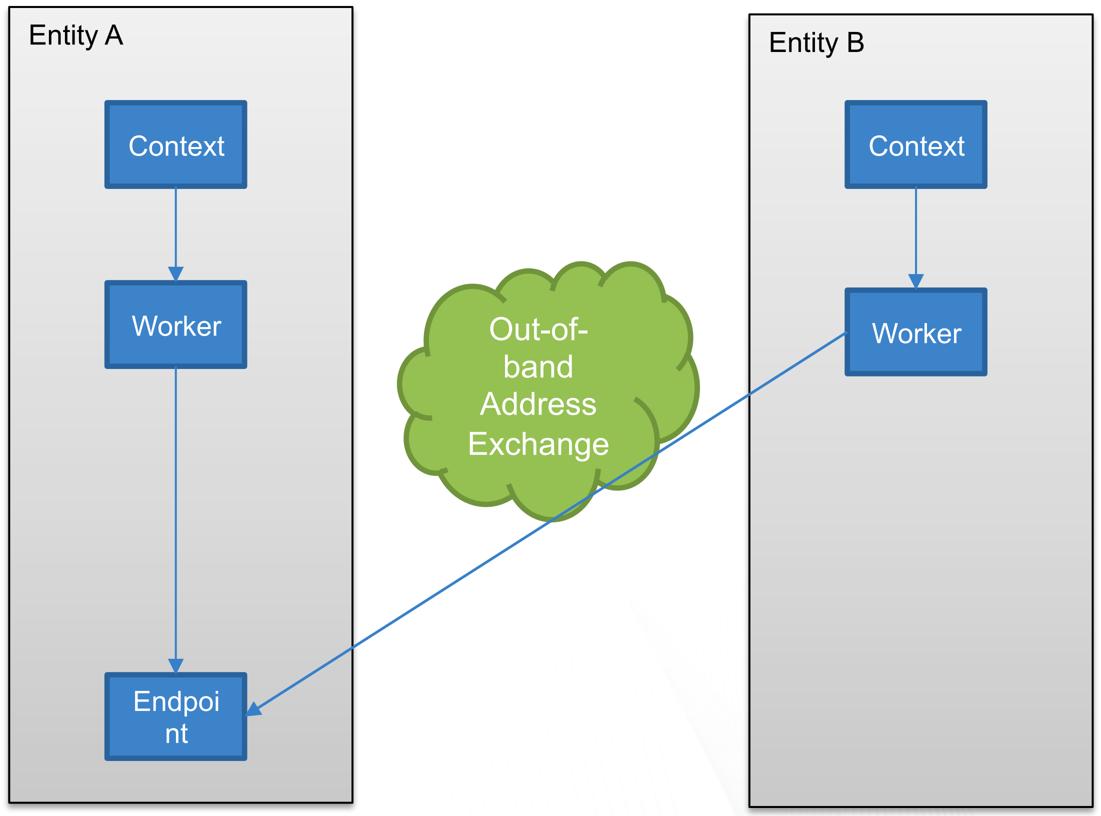
- All operations are non-blocking
- Return value indicates the status:
  - OK – operation is completed
  - INPROGRESS – operation has started, but not completed yet
  - NO_RESOURCE – cannot initiate the operation right now. The user might want to put this on a pending queue, or retry in a tight loop
  - 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

UCP (Protocol Layer)

- Mix-and-match transports, devices, and operations, for optimal performance, fragmentation, emulate unsupported operations, expose one-sided connection establishment
  - Based on UCT capabilities and performance estimations.
- Enforce ordering when required (e.g tag matching)
- Tag-matched send/receive:
  - Blocking / Non-blocking
  - Standard / Synchronous / Buffered
- Remote memory operations (Put, Get, AMO):
  - Blocking/ Non-blocking

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

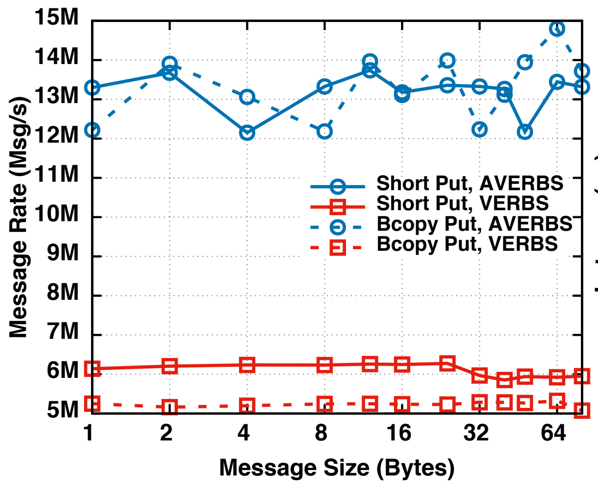


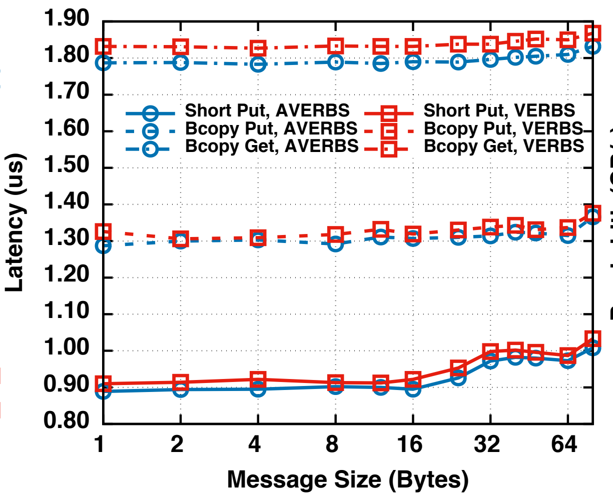
OAK RIDGE National Laboratory

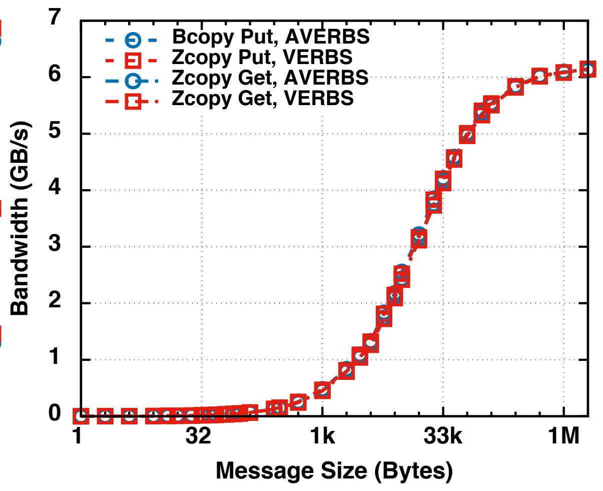
UCP Communications

- Tag-matched send/receive
  - Blocking / Non-blocking
  - Standard / Synchronous / Buffered
- Remote memory operations (Put, Get, AMO)
  - Blocking/ Non-blocking

Preliminary Evaluation (UCT)







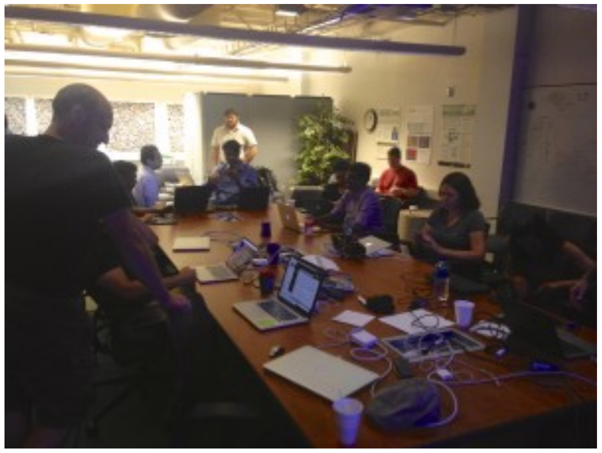
- Two HP ProLiant DL380p Gen8 servers
- Intel Xeon E5-2697 2.7GHz CPUs
- Mellanox SX6036 switch
- Single-port Mellanox Connect-IB FDR (10.10.5056)
- Mellanox OFED 2.4-1.0.4. (VERBS)
- Prototype implementation of Accelerated VERBS (AVERBS)

OAK RIDGE National Laboratory

Acknowledgments

Acknowledgments

- Argonne National Laboratory
- Lawrence Livermore National Laboratory
- Los Alamos National Laboratory
- Sandia National Laboratory
- University of Houston



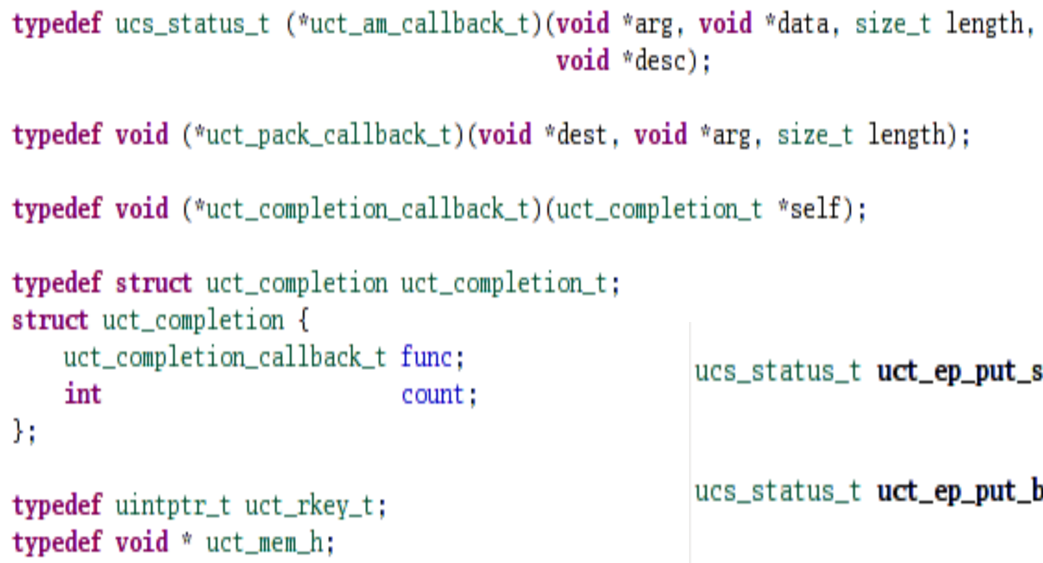
UCX

Unified Communication - X Framework

WEB: www.openucx.org
Contact: info@openucx.org
: https://github.com/orgs/openucx

Mailing List:
https://elist.ornl.gov/mailman/listinfo/ucx-group
ucx-group@elist.ornl.gov

UCT API



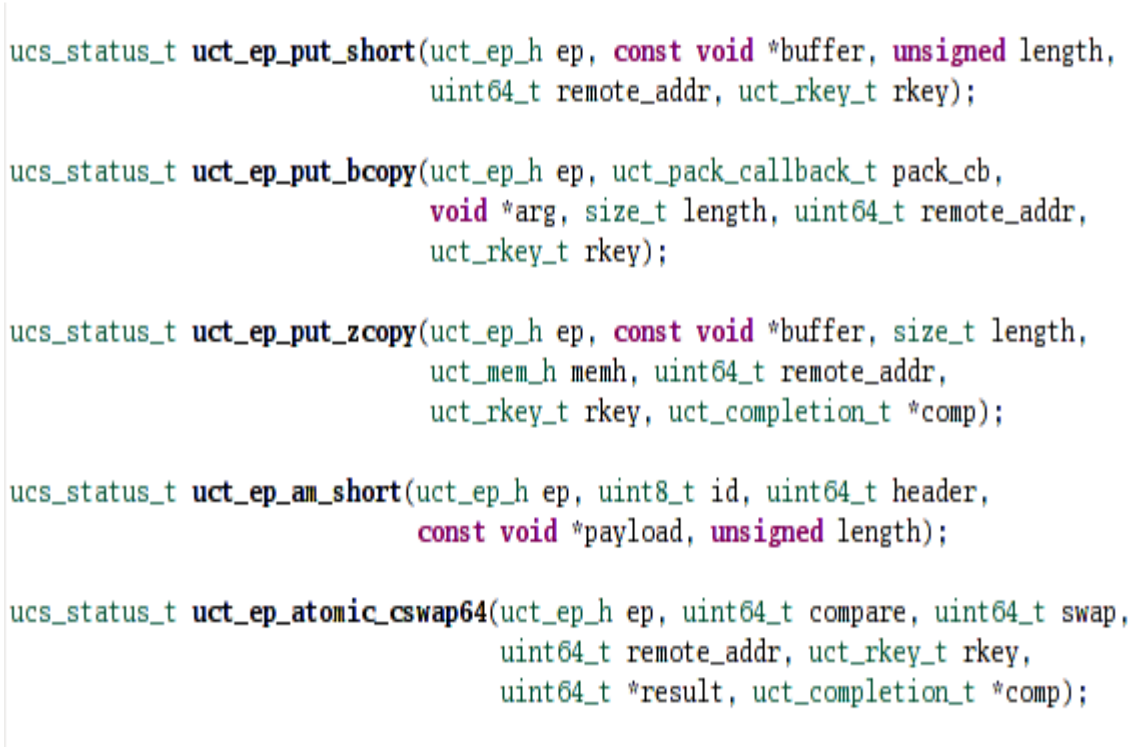
```c
typedef ucs_status_t (*uct_am_callback_t)(void *arg, void *data, size_t length,
                                          void *desc);

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);
```

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);
```