Tutorial:
Leveraging SmartNICs for HPC and Data Center Applications

Presenters: Richard Graham, Oscar Hernandez, Antonio J. Peña, Yong Qin, Jeffrey Young
Special Thanks To Helpers: Clay Hughes, Sergio Iserte, Muhammad Usman, Mariano Benito, Rich Vuduc
Many thanks to Aaron Jezghani and Will Powell for cluster support

Audience Survey

- Please follow presenter instructions to take a short survey on SmartNIC usage!

Tutorial Agenda

Agenda (Timing)

Time (EDT)	Topic	Presenter
8:30 - 8:40	Introduction and Attendee Survey	Jeff
8:40 - 9:10	SmartNIC introduction and overview	Rich G / Jeff
9:10 - 9:20	Programming approaches for HPC with SmartNICs	Jeff
9:20 - 9:35	SmartNIC Research – State of the Art	Oscar
9:35 - 10:00	DOCA MPI Implementations; Security Topics	Rich G / Yong
10:00 - 10:30	Break	
10:30 - 11:10	OpenMP offload to the SmartNIC; ODOS Demos	Antonio
11:10 - 11:55	Hands On - Application Experiences	All
11:55 - 12:00	Wrap-up	All

See the updated agenda at **https://github.com/gt-crnch-rg/smartnic-tutorial-sc24/**

Introduction

Tutorial Objectives

- Describe how SmartNICs can be used as HPC/AI accelerators
- Describe how users can benefit from acceleration engines in their current and future applications
- Describe how one can use one family of SmartNICs to accelerate applications and communication libraries
- Share demo experiences using SmartNICs with canned examples

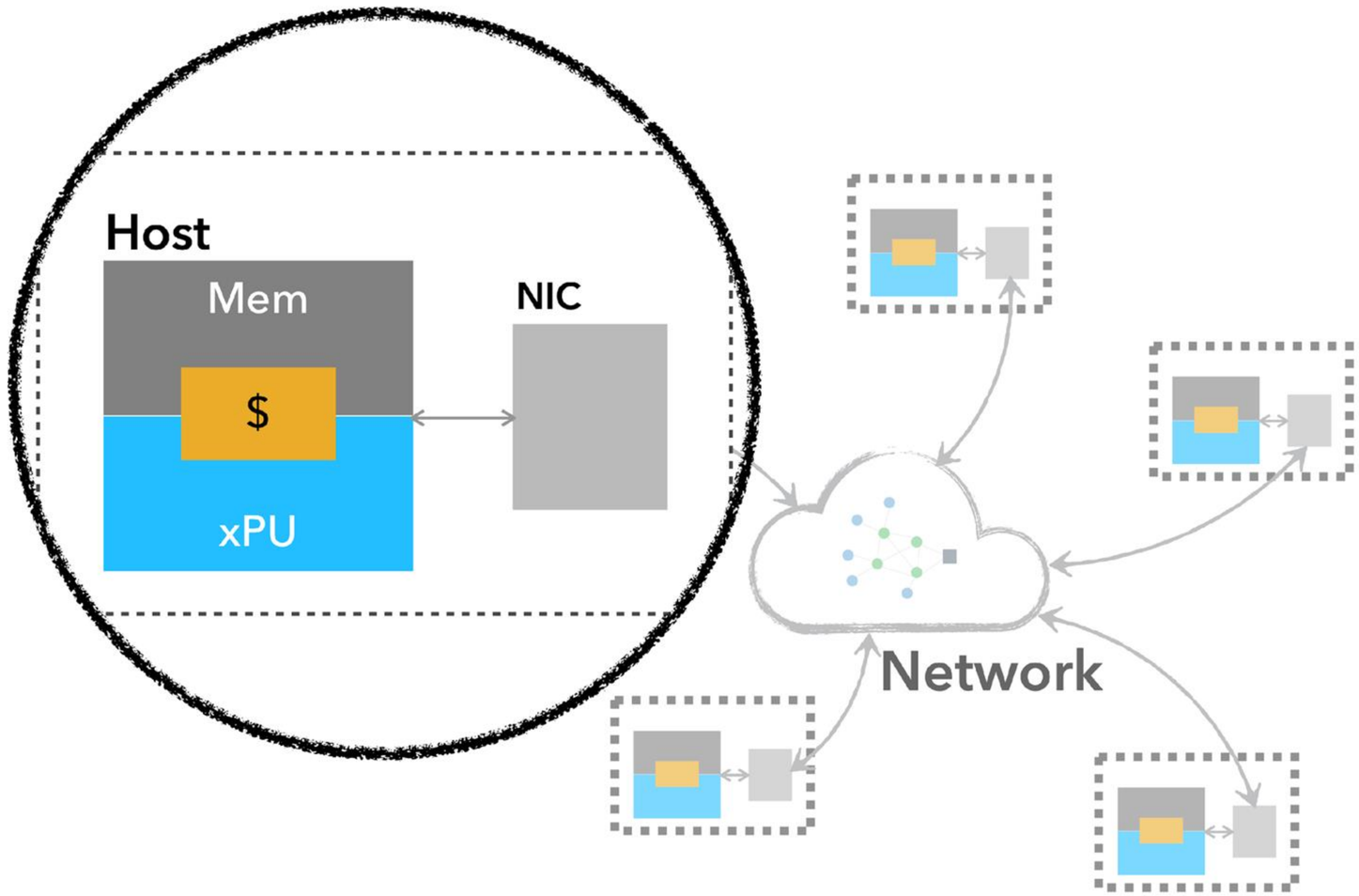
Anatomy of a supercomputer

The basic building block of a distributed-memory cluster or supercomputer is a node.

Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.



DPUs in modern clusters

The basic building block of a distributed-memory cluster or supercomputer is a node.

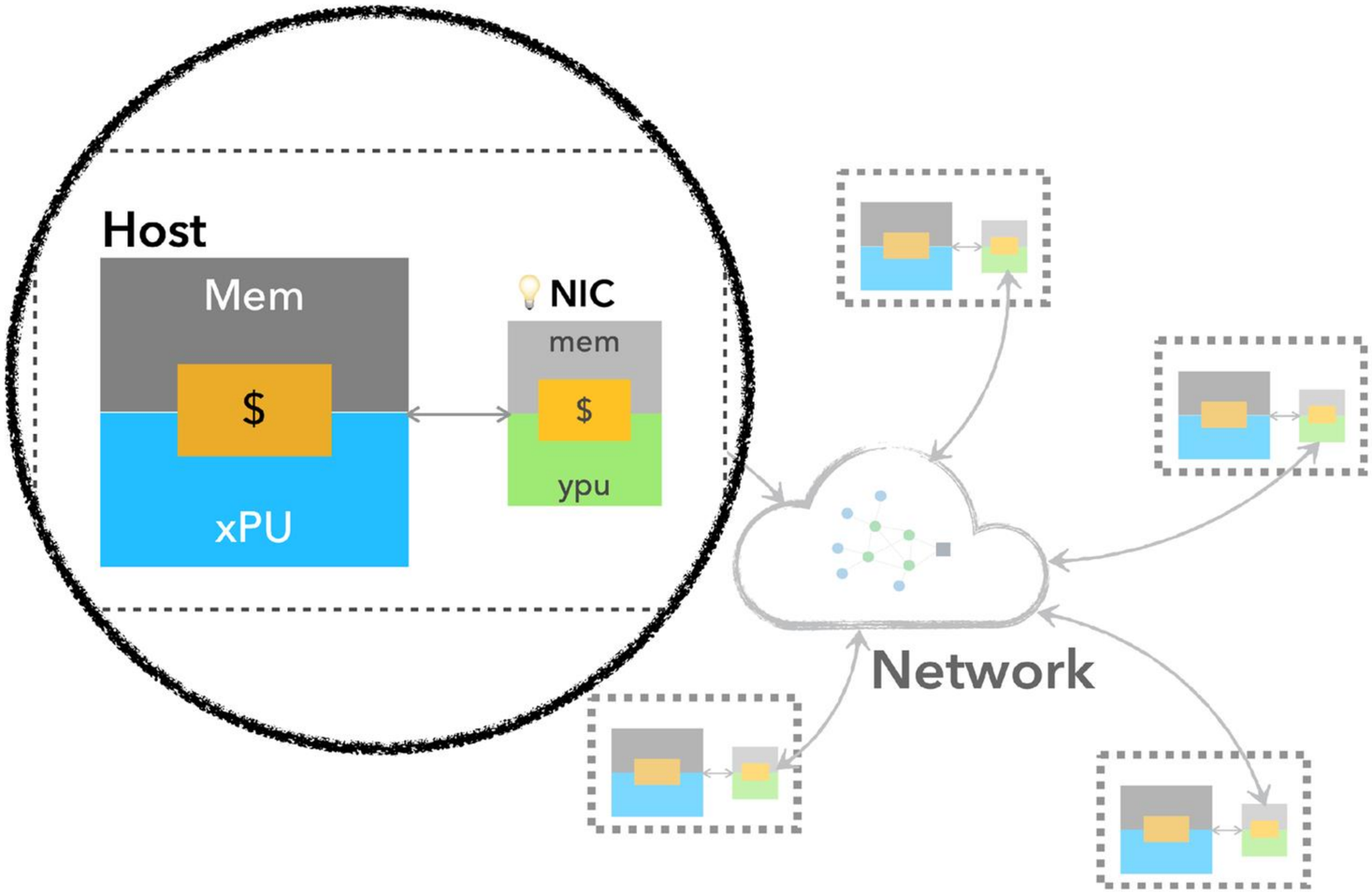
Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.

In a DPU, the NIC becomes "host-like" via the addition of processing (ypu), memory and other accelerations engines.

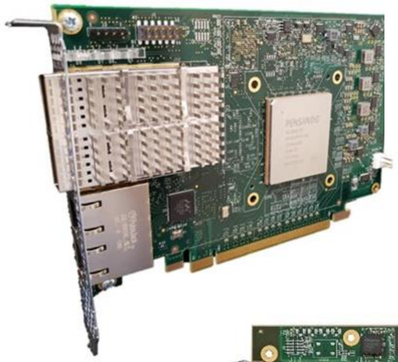
Node



SmartNICs and Data Processing Units (DPUs)

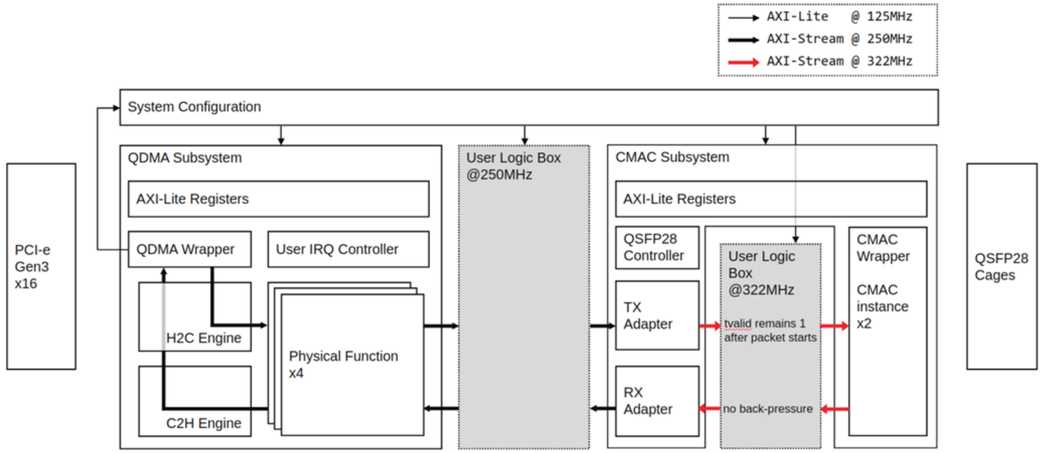
SmartNICs Available Today

- NVIDIA – BlueField family of Data Processing Units
- AMD – Pensando and Alveo FPGA
- Intel Infrastructure Processing Units (IPUs)
- Intel FPGA variants (Silicom)
- Marvell Octeon
- Research projects like sPIN



DPU Example - AMD SmartNICs

- Many variants of SmartNICs that largely fall into CPU-based and FPGA-based
- As an example, AMD has both Pensando Distributed Service Cards (DSC) and Alveo-based SmartNICs
  - Pensando DSC supports Programming Protocol-independent Packet Processors (P4) for flexible packet processing
  - Alveo FPGAs couple programmable logic with tightly integrated NICs
    - OpenNIC shell has led to support of P4 via projects by Esnet.



OpenNIC overview diagram from https://github.com/Xilinx/open-nic

Learn more about the variety of DPU products and projects at the 2023 International Workshop on Smart Networks, Data Processing and Infrastructure Units at https://dpu.ornl.gov

AMD presentation: https://dpu.ornl.gov/wp-content/uploads/2023/06/DPU23-AMD.pdf

DPU Example - Intel, Marvell

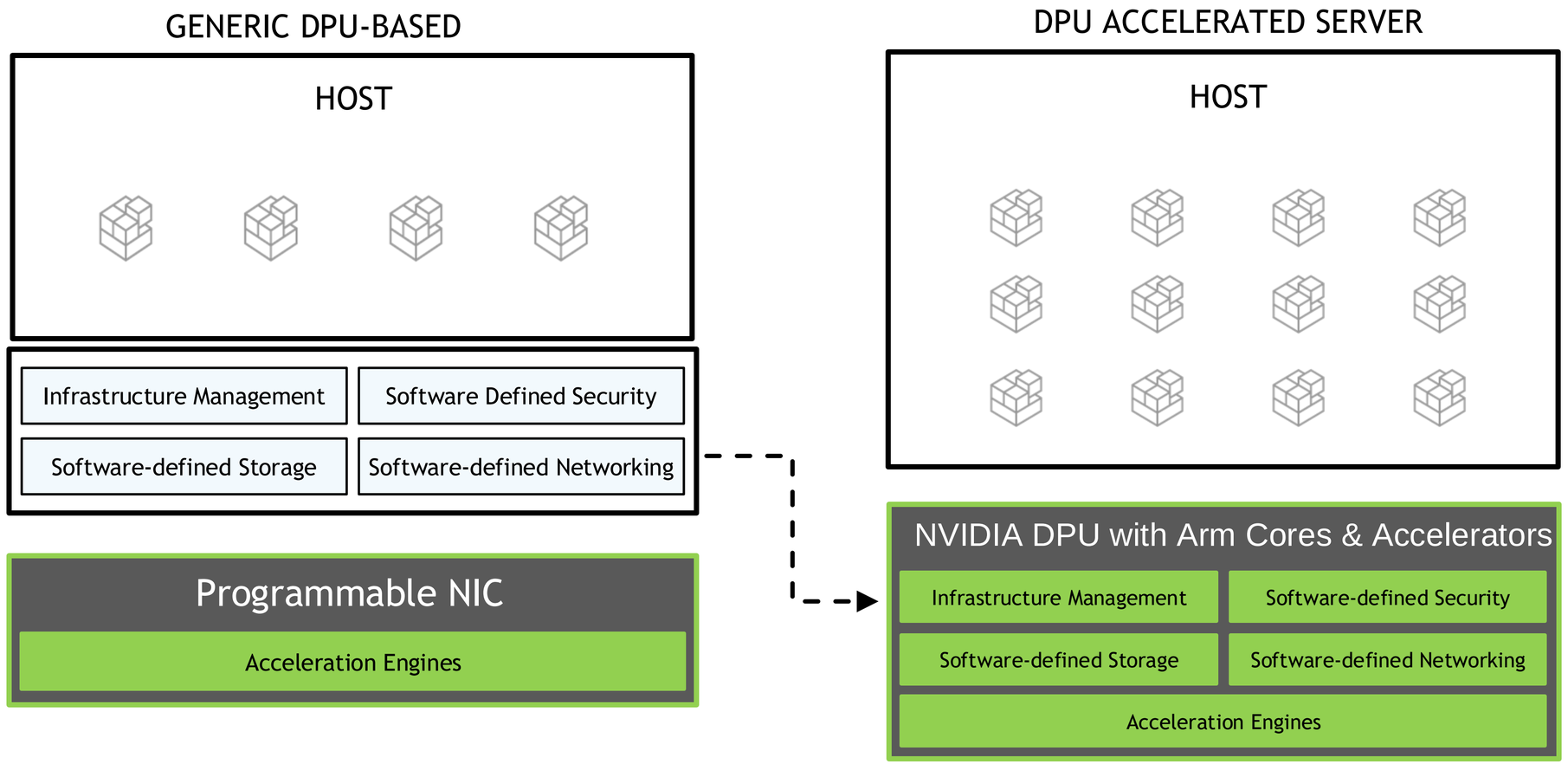
- Intel's Infrastructure Processing Units (IPU) focuses on accelerating common network services for virtualized environments
  - Offload networking stack, storage processing, security functions and even hypervisor capabilities
  - IPUs contain a Stratix/Agilex FPGA for line-speed packet processing and a CPU for serial computations
    - No OS support on the IPU!
- Marvel Octeon DPUs utilize Arm-based CPUs
  - Focus on supporting Data Plane Development Kit (DPDK) functions like virtualization, storage offload
- Both of these DPUs are focused on in-line and service-level processing, not HPC

https://www.intel.com/content/www/us/en/products/docs/programmable/ipu-based-cloud-infrastructure-white-paper.html

https://www.marvell.com/products/data-processing-units.html

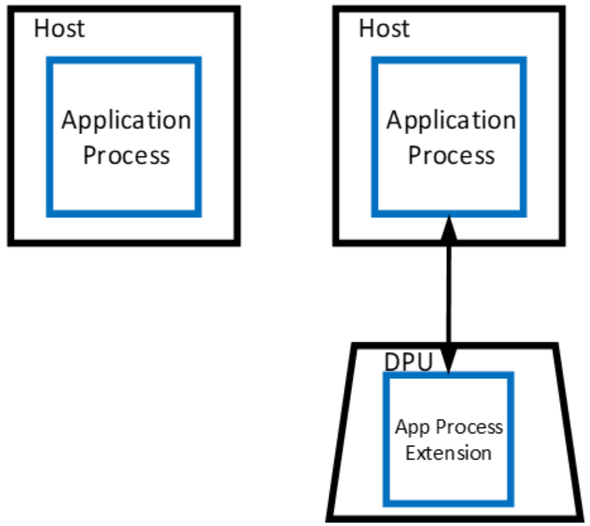
NVIDIA’s BlueField DPUs

NVIDIA’s BlueField Data Processing Unit



DPU offloading Models

Extension of application processing – algorithms split between host and DPU

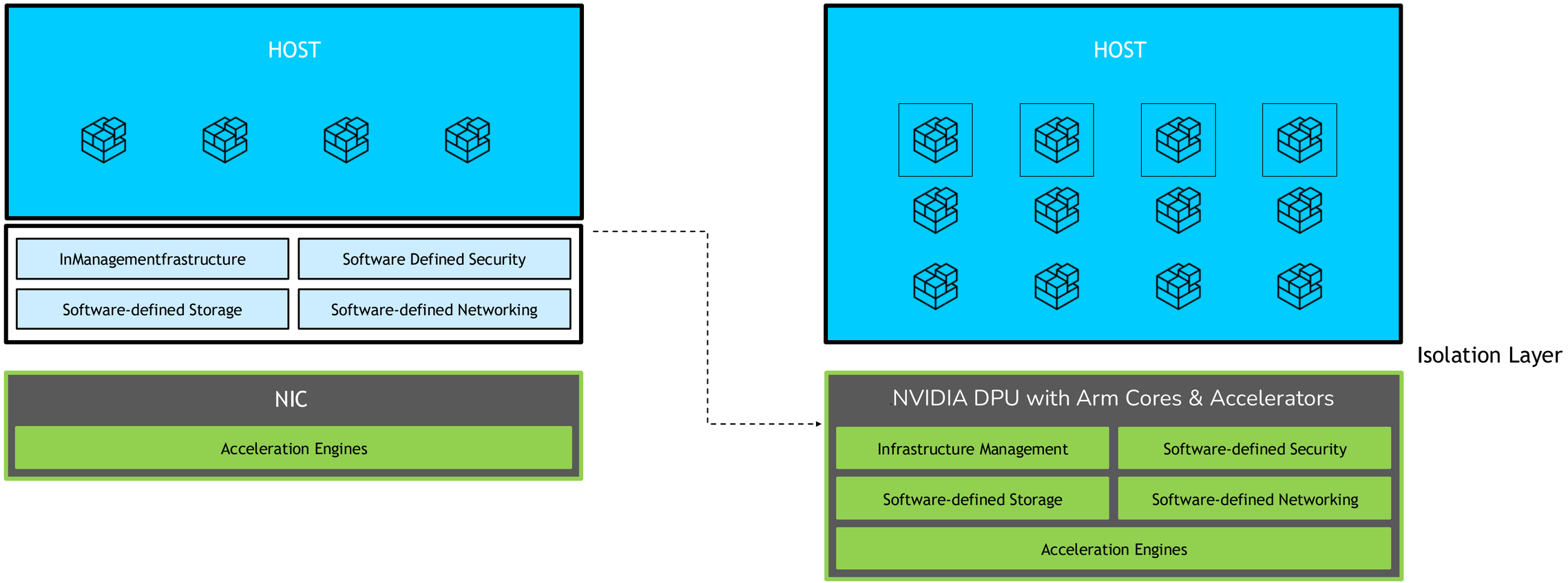


- **Offloaded system services**
  - Partial offload, such as storage – similar to application offload
  - Full offload, such as system security services
- Fully offloaded application service
  - Such as application telemetry collection

NVIDIA BLUEFIELD

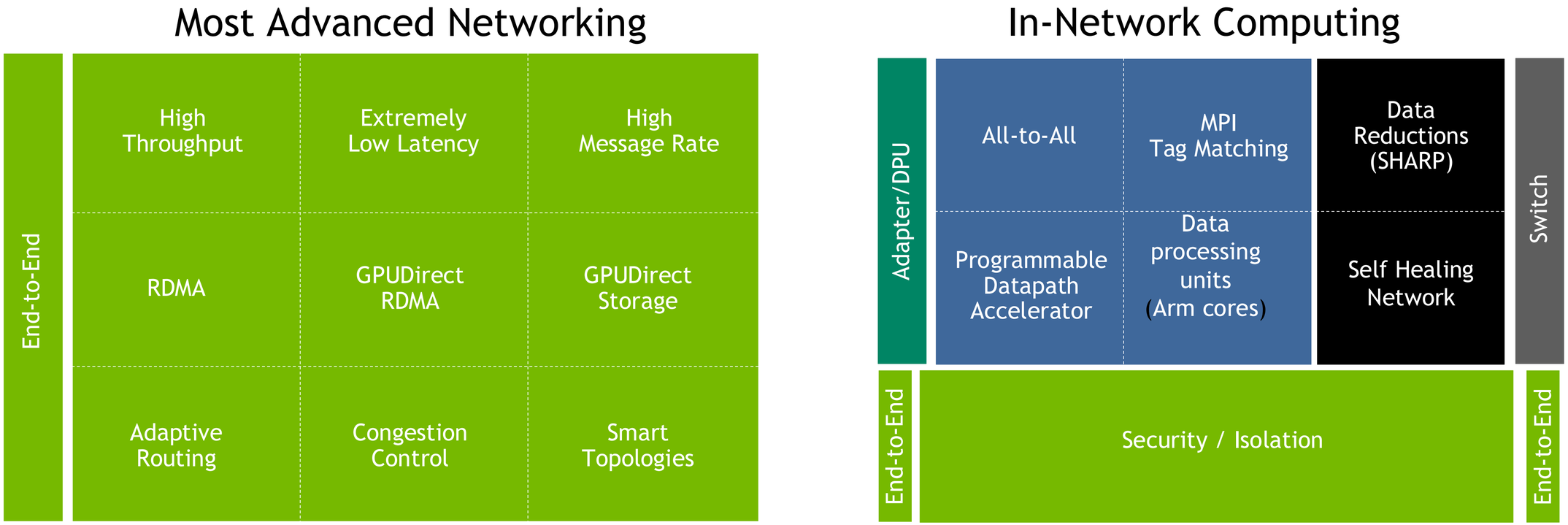
	BlueField-2	BlueField-3
Network Bandwidth	200Gb/s	400Gb/s
RDMA max msg rate	215Mpps	370Mpps
Compute Cores	8	16
Compute	SPECINT2K6: 70	SPECINT2K6: 350
Memory Bandwidth	17GB/s	80GB/s
NVMe-OF	10M IOPs @ 4KB	18M IOPs@ 4KB
NVMe SNAP	5.4M IOPs @ 4KB	10M IOPs @ 4KB

BlueField Data Processing Unit



- Software-Defined, Hardware-Accelerated Data Center Infrastructure-on-a-Chip

In-Network Accelerated Supercomputing



- Software-Defined, Hardware-Accelerated, InfiniBand Network

Design Considerations

- Asynchronous computation style
  - Host and DPU need to be “in sync”

- Compute limitations
  - DPU cores may be less powerful computationally with respect to the host compute engines
  - At least one order of magnitude less compute capabilities than the compute complex
    - Selective as to how much work to provide, so as not to become the bottleneck
    - Requires work sharing
- However, DPUs can have targeted acceleration engines for features like security

More Design Considerations for DPUs

- Network access
  - Source/destination of network traffic
  - Can post network requests on behalf of memory locations that are host-resident
  - Agnostic to the type of compute host

- BlueField enhancements
  - Work requests can be posted on behalf of memory that is host-resident – Cross-GVMI memory keys
  - Some optimized data paths between the host and the BlueField – GGA

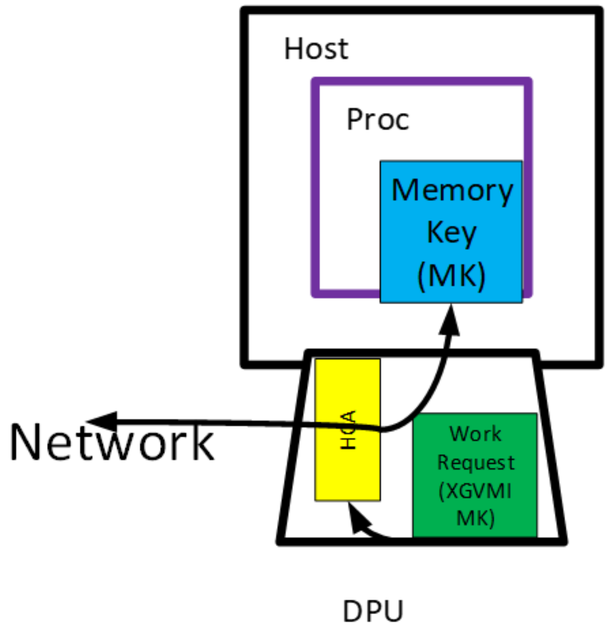
- Possess memory bandwidth independent of that of the host
  - Selectively use this memory resource to supplement what is available in the compute complex – not an all or none proposition

- Can’t do any better than saturate the network BW – need to do just enough to saturate the network

BlueField-based Acceleration Engines

Cross-GVMI Memory

- Memory keys that allow DPU based work-requests to reference host-side memory



- DPU can initiate data transfer on behalf of the host, with no host involvement
  - Do not need to move data to the DPU before the DPU can send it
  - Can post receive work requests on behalf of memory that is host resident

BlueField DPUs: Offload with DPDK and SPDK

- Bluefield-based HW accelerations available using open source DPDK and SPDK frameworks
  - DPDK - APIs for performing userland packet and buffer processing
  - SPDK - userland implementation of the NVMe protocol (including NVMe over fabrics)
- Provide high-level API access to accelerators DPDK mlx5 driver provides access to several accelerator driver classes:
  - class=crypto – memory region encryption/decryption
  - class=eth – ethernet polling offload
  - class=vdpa – performs packet checksum offloads (among many others)
- DPDK accelerations are currently only supported when BF is in Ethernet mode

	Bluefield-2 DPDK	Bluefield-2 SPDK	Bluefield-3 DPDK	Bluefield-3 SPDK
Crypto	Y		Y	
vDPA (no relation to BF3 DPA)	Y		Y	
NVMe Target Offload		Y		Y

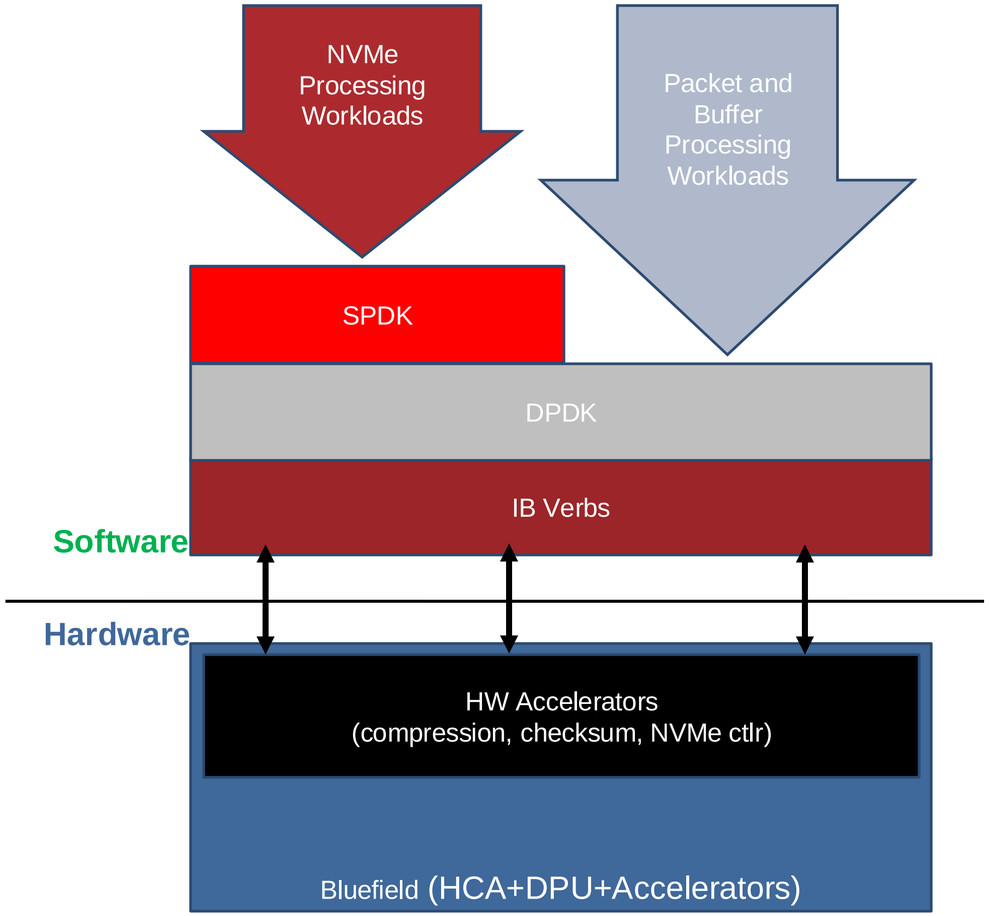
BlueField-2: Accessing Common Offloads

- DPDK Compression example:

  dpdk-test-compress-perf -l0-1 -n 1 -a <pcie address>,class=compress --
  --driver-name mlx5 --input-file <file.txt> --compress-level 1:1:9
  --num-iter 10 --extended-input-sz 1048576 --max-num-sgl-segs 16
  --huffman-enc fixed

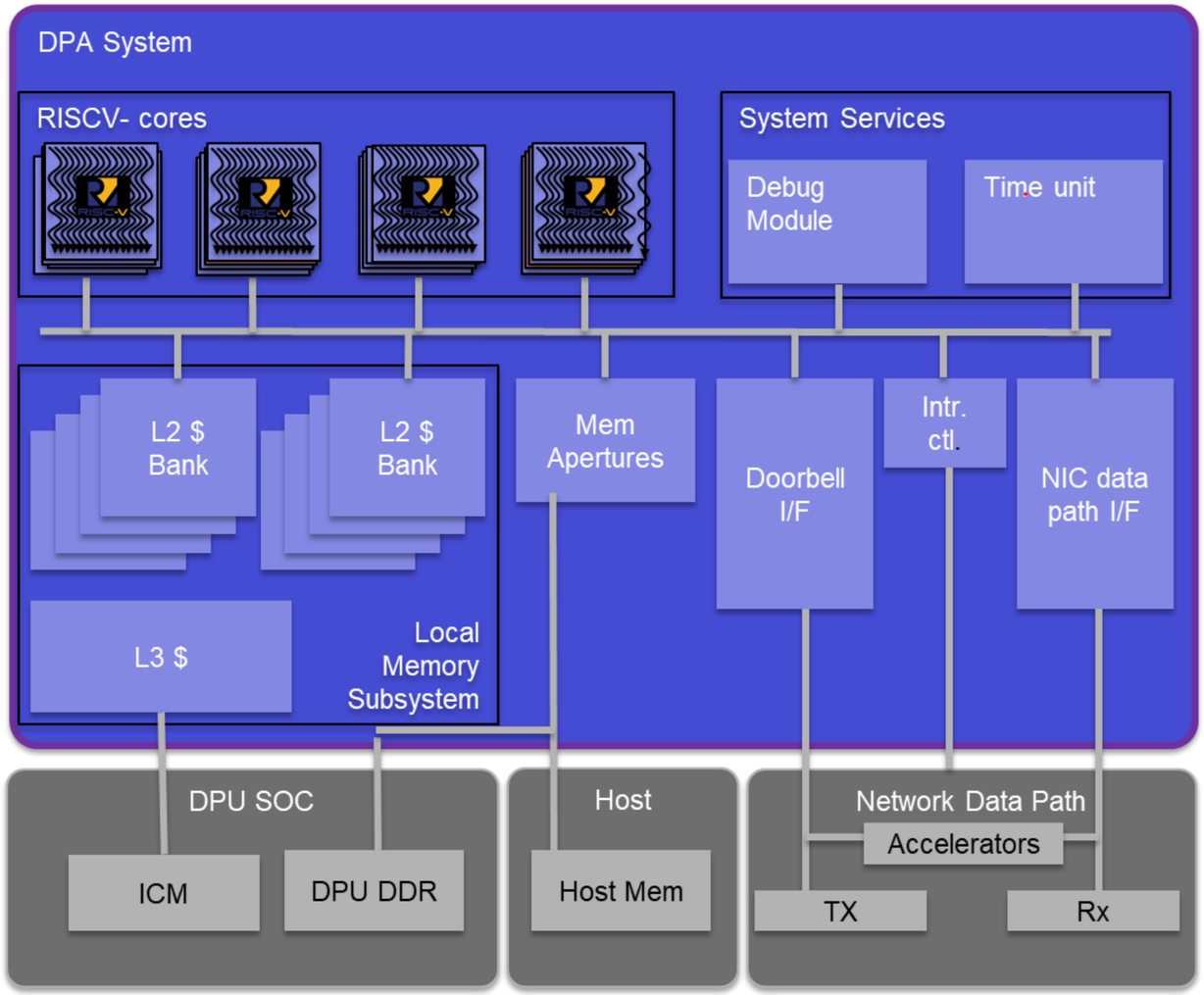
- SPDK offload example:

  Simple NVMe-oF Target Offload Benchmark
  https://enterprise-support.nvidia.com/s/article/simple-nvme-of-target-offload-benchmark



Data path accelerator (DPA)

- RV64IMAC(B) + NVIDIA extensions (e.g. arithmetic ops)
- Part of the Connect-X network core
- High BW Dedicated local cluster cache
- Access to Host system memory
- NIC direct access to DPA caches
- NIC interrupts and doorbells
- Full access to accelerators
- Standard Debug Module and timer unit
- DPA-RTOS
  - Low memory footprint, highly threaded
  - Low latency scalable HW based scheduling
  - Process isolation enforced



DPA highlights – Bluefield-3 (ConnectX-7 core)

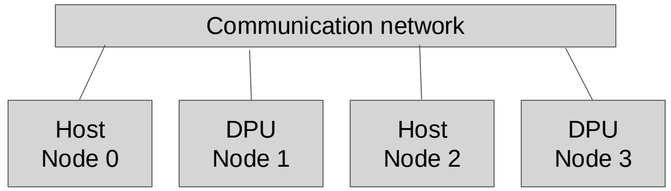
DPA			
Architecture		Standard RISC-V RV64IMAC(B)-USM	
Pipeline Width		1 Instruction / Clock	
Threads per Core		16	
Number of Cores		16	
Frequency Target		Core 1.8 GHz / System 505 MHz	
Caches		Capacity	Access / Clock
	L1 Code / Core	8 KB (2-4K inst.)	1 (per Core)
	L1 Data / Thread	1 KB	1 (per Core)
	L2 / Cluster	1.5 MB	8
	L3 / Cluster	3 MB	2
	ICMC / NIC	4 MB	2

Programming Models and Frameworks

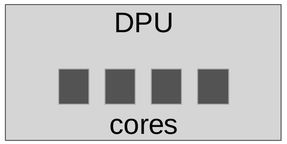
High-level HPC Programming Models

Four programming models are available to program DPUs:

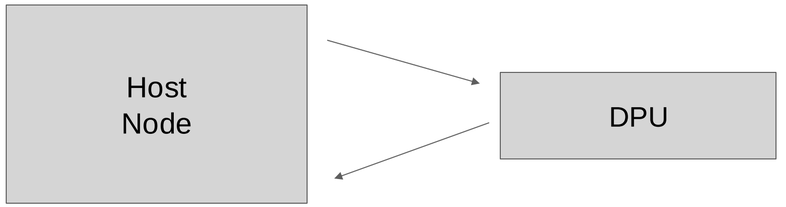
Distributed and distributed-shared memory
- MPI – send/receive based
- OpenSHMEM – put/get based



Shared memory
- OpenMP (shared memory)

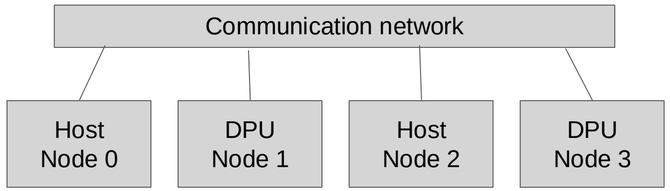


Accelerator offloading
- MPI, OpenSHMEM
- OpenMP offload



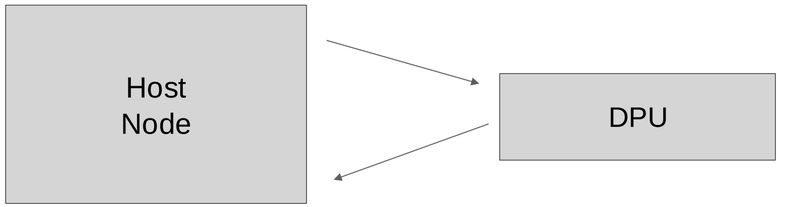
"Offloading network operations"

Distributed Model



- Ranks are mapped to compute nodes and DPUs.
- A DPU is treated as another node in the system and its rank can be part of a communicator or sub communicator.
- DPU ranks will have different computing capabilities
- Potential use-cases:
  - Application is decomposed to use DPUs for computation or I/O
    - Examples:
      - DPUs cores are used for computation
      - DPUs handle the I/O ranks of an application

Accelerator Offload



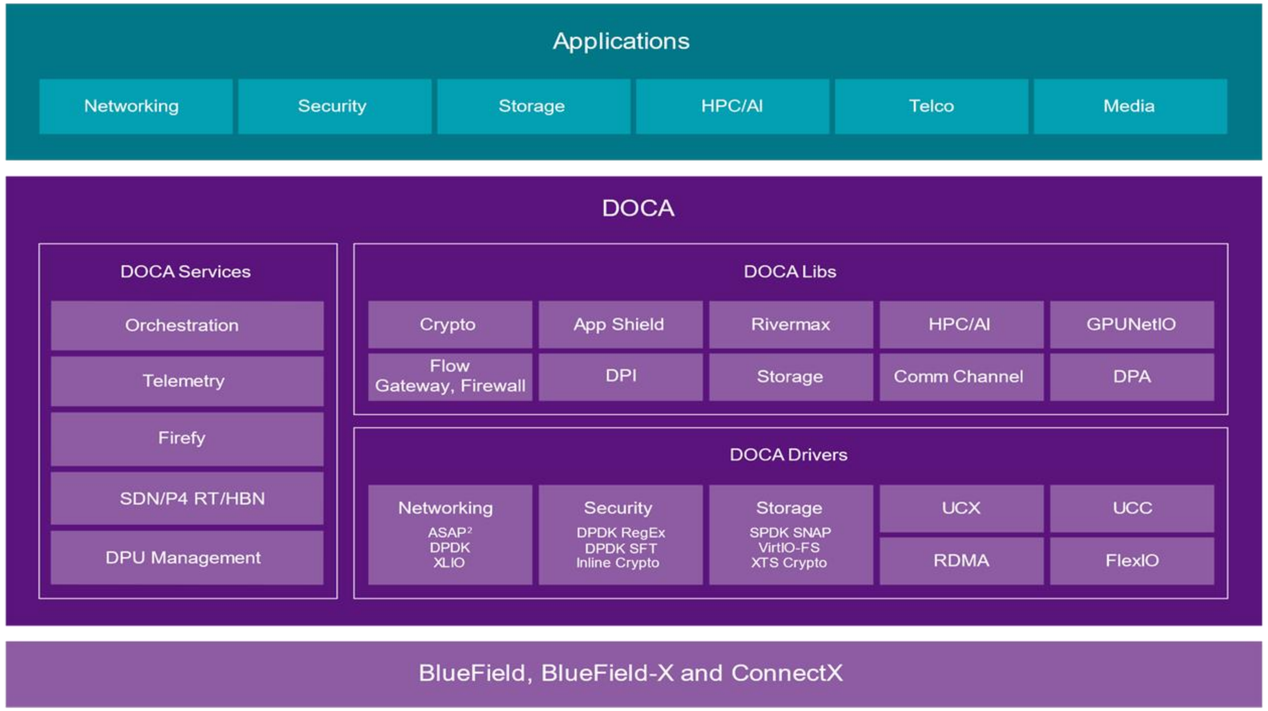
"Offloading network operations"

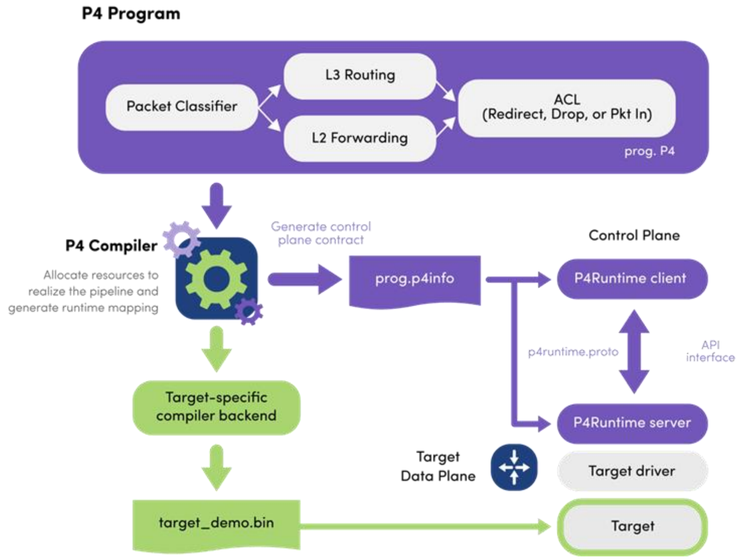
Two model views:

- Programming model exposes the accelerator
  - When accelerator is exposed
  - Program explicitly interacts with the accelerator, such as with an MPI library implementation

- Programming model does not expose the accelerator
  - Accelerator is used by services the application uses such as MPI, OpenSHMEM and storage
  - Services explicitly interact with the accelerator

Other Frameworks for DPUs





- DOCA
  - NVIDIA-specific SDK for application developers
  - Supports Data Plane Development Kit (DPDK), RDMA, compression, etc.
- P4
  - Provides a vendor-neutral specification for packet processing and manipulation

Learn more about P4 and DOCA at:

P4 - https://p4.org

DOCA - https://developer.nvidia.com/networking/doca/getting-started

DPU Research – State of the Art

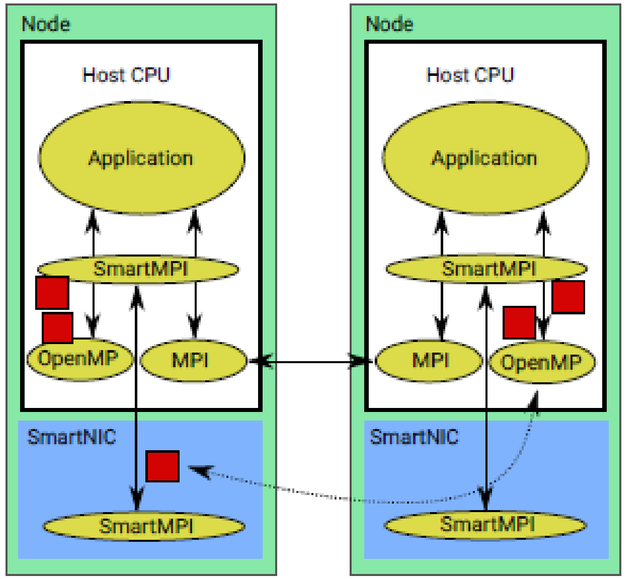
Future Directions for SmartNICs Research

Several areas (not limited to):

- Task-based runtimes
- AI and Machine learning acceleration
- Monitoring and performance tools
- Data Compression
- Acceleration storage

Additionally, there are several opportunities to extend programming models for SmartNICs

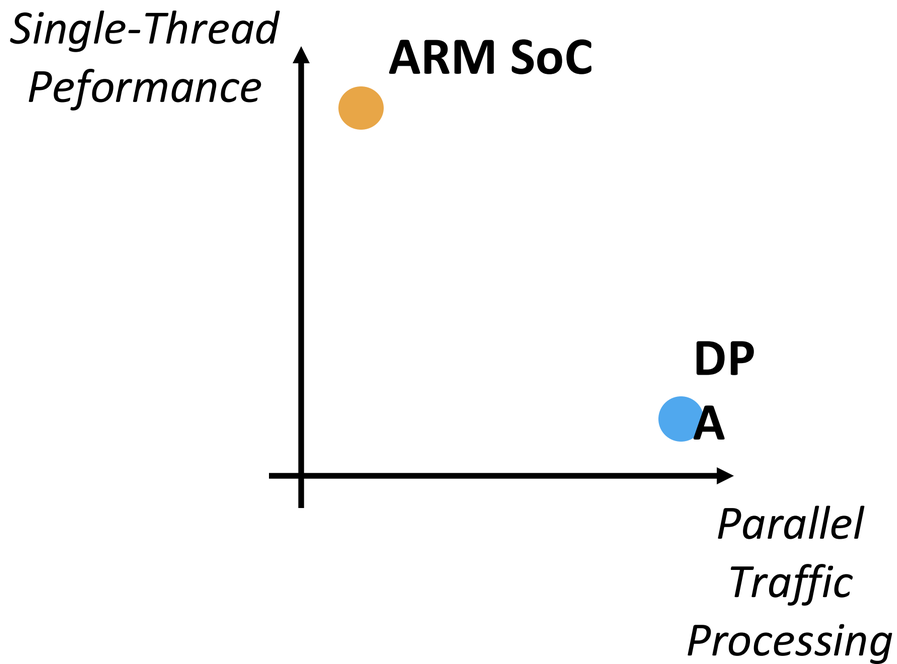
- Kokkos
- PGAS
- Task-based runtimes

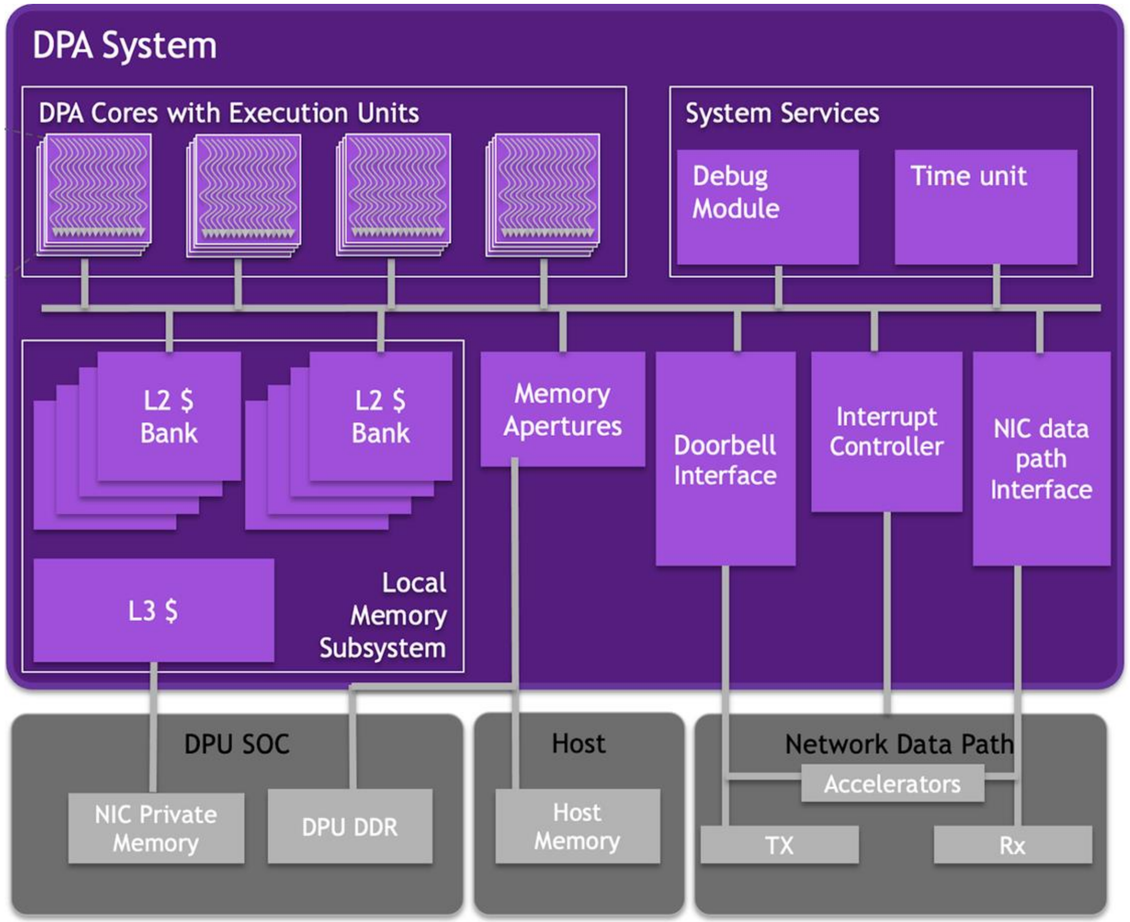


Example of task-based runtime work by *Weinzierl, et al. "Intelligent algorithms on intelligent networks—experiences and challenges using NVIDIA's BlueField technology". Slides at https://dpu.ornl.gov/*

Datapath Accelerator (DPA)

- Introduced in BlueField-3/ConnectX-7/SuperNIC products
- 16 energy-efficient cores, 16 hardware threads each
- Direct access to NIC RDMA engine and accelerators
- Low-footprint C API
- Designed for highly-parallel packet processing:
  - RPCs, Storage, Collectives offloading





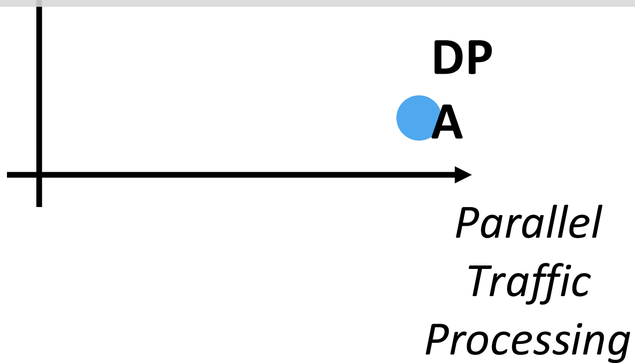
Datapath Accelerator (DPA)

- Introduced in BlueField-3/ConnectX-7/SuperNIC products
- 16 energy-efficient cores, 16 hardware threads each
- Direct access to NIC RDMA engine and accelerators

Bandwidth-Optimal, Fully-Offloaded Collectives

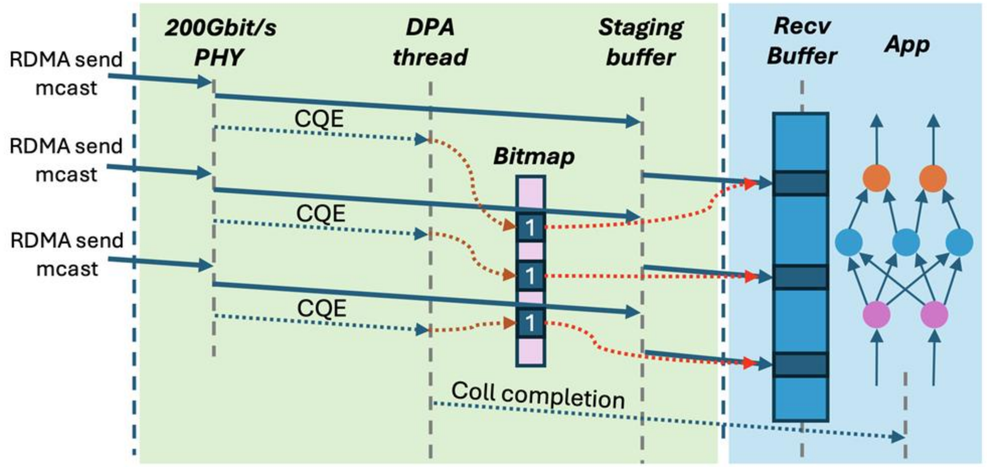
Mikhail Khalilov[1], Salvatore Di Girolamo[2],
Marcin Chrapek[1], Rami Nudelman[2],
Gil Bloch[2], and Torsten Hoefler[1]

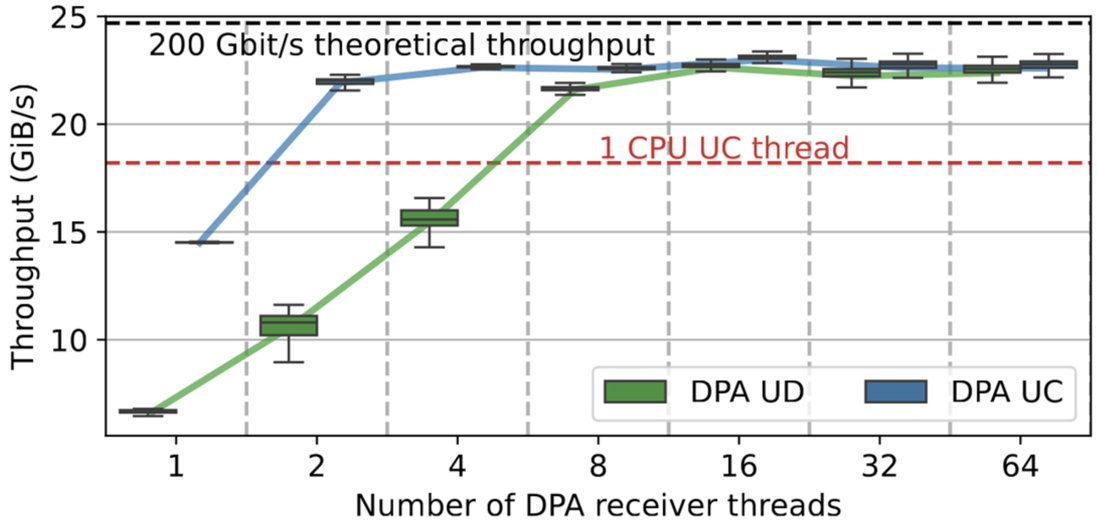
[1]ETH Zurich, 8092 Zurich, Switzerland
[2]NVIDIA Corporation



DPA use case: AI collectives acceleration

- Fully Sharded Data Parallel (FSDP) training used for LLAMA 3 training
- FSDP relies on Allgather that can be accelerated with hardware RDMA multicast
- Multicast 4KB datagram processing introduces high CPU overheads at >= 200 Gbit/s
- DPA is an ideal offloading Multicast Send/Receive





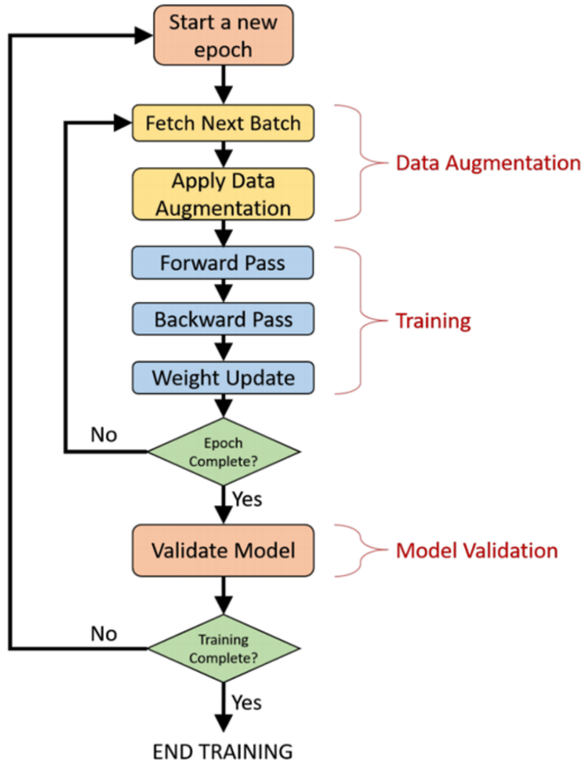
From ***"Bandwidth-optimal, Fully-Offloaded Collectives"***
Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler
Paper Presentation at SC '24 on Thursday, 21st in B312-B313A at 3:30 ET

Accelerating MPI and Deep Learning Applications with the DPU Technology

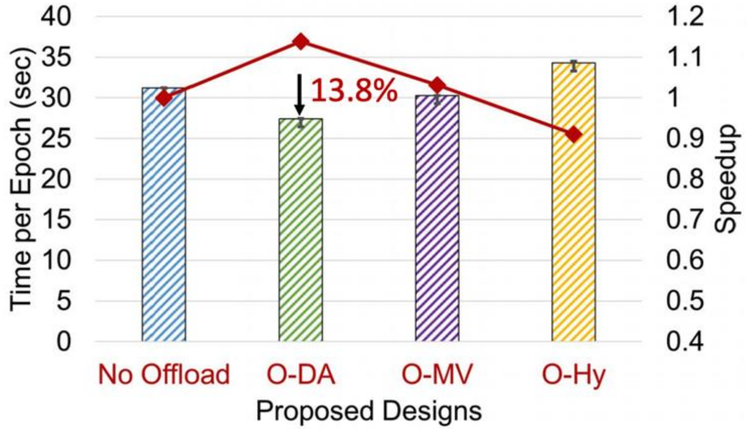
Nick Sarkauskas, Arpan Jain, Nawras Alnaasan, Tu Tran, Bharat Ramesh, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. (DK) Panda

EXPLOITING DPUS FOR DEEP NEURAL NETWORK TRAINING

- **There are several phases in Deep Neural Network Training**
  - Fetching Training Data
  - Data Augmentation
  - Forward Pass
  - Backward Pass
  - Weight Update
  - Model Validation
- **Different phases can be offloaded to DPUs to accelerate the training.**



- **Speedup**
  - Single node: O-DA (13.8%) and O-MV (3.1%)
  - Multi-node: Achieves average 13.9% speedup on 1-16 nodes



Single Node Experiments

Source: https://www.openfabrics.org/wp-content/uploads/2022-workshop/2022-workshop-presentations/101_DKPanda.pdf

Getting data where it's needed

potential for CPU, GPU, accelerators, and smart networks

Sam Antao, Nick Malaya, Matthew McIntire, Chuck Gilbert

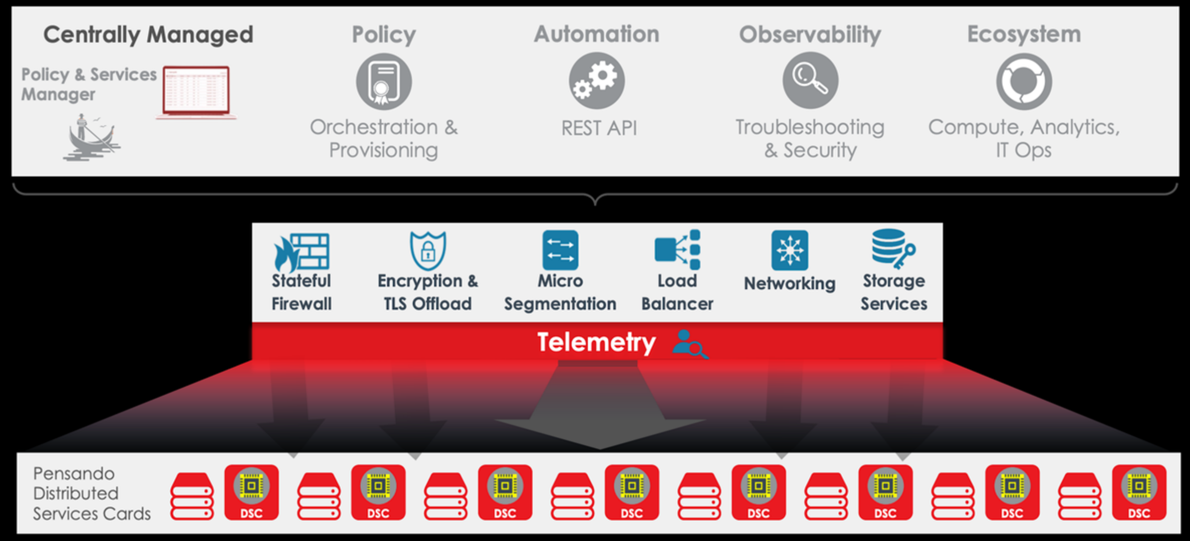
AMD SmartNIC Accelerator

Complete function set tightly coupled with programmable logic (up to 400G)

AMD together we advance_

[Public]

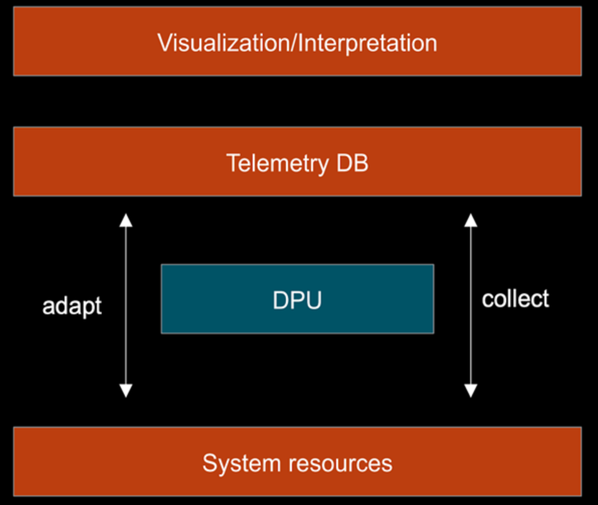
Pensando DSC



AMD together we advance_

Tooling and telemetry

- Performance is hard to assess
  - Collect data seamlessly
  - No workload disruption
  - Minimum user intervention
  - Low overhead performance sampling
- Integrate with popular profiling tools
- Visualization of traces
  - Realtime view
- Find what is actionable
  - How far off from peak empirical performance
  - Self calibration
- Translate findings into the network configuration
- Telemetry for different workloads
  - Learn what is latency bound
  - Learn what is bandwidth bound
  - Learn when to save power
    - Link cooling down



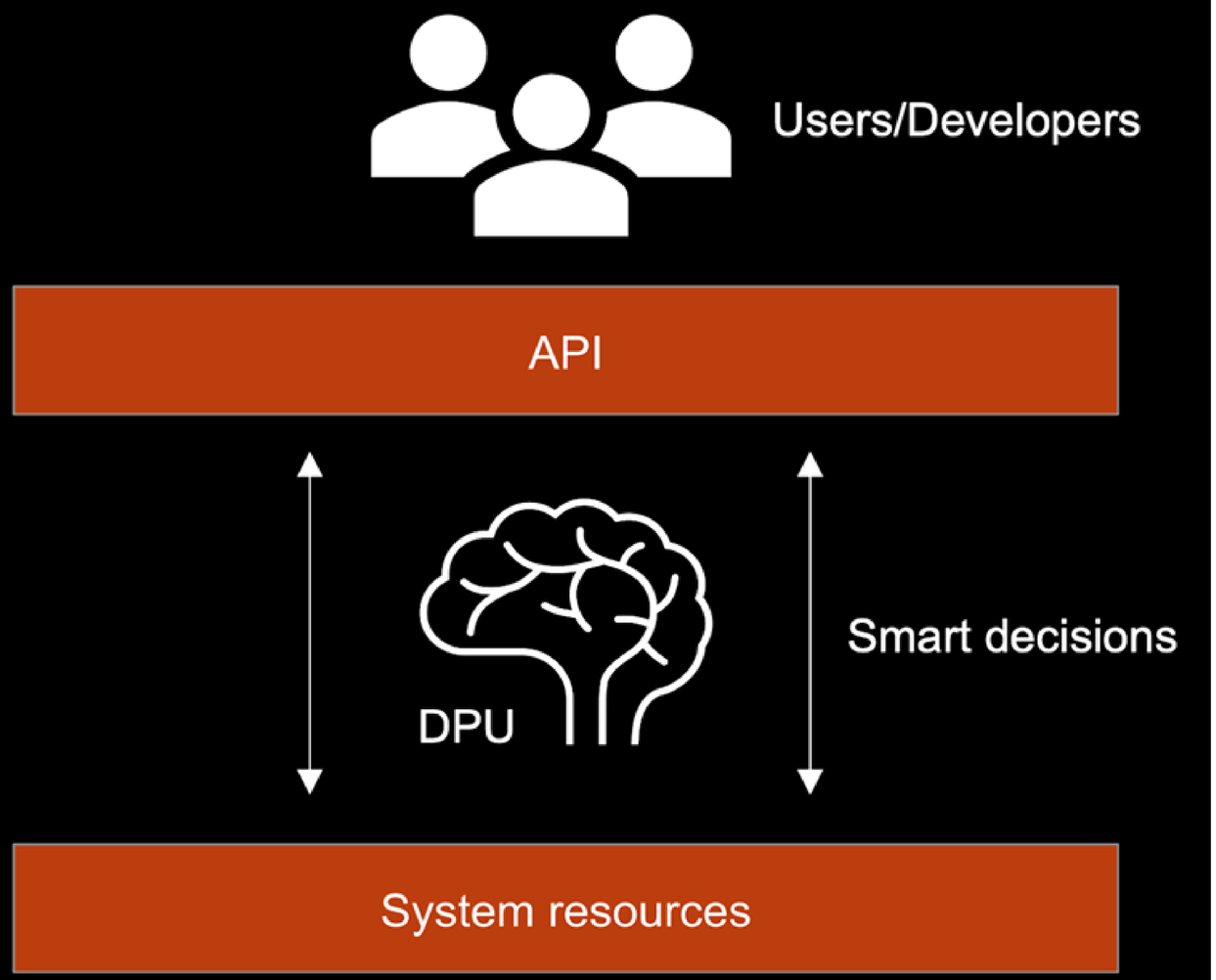
25 May 2023

AMD together we advance_

Source: https://dpu2023.ornl.gov/wp-content/uploads/2023/06/DPU23-AMD.pdf

Making smarter APIs

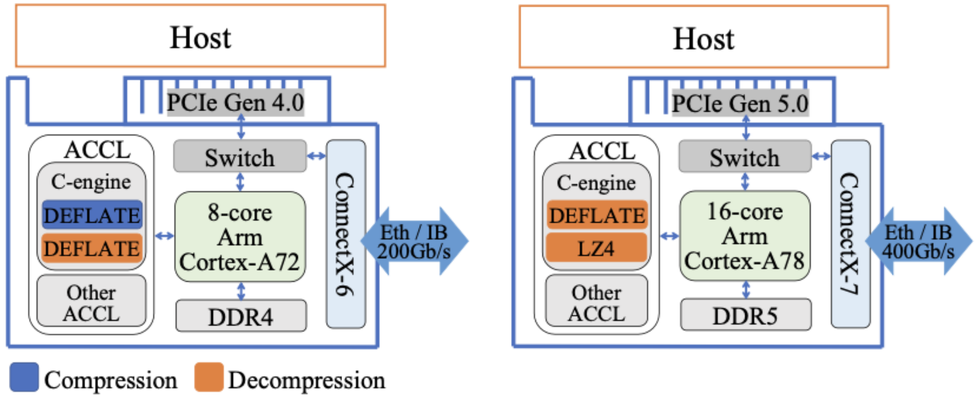
- Users (would) love
  - Stick with legacy APIs
  - Have simple and high-level application-level API
  - Ignore hardware intricacies
  - Simple abstractions
  - Focus on their science
- API developers (would) love
  - Digest all possible application-level information
  - Offload parameterization to the user
  - Have the user pick sensible defaults
- Can we meet in between?
- DPUs to make API implementation smarter
  - Collect telemetry
  - Realize sub-optimal decisions
  - Tune on-the-fly
  - Maintain data-base of good practices
  - Identify application signatures and map it to better defaults



Source: https://dpu2023.ornl.gov/wp-content/uploads/2023/06/DPU23-AMD.pdf

Accelerating Compression on BlueField DPU Architectures

- Goal
  - Optimize lossy and lossless data compression using NVIDIA BlueField DPUs.
  - Address performance bottlenecks in HPC, Big Data, and Deep Learning applications.
- PEDAL Library:
  - Compression:
    - Unified library leveraging BlueField's SoC and C-Engine for enhanced compression.
    - 101x compression speedup.
  - Communication:
    - BlueField-2/3 DPUs optimized for communication-oriented workloads.
    - Reduced communication latency by up to 88x.
- Highlights:
  - Significant acceleration in compression time.
  - Notable improvements in communication efficiency for HPC applications.
  - Paper:



(a) BlueField-2 DPU Arch. (b) BlueField-3 DPU Arch.

Li, Y., Kashyap, A., Chen, W., Guo, Y., & Lu, X. (2024). Accelerating Lossy and Lossless Compression on Emerging BlueField DPU Architectures. 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)

University of California Merced (UC-Merced)

More information about SmartNIC Research Directions

- ***SIAM PP 2024 minisymposium at https://crnch.gatech.edu/siam-pp-2024/***
- ***ISC'2023 DPU Workshop https://dpu.ornl.gov***

Transforming applications with MPI

Leveraging SmartNICs

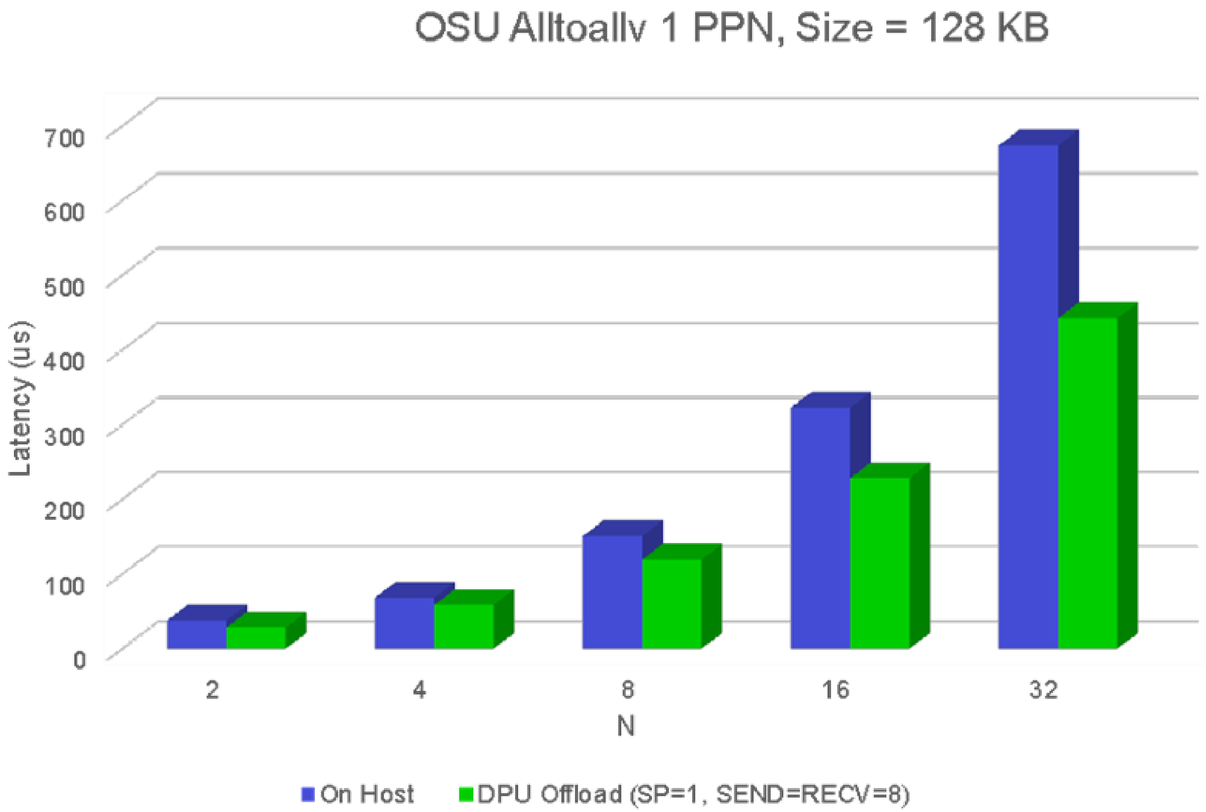
- Run a compiled application against a communication library
  - No application changes
  - Communication library treats the SmartNIC as it would any other communication medium
- Modify the application use communication routines that leverage the SmartNIC's capabilities
  - Example: Change the application implementation so that it provides opportunities for overlapping computation and communication and switch from using blocking to nonblocking communication library primitives

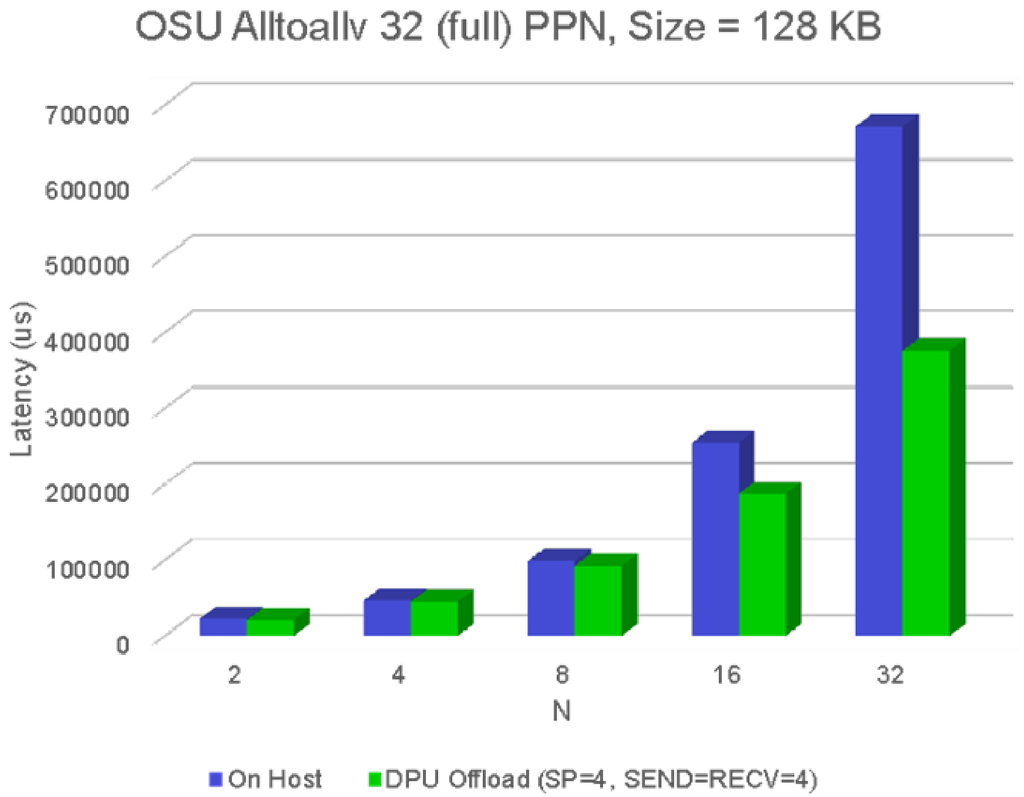
High-level Programming Models

Open MPI DPU support

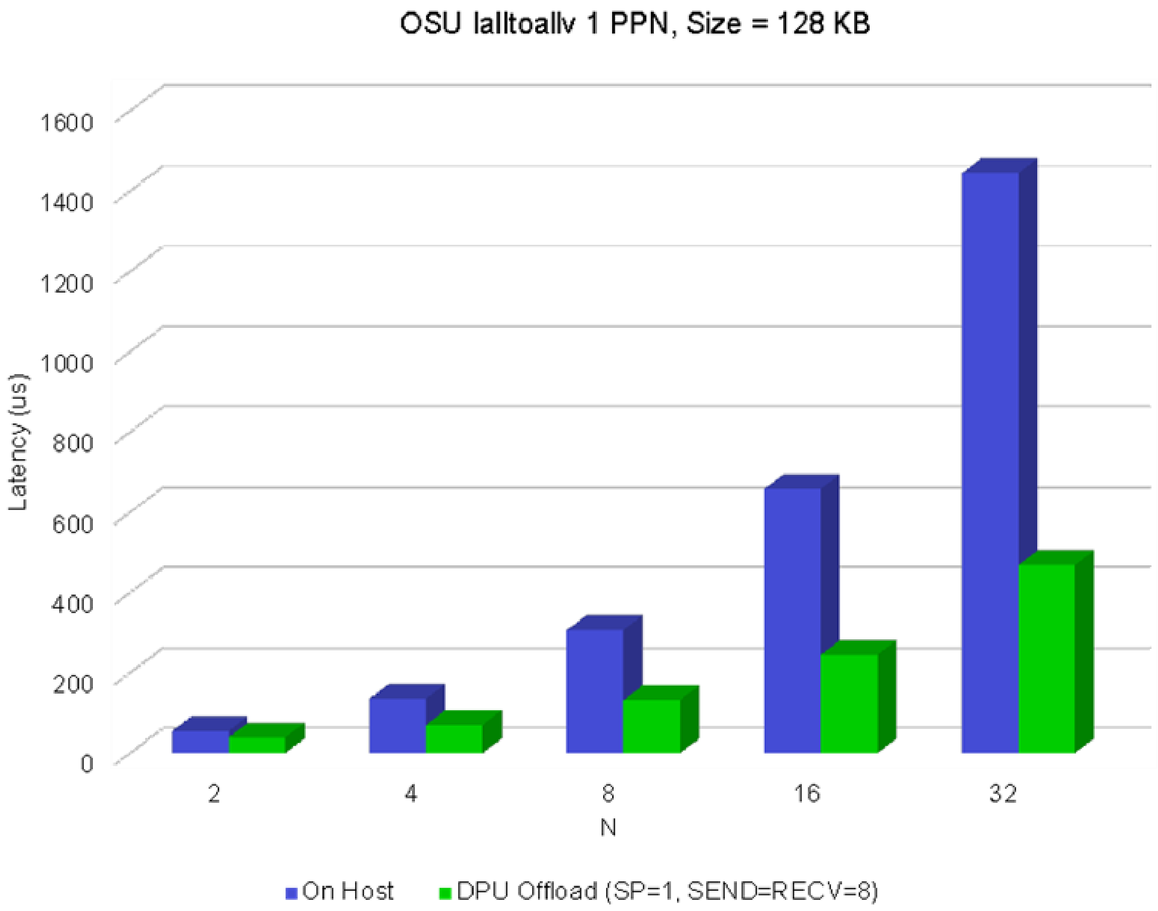
- Collectives supported: Alltoall, Alltoallv, Allgather, Allgatherv
- Enabled via mpirun
  - To select appropriate algorithm that will run on the DPUs
  - Works with Open MPI versions that supports Using Unified Communication Collectives (UCC)
- No application changes are required

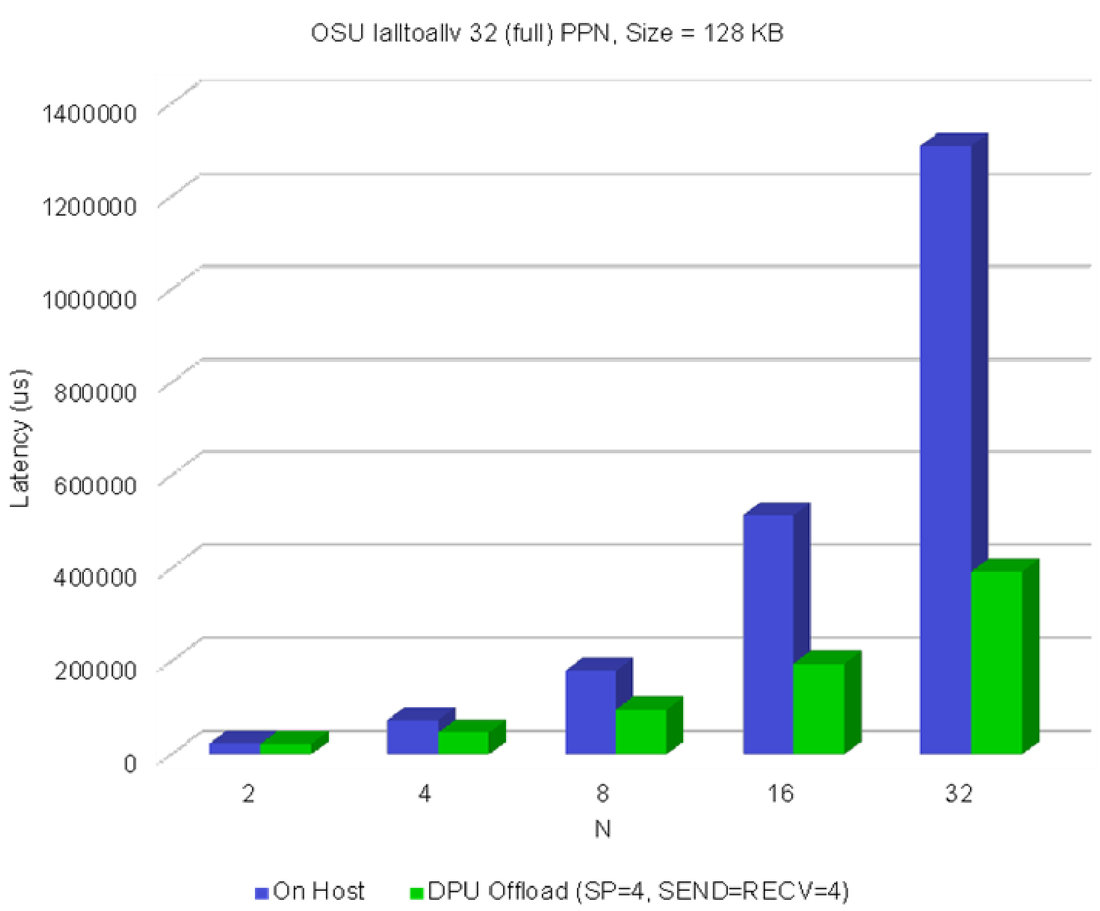
Alltoallv Latency



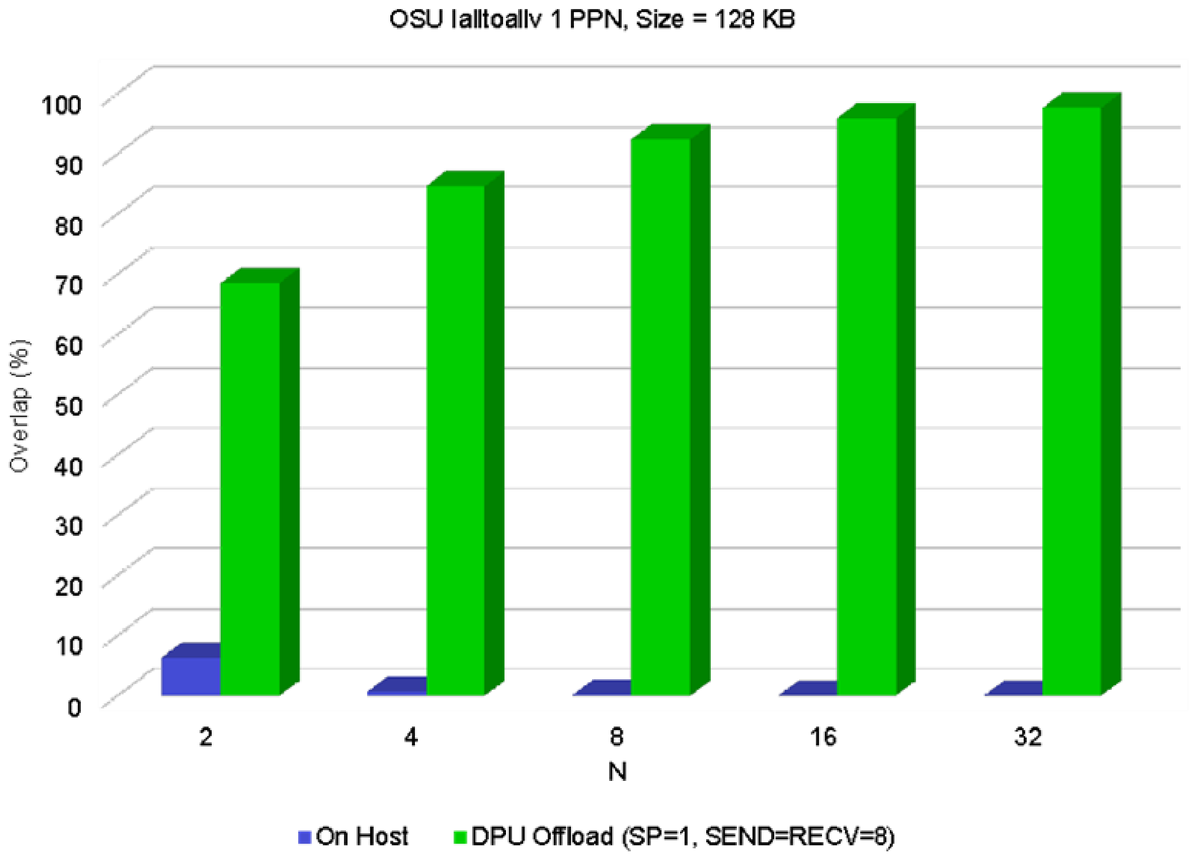


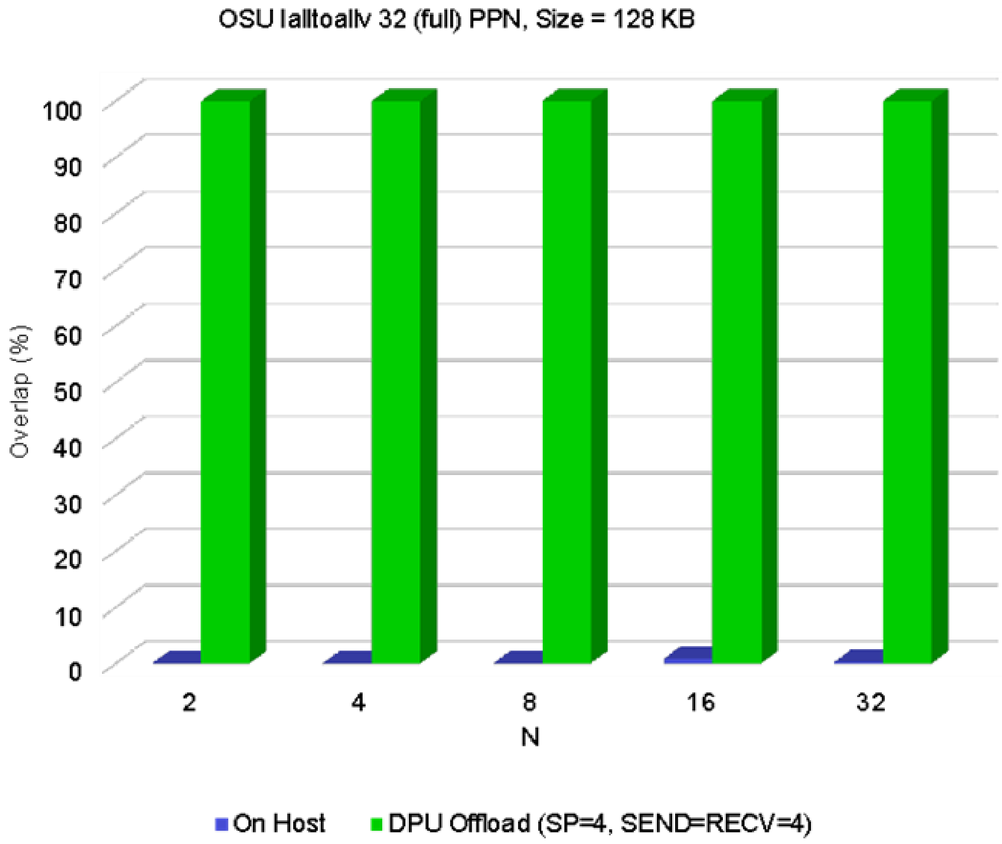
iAlltoallv Latency





iAlltoallv Computation/Communication Overlap



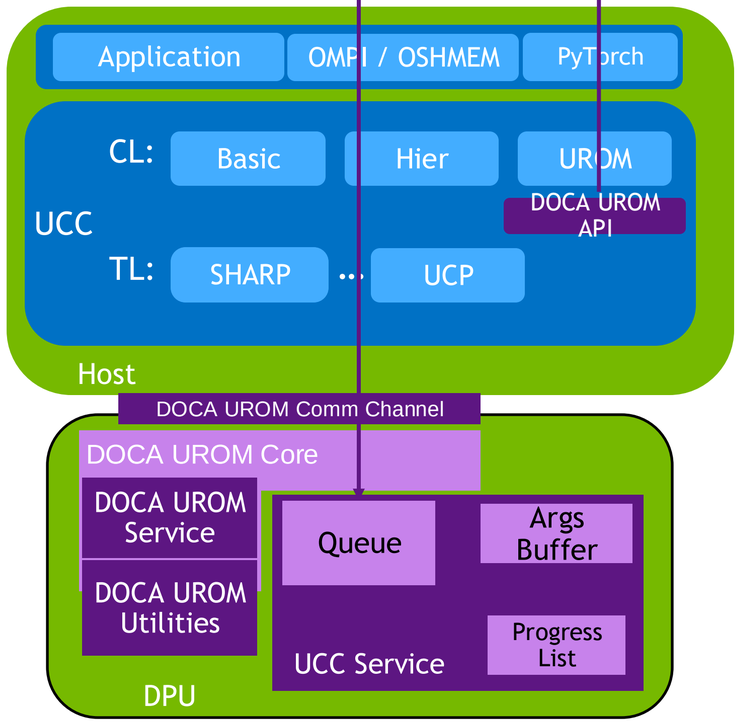


DPU UCC Offload with one-sided collectives

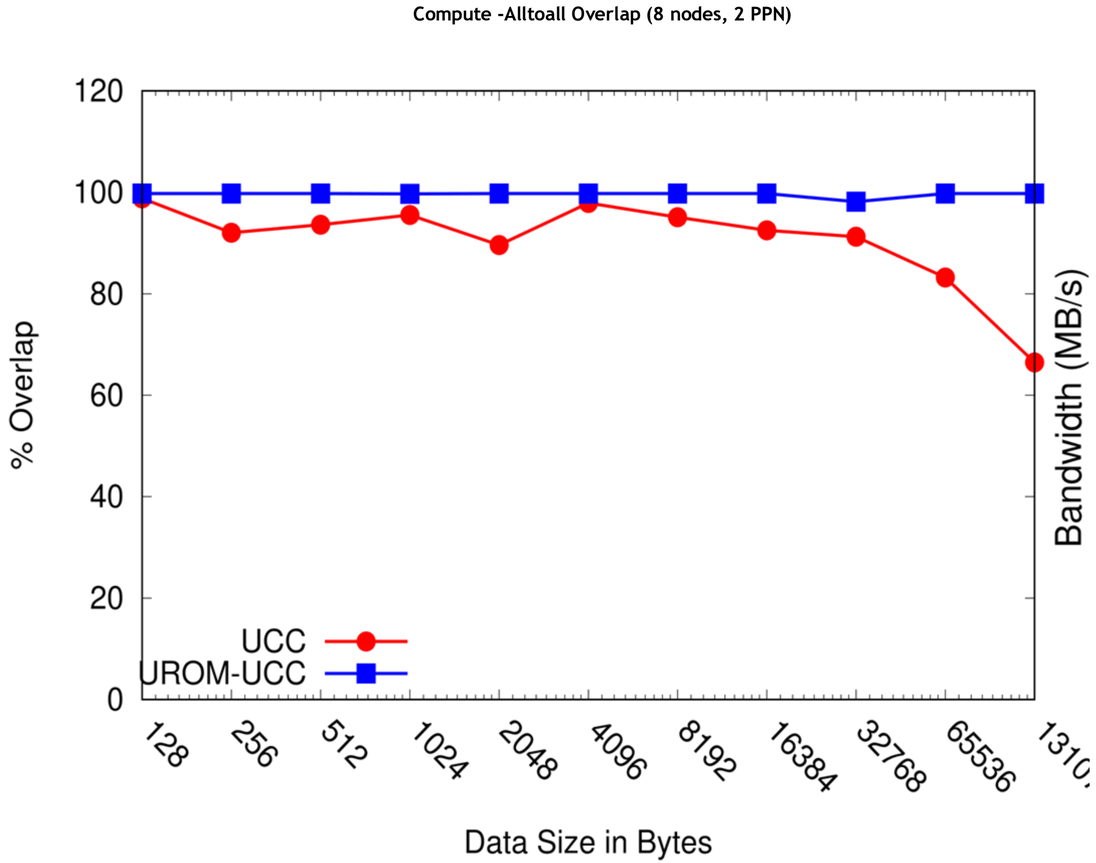
- Complete Offload of UCC library to DPU
  - Offloaded Collective Progress
    - Capable of 100% overlap
  - Capable of smart collective progress to minimize congestion
    - Leverage network telemetry and make informed decisions
- Enabled by UCC UROM CL
  - Execution of unmodified applications (e.g., OSHMEM / MPI applications)
  - Example:

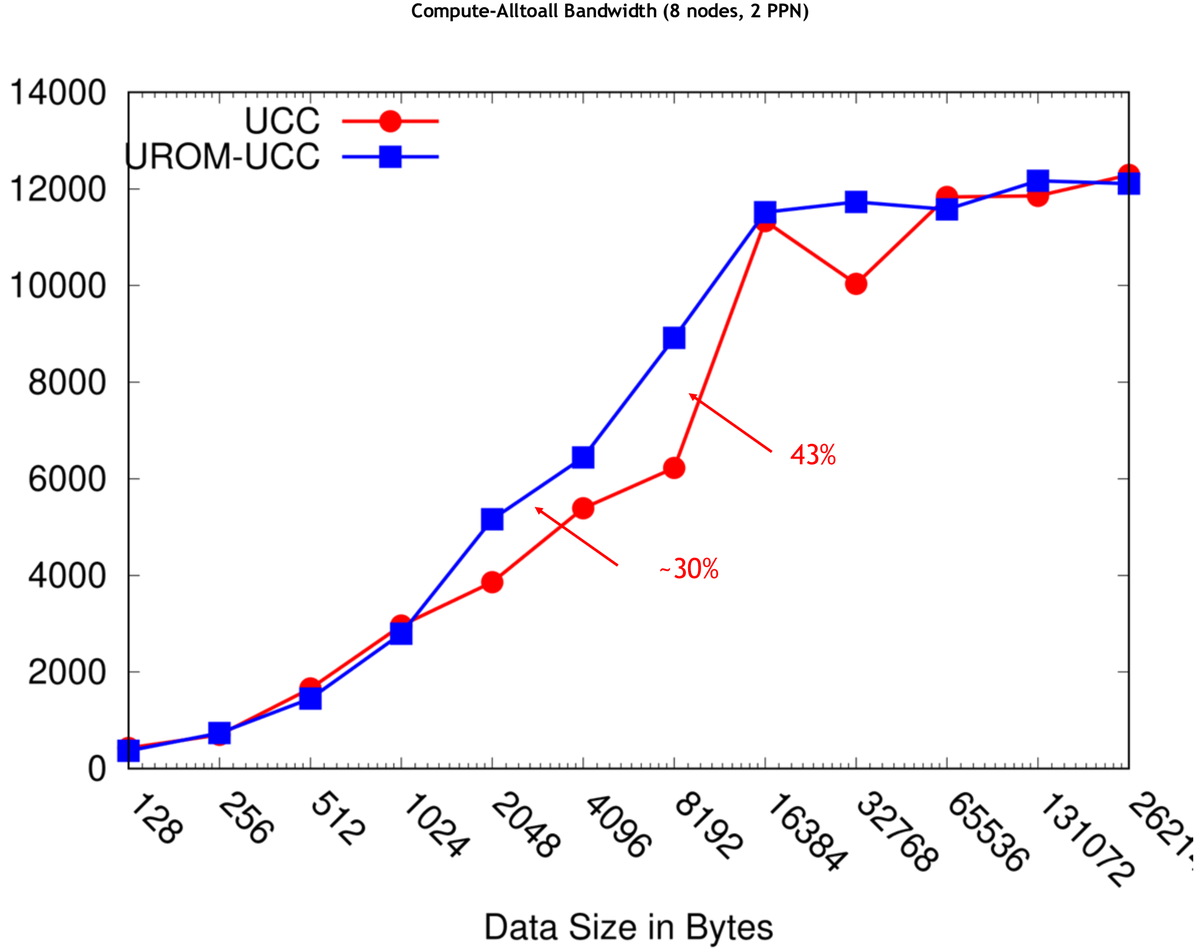
```
oshrun -np 4 -mca scoll_ucc_enable 1 --mca
scoll_ucc_priority 100 --mca scoll_ucc_cls urom,basic
./my_ucc_app
```

- Evaluated using Alltoall with compute micro-benchmark
  - Near 100% overlap with increasing data sizes
    - 25% improved overlap vs host due to offloaded progress
  - Up to 43% performance improvement for bandwidth
    - GET-based algorithm for small / medium messages
    - PUT-based algorithm for large messages



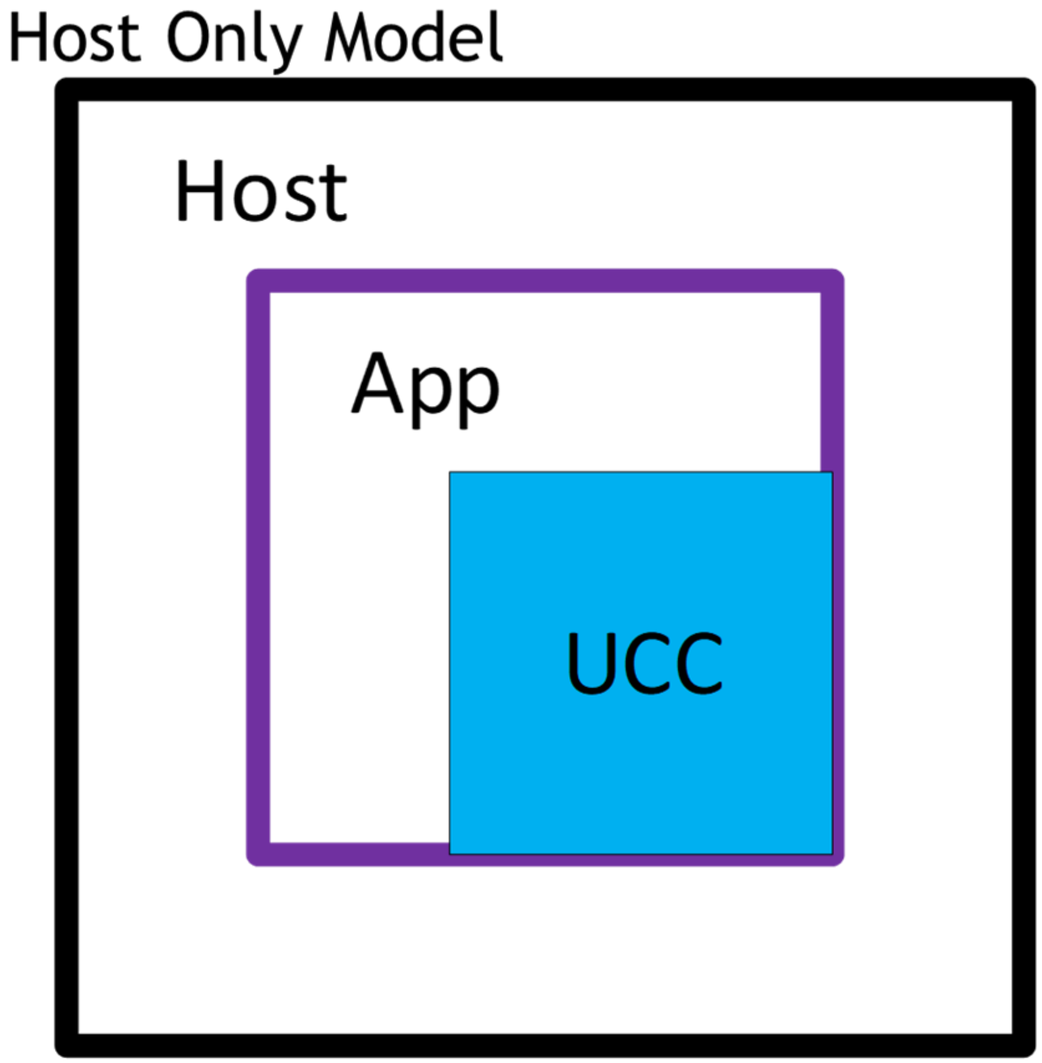
PERFORMANCE OF COMPUTE-ALLTOALL BENCHMARK

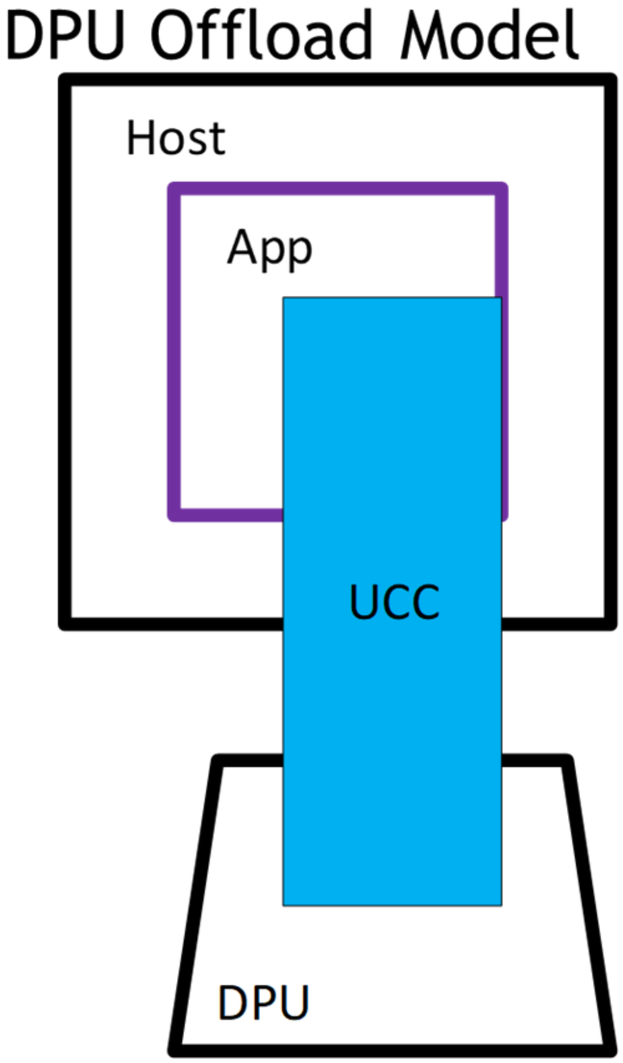




Using the SmartNIC as a Communication Accelerator

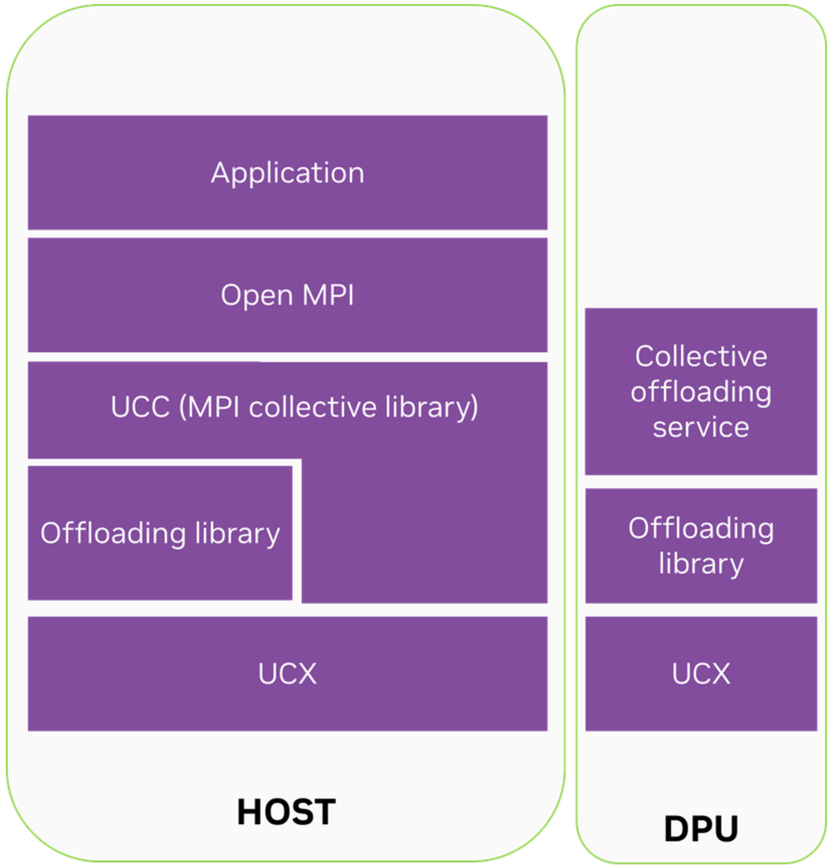
UCC Offload Model





Overview of the Software Stack

- UCX - to access hardware features (XGVMI)
- DOCA UROM - asynchronous agent to progress the offloaded collectives
- UCC - collective algorithms (selective)
- Open MPI (vanilla)
- User applications (vanilla)



UCC Offload Software Stack

Host

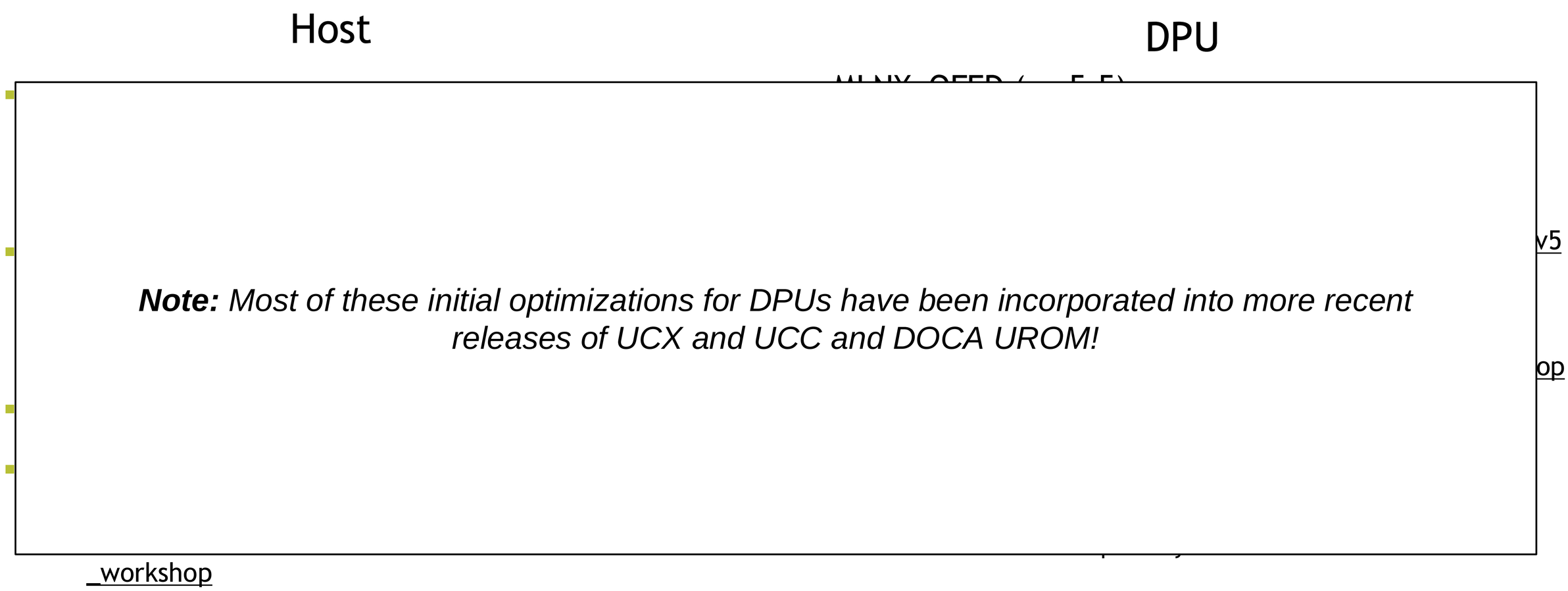
- DOCA_OFED (>= 5.5)
  - https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed
- UCX
  - https://github.com/yqin/ucx/tree/topic/dpu_offload_v5
- DPU Offload Service (DOCA UROM)
- UCC (example)
  - https://github.com/yqin/ucc/tree/topic/rdma_workshop
- Open MPI
  - https://github.com/open-mpi/ompi

DPU

- DOCA_OFED (>= 5.5)
  - https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed
- UCX
  - https://github.com/yqin/ucx/tree/topic/dpu_offload_v5
- DPU Offload Service (DOCA UROM)
- UCC* (example)
  - https://github.com/yqin/ucc/tree/topic/rdma_workshop

* UCC is not strictly required on the DPU, but the offload daemon and associated build script are located in this repository

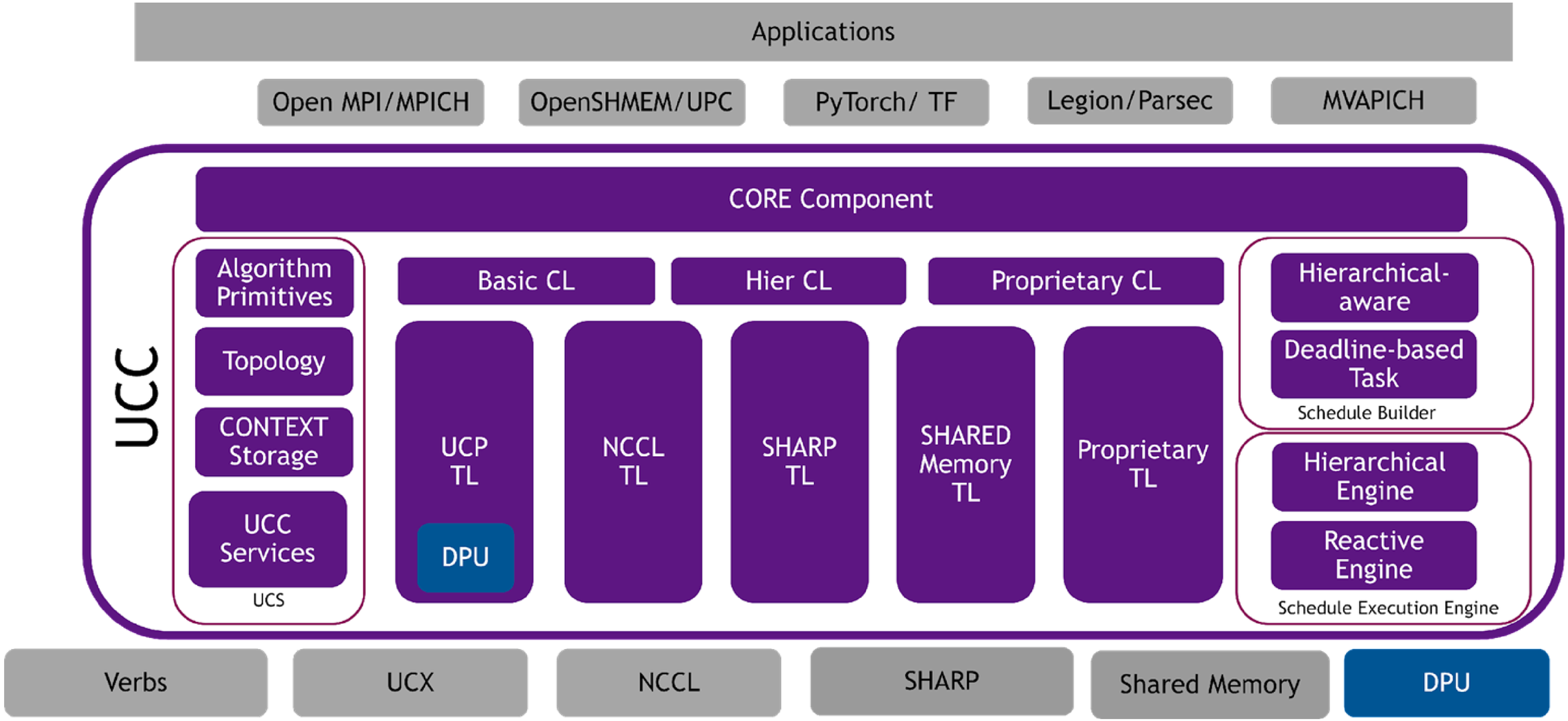
UCC Offload Software Stack



Note: *Most of these initial optimizations for DPUs have been incorporated into more recent releases of UCX and UCC and DOCA UROM!*

- Open MPI
  - https://github.com/open-mpi/ompi

Unified Collective Communication (UCC) Architecture

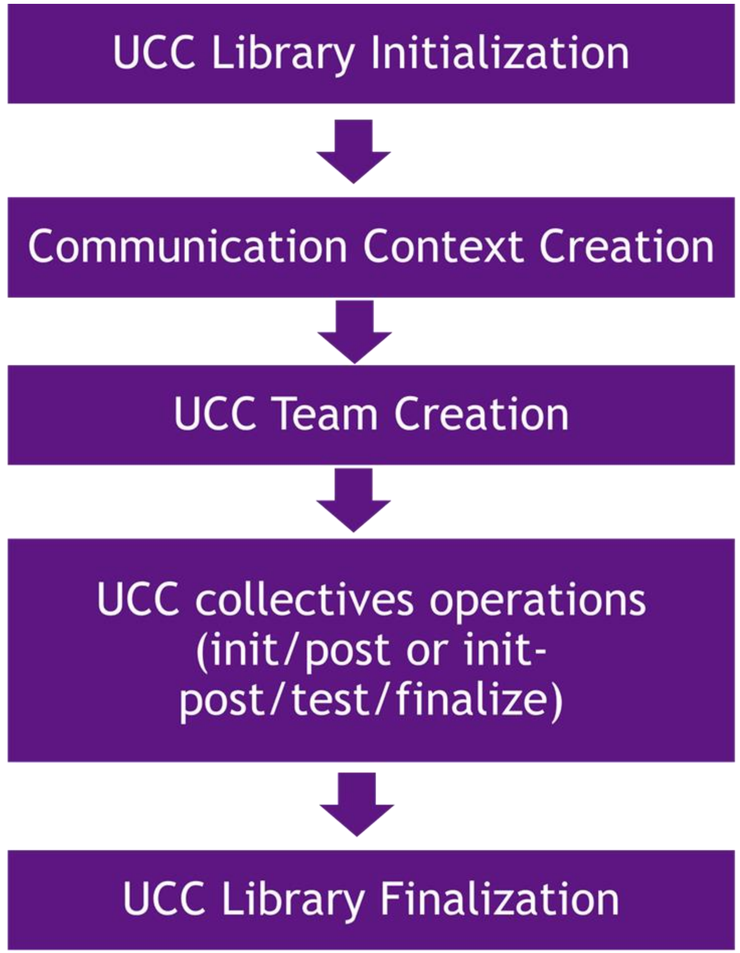


UCC Key Concepts

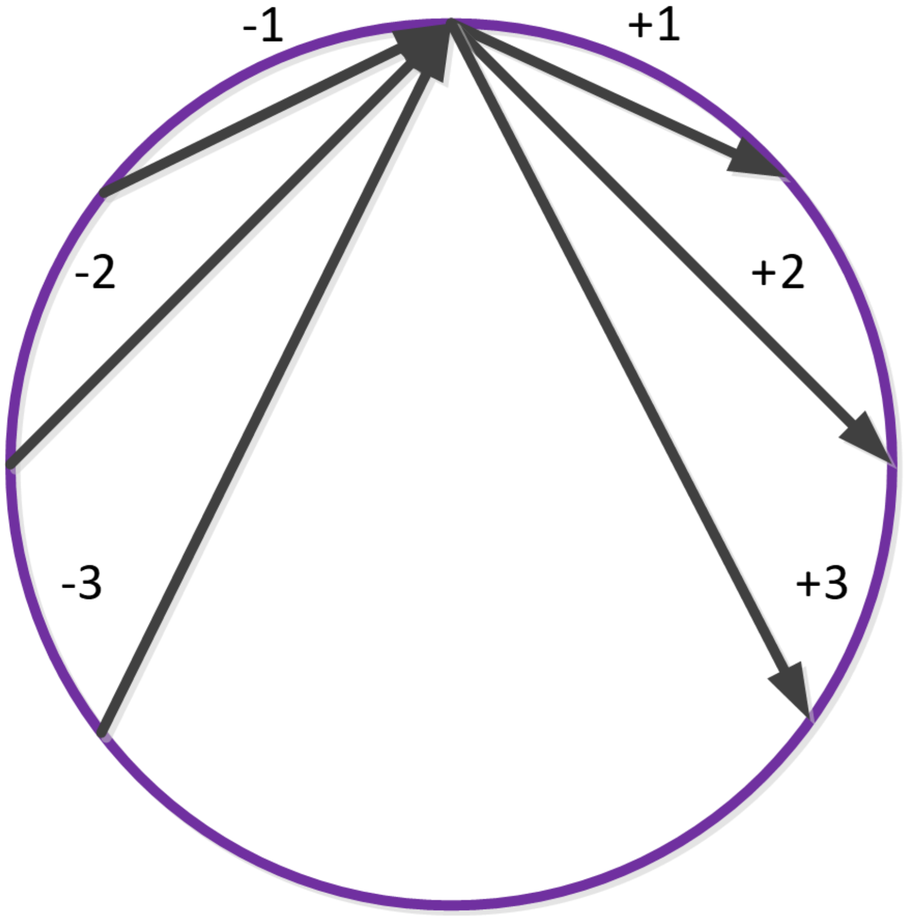
- Abstractions
  - Abstracts the resources required for collective operations
  - Local: Library, Context, Endpoints, Execution Engine
  - Global: Teams
- Operations
  - Defines how to interact with the abstractions
  - Create/modify/destroy the resources
  - Build, launch, and finalize collectives
- Properties
  - Defines how to customize abstractions and operations
  - Explicit way to request optional features, semantics, and optimizations (opt-in or opt-out model)
  - 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 is queried

UCC Code Flow

- Library Initialization
- Communication Context Creation
- Team Creation
- UCC Collective Operations
- Library Finalization



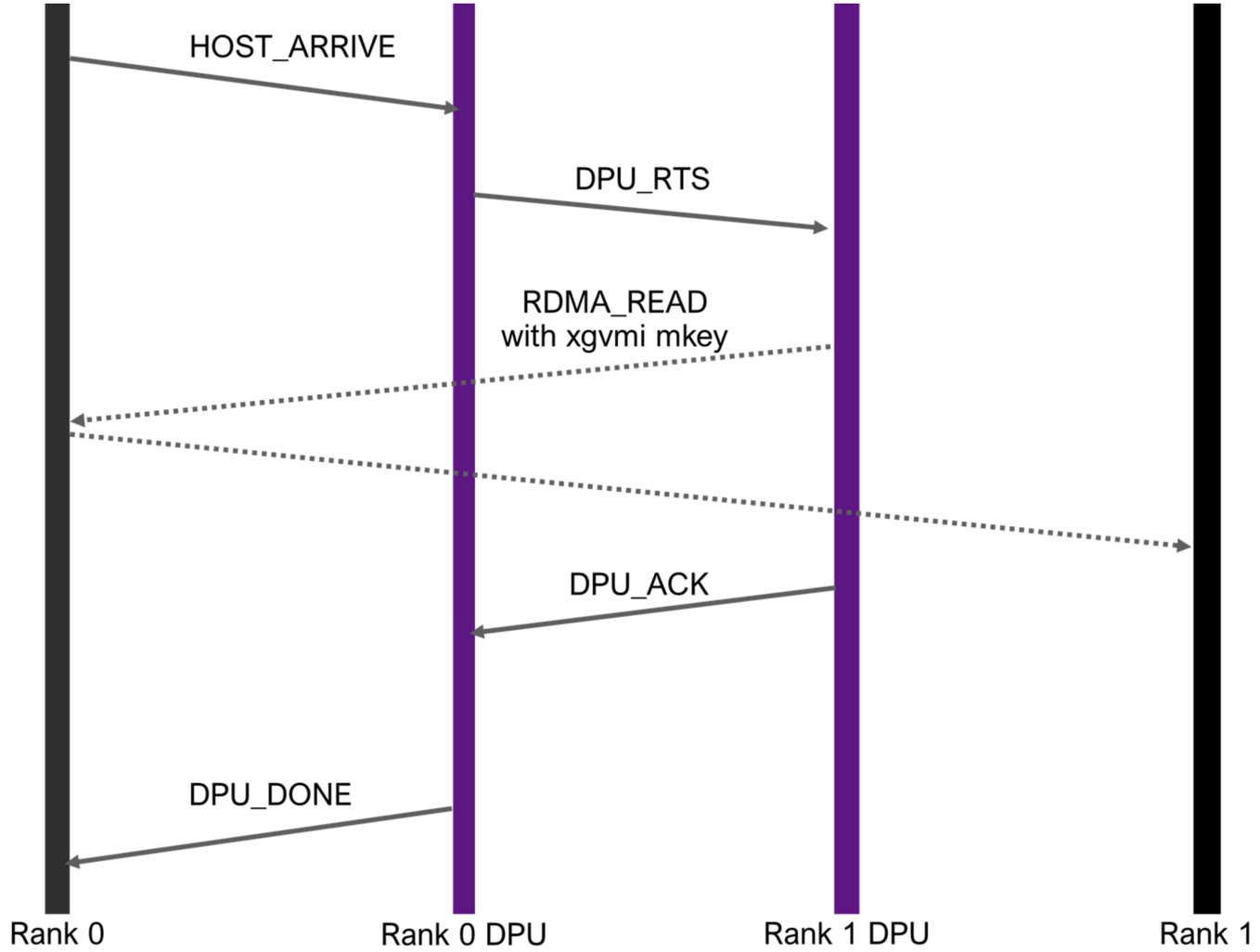
Host-Based Algorithm (Ring)



DPU Offload Algorithm

Sender: Rank 0

Receiver: Rank 1



Pros and Cons of DPU Offloaded Allgatherv

- Advantages
  - Better utilization of network bandwidth
  - Frees up CPU cycles on the host
- Disadvantages
  - Not suitable for latency sensitive messages sizes (small message sizes)

A Brief Intro to DPU (BlueField-3) Security

Security - Functions

Adapter SoC Security

- Hardware Root-of-Trust (RoT)
- Side-Channel/glitches mitigation
- Device Identifier Composite engine (DICE)

Performance

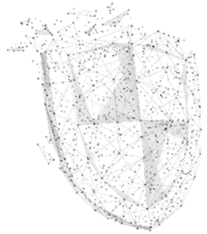
- Wire speed enforcement
- TLS offload engine
- IPsec inline offload
- Zero to very low CPU utilization on security tasks

Data-at-rest/in-motion Encryption

- Encryption offloads: IPsec, MAC-SEC, TLS, AES-XTS, AES-GCM
- Encrypted memory
  - AES-XTS

Security Applications and Services

- Deep Packet Inspection
- Threat Detection
- Key management (KMIP, PKCS#11, TPM)
- Isolated Execution



Security - Physical Protection

Trusted Execution Environment

- Hardware Root-of-Trust (RoT)
- Device Identifier Composite engine (DICE)
- SPDM support

Key Management

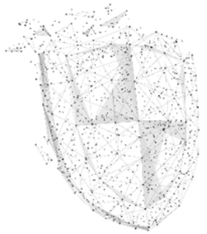
- Anti-DPA/SPA protection
- Confidentiality preserving Key transport
- FIPS 140-2 level 2 CAVP
- TRNG/DPRG/CRNG
- KMIP, PKCS#11, TPM

Anti- Fault Injection

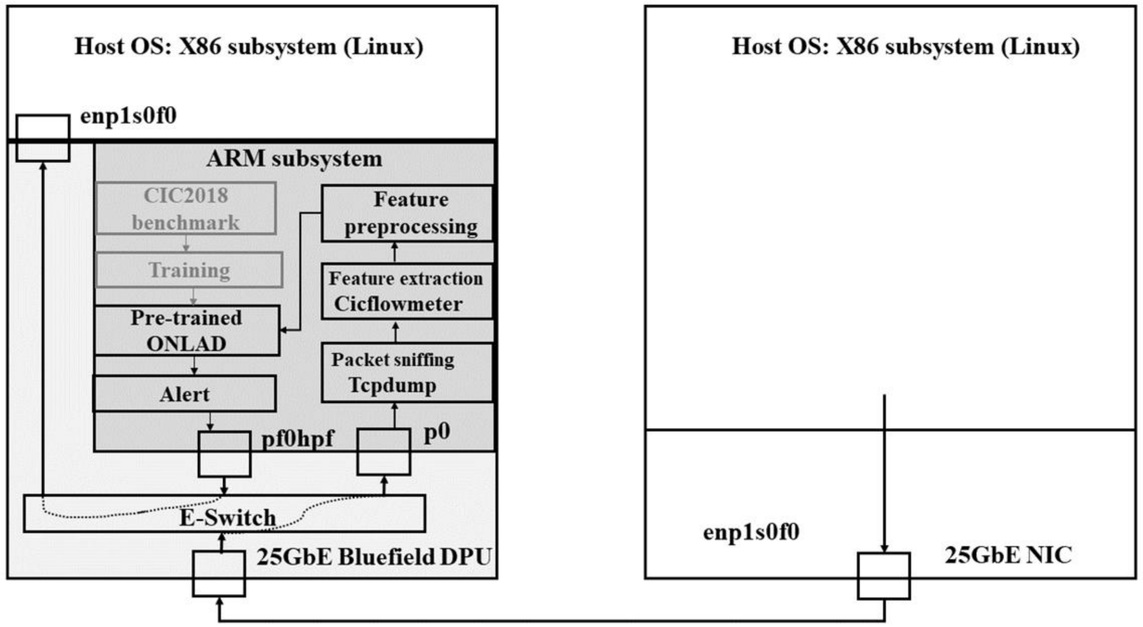
- Glitch mitigations (Frequency, Voltage)
- Canary core for anti-tampered execution
- Double detection/ Hardware Erasure SRAM
- Hamming distance for critical configuration

Secure Debug and Testing

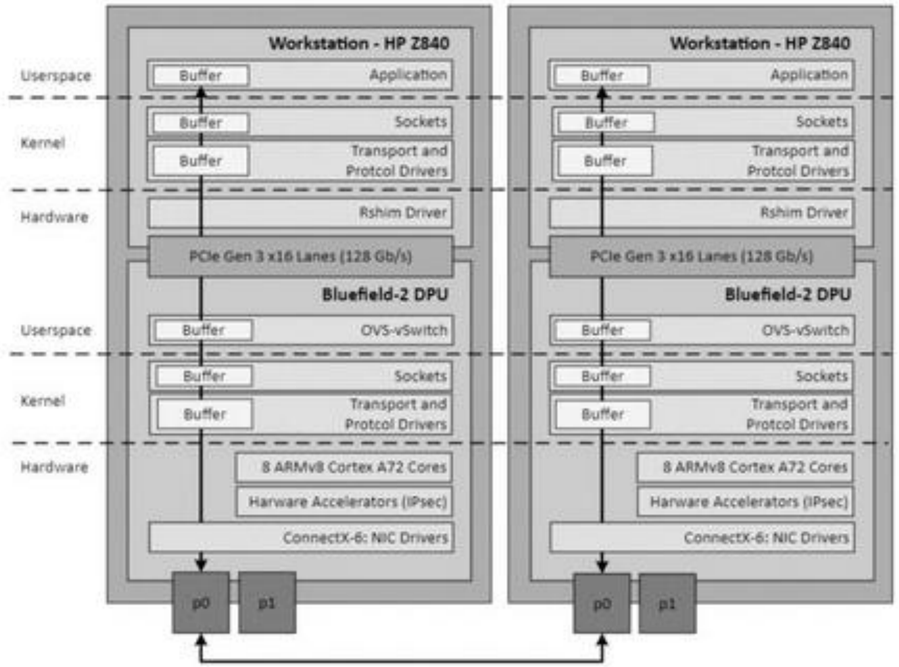
- Non-scannable regions
- Key access restriction



Security Related Research on DPUs



Man Wu, Hiroki Matsutani, and Masaaki Kondo. "ONLAD-IDS: ONLAD-Based Intrusion Detection System Using SmartNIC." In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; (HPCC/DSS/SmartCity/DependSys)*, pp. 546-553. IEEE, 2022.



From Noah Diamon Scott Graham, and Gilbert Clark. "Securing InfiniBand networks with the Bluefield-2 data processing unit." In *International Conference on Cyber Warfare and Security*, vol. 17, no. 1; 2022

- Recent work has looked at using frameworks like DOCA to implement zero trust architectures, but the addition of processing cores and a full OS stack to the DPU allows for the design of sophisticated IDS and secure network stacks

Break

Time (EDT)	Topic
8:30 - 8:40	Introduction and Attendee Survey
8:40 - 9:10	SmartNIC introduction and overview
9:10 - 9:20	Programming approaches for HPC with SmartNICs
9:20 - 9:35	SmartNIC Research – State of the Art
9:35 - 10:00	DOCA MPI Implementations; Security Topics
10:00 - 10:30	**Break**
10:30 - 11:10	OpenMP offload to the SmartNIC; ODOS Demos
11:10 - 11:55	Hands On - Application Experiences
11:55 - 12:00	Wrap-up

See the updated agenda at **https://github.com/gt-crnch-rg/smartnic-tutorial-sc24/**

OpenMP Offload to DPUs

Brought to you by the AccelCom Group at BSC
M. Usman, S. Iserte, **A. J. Peña** - accelcom@bsc.es

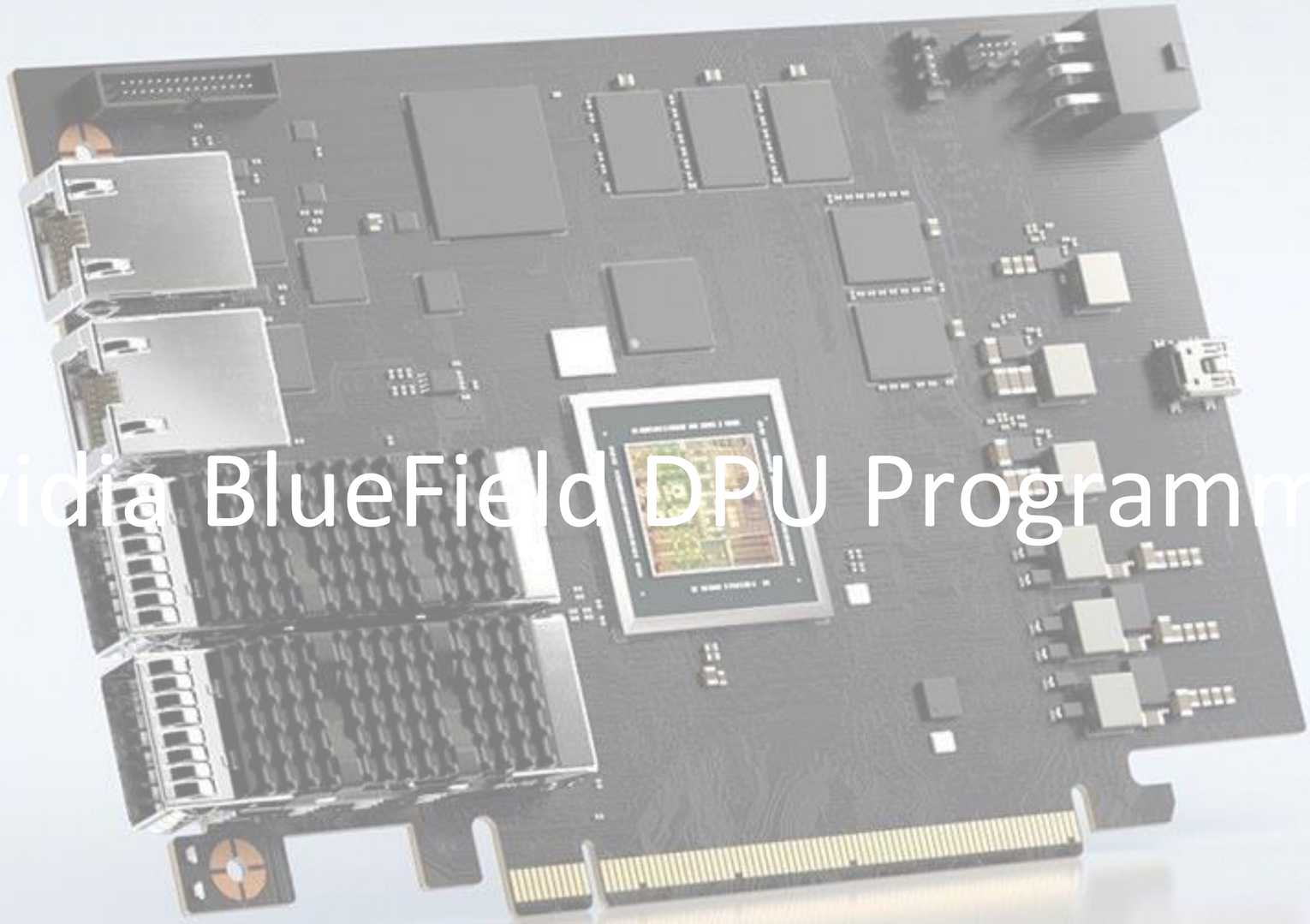
Outline

- Nvidia BlueField DPU Programming
- OpenMP DPU Offloading Support (ODOS)
  - Demo #1
- ODOS MPI Support
  - Demo #2
- Use case: Halo Exchange
- Performance

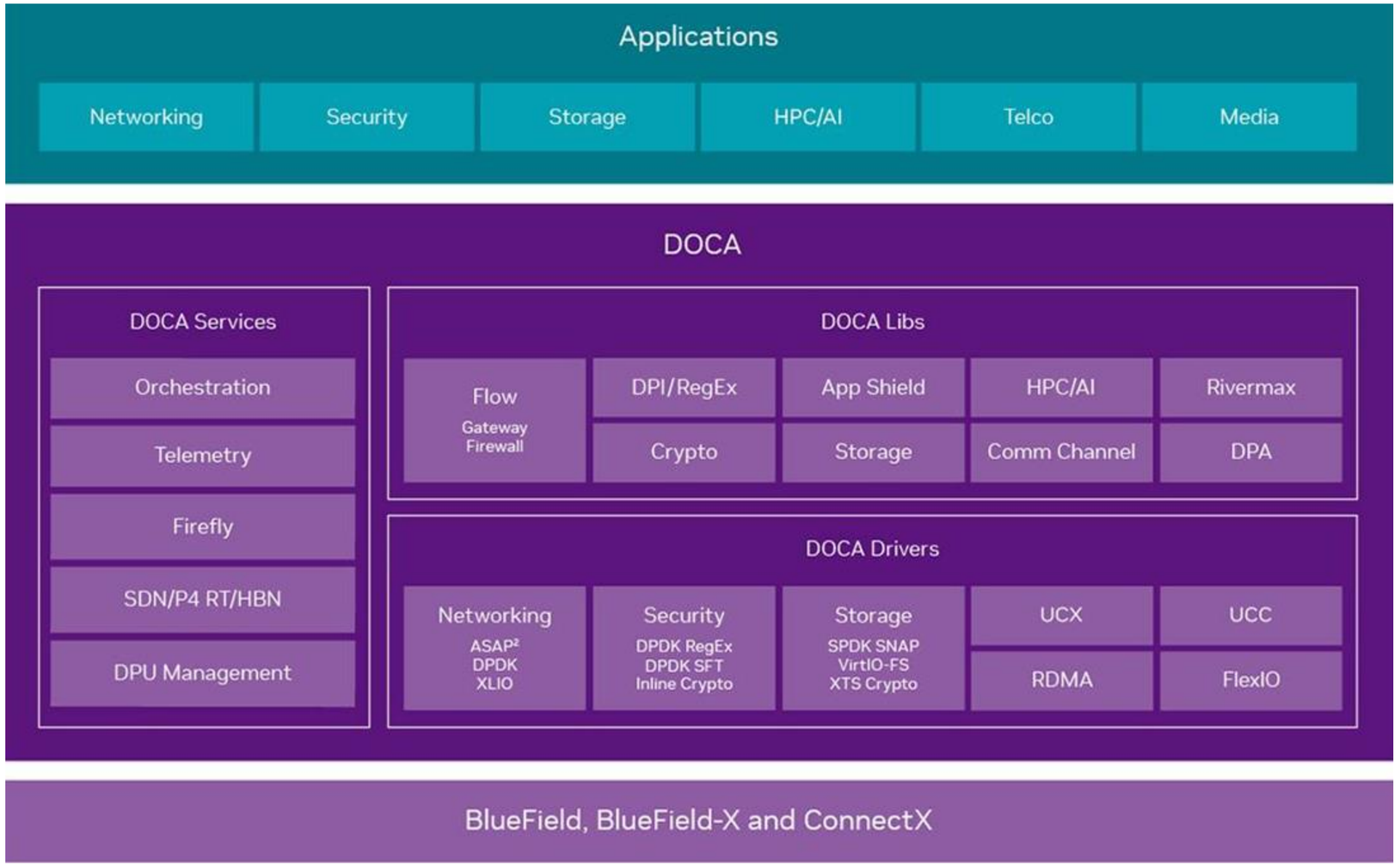
Sandia National Laboratories

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

NVIDIA

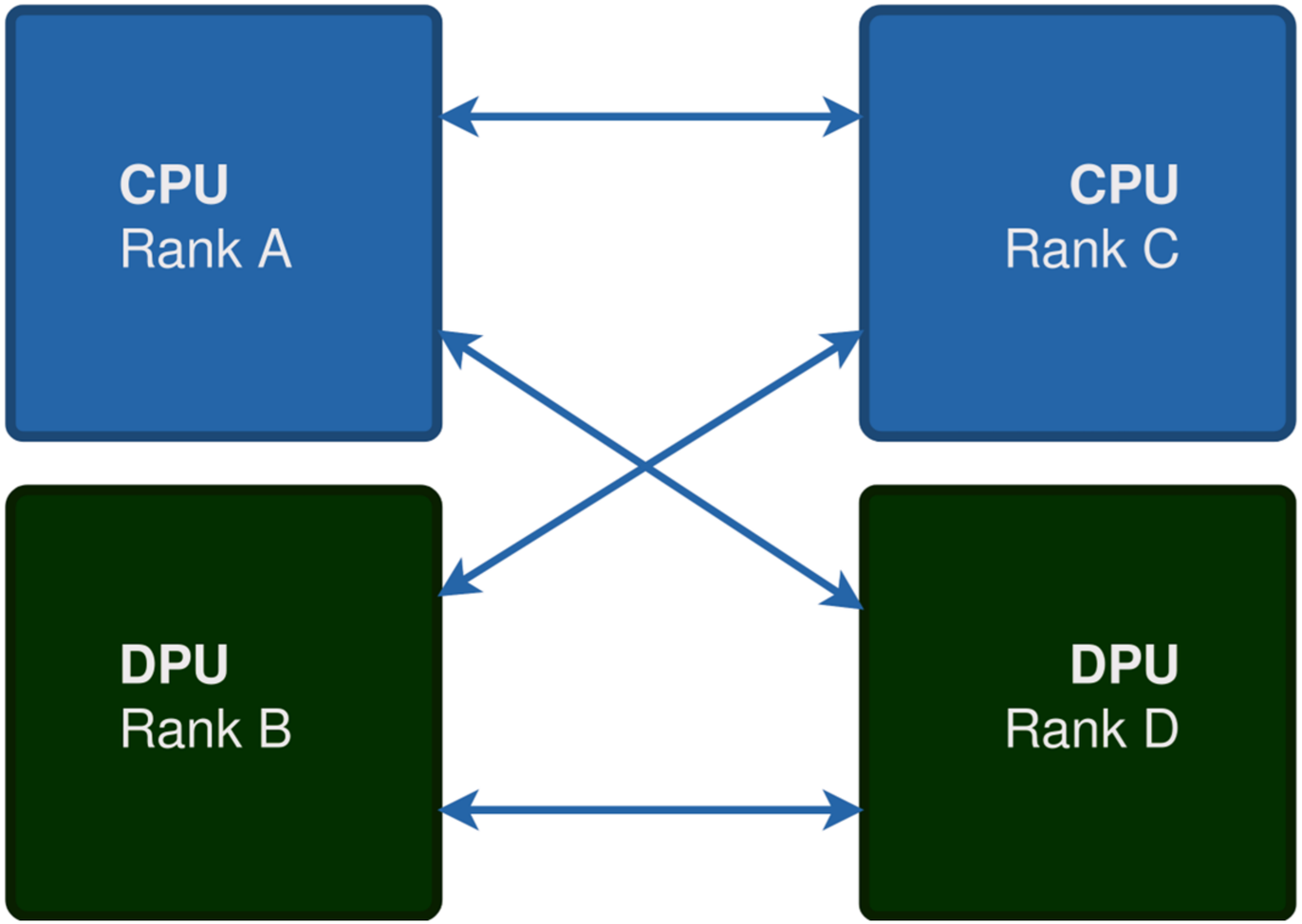


Nvidia BlueField DPU Programming



BlueField DPU Programming Low Level API

- DOCA is to DPUs what CUDA is to GPUs
- DOCA is not an offloading API
- Too low level for domain scientists



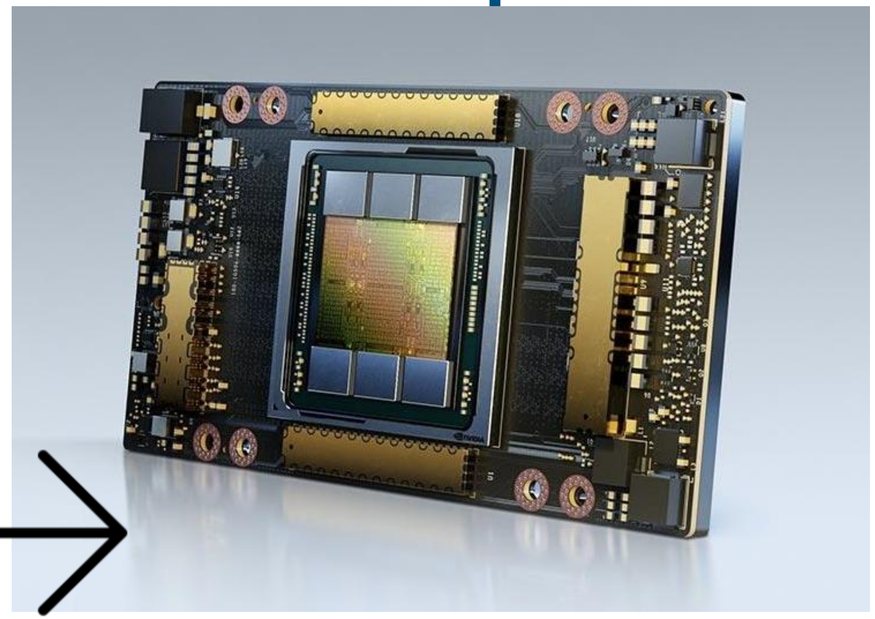
BlueField DPU MPMD Programming

- Multiple program multiple data execution model in MPI
- Poor productivity because the processors are widely different

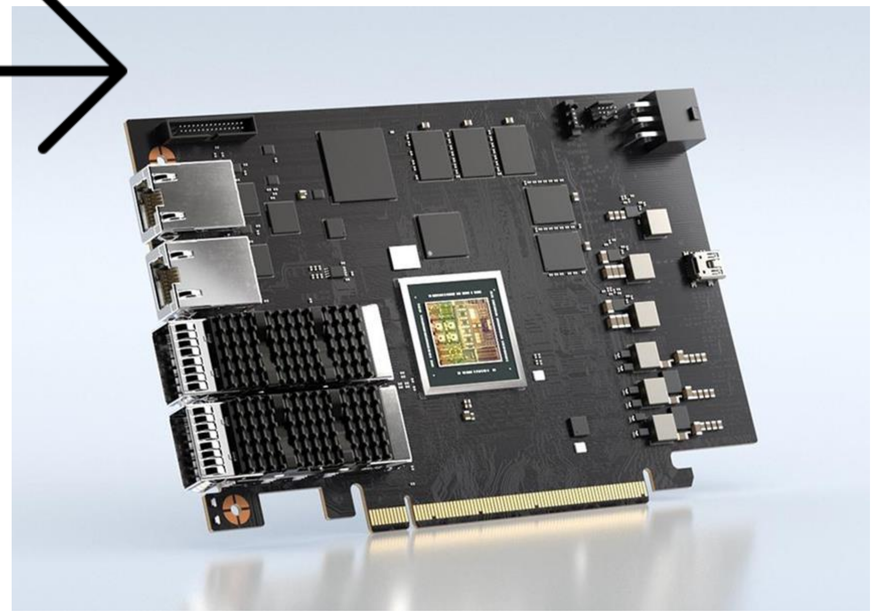
DPU Offloading Programming with standard OpenMP

```
int main ()
{
    /*************/
    /* exec in host */
    /*************/

    #pragma omp target device (  )
    {
        /***************/
        /* exec in target */
        /***************/
    }
}
```

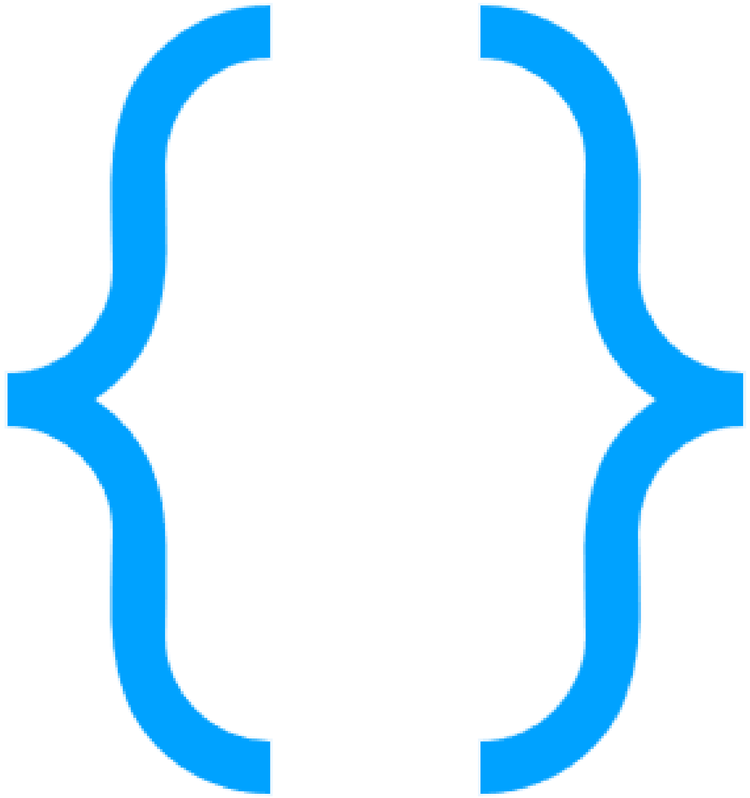
GPU



DPU

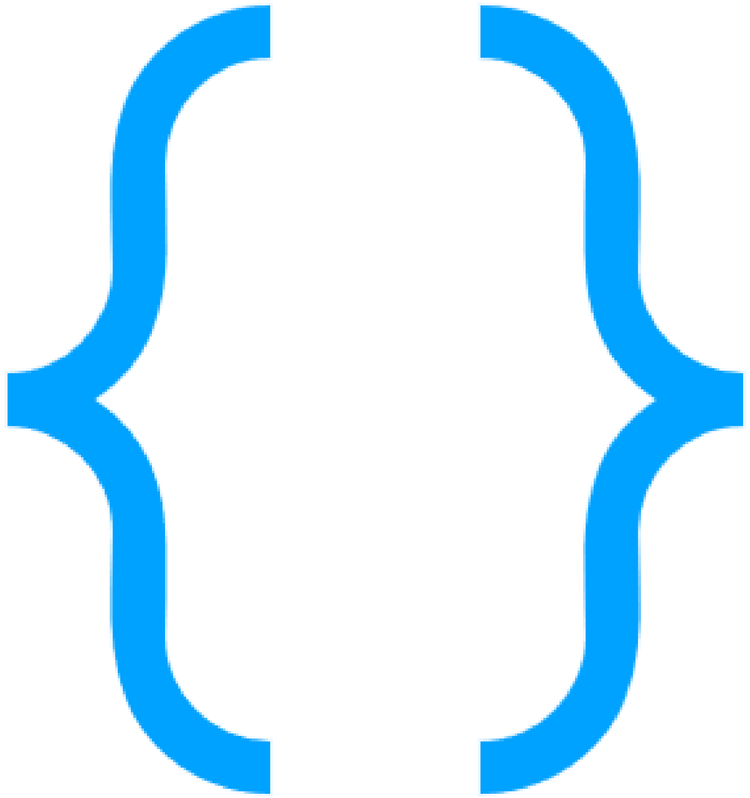


Basic Syntax



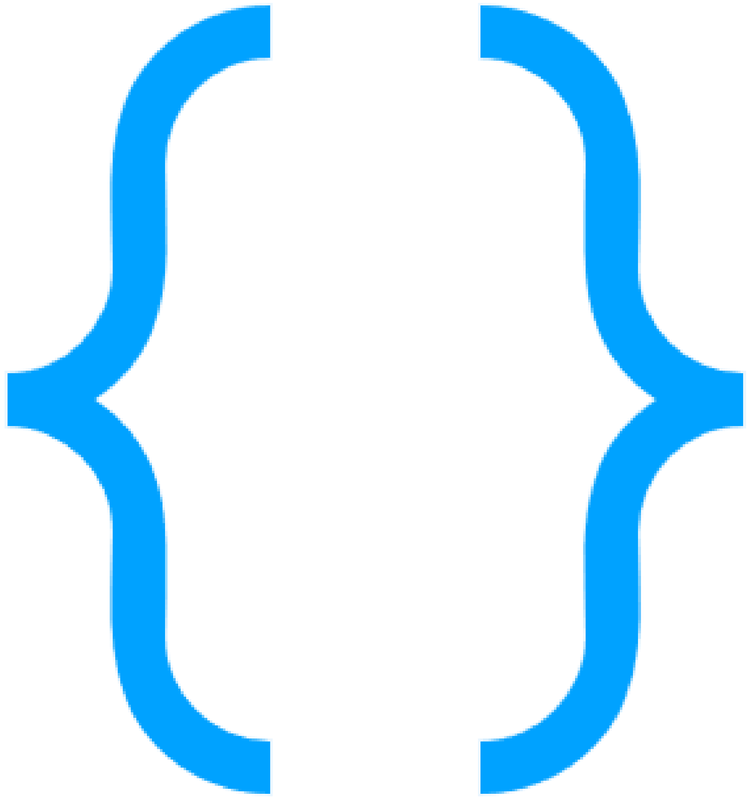
```
// exec in host

#pragma omp target
{
   // exec in target
}
```

How do I Know the Device Number?



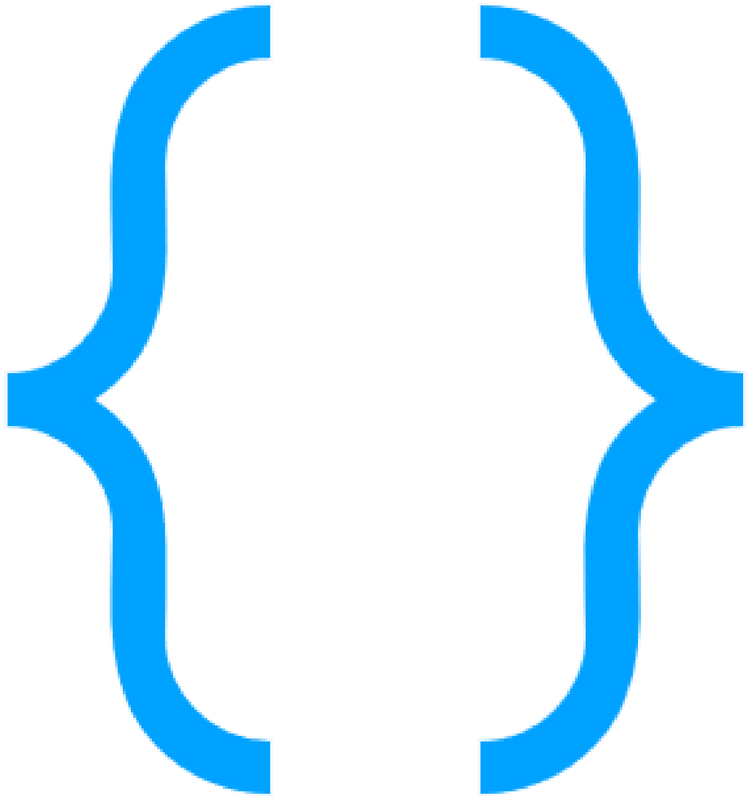
```
$ llvm-omp-device-info

Device (4):
    CUDA Driver Version:                    10020
    CUDA Device Number:                     0
    Device Name:                            Tesla V100-SXM2-16GB
    Global Memory Size:                     16911433728 bytes
    Number of Multiprocessors:              80
    Concurrent Copy and Execution:          Yes
    Total Constant Memory:                  65536 bytes
    Max Shared Memory per Block:            49152 bytes
    Registers per Block:                    65536
    ...
    Compute Capabilities:                   70
```

How do I Know the Device Number?



```
$ llvm-omp-device-info

...
Device (7):
    BlueField DPU device
        iface     :  ib0
        ib dev    :  mlx5_2
        doca id   :  2
        pci addr  :  42:00:0
        comm channel:
            max msg size             :   4080
            max send    queue size   :   8192
            max receive queue size   :   8192
...
```

How to Choose a DPU?



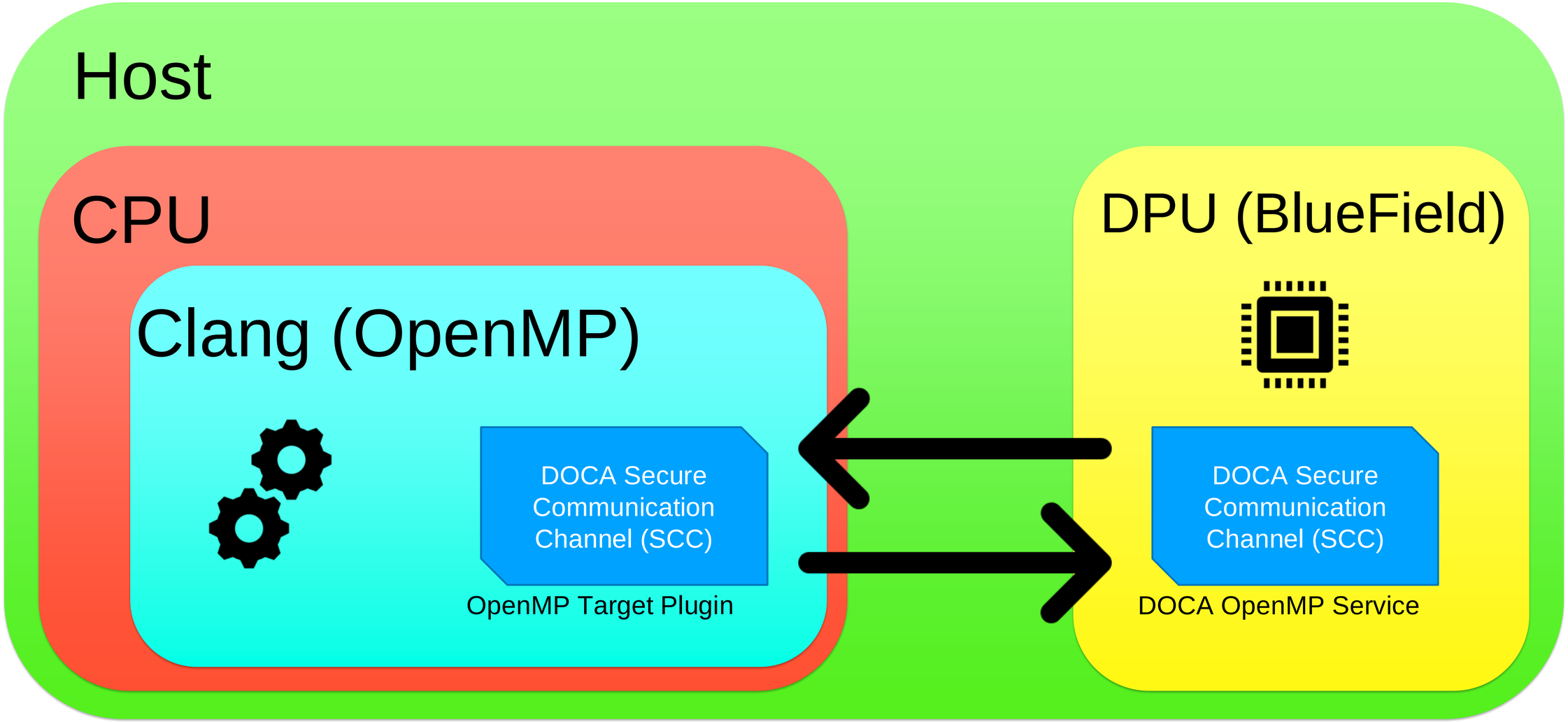
```
// exec in host
#define __DPUID__ 7
#pragma omp target device(__DPUID__)
{
   // exec in target
}
```

```
void main( int argc, char* argv[] ) {
    // Data initialization
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &dpurank );
    if ( dpurank )      // DPU rank receives data
        MPI_Recv( &buf, SIZE, MPI_INT, 0, … );
    else          // Host rank sends data
        MPI_Send( &buf, SIZE, MPI_INT, dpurank, … );
    for( int t = 1; t <= TIMESTEPS; t++ ) {
        if ( dpurank );    // Offloaded compute
        else;              // Host compute
    }
    if ( dpurank )      // DPU rank send data
        MPI_Send( &buf, SIZE, MPI_INT, 0, … );
    else          // Host rank receive data
        MPI_Recv( &buf, SIZE, MPI_INT, dpurank, … );
    MPI_Finalize();
}
```

```
void main( int argc, char* argv[] ) {
    // Data initialization
    #pragma omp target
        data map(tofrom:buf[0:SIZE]))
    {
        for( int t = 1; t <= TIMESTEPS; t++ ) {
            #pragma omp target
        {
                // Offloaded compute
        }
        // Host compute
        }
    }
}
```

OpenMP DPU Offloading Support

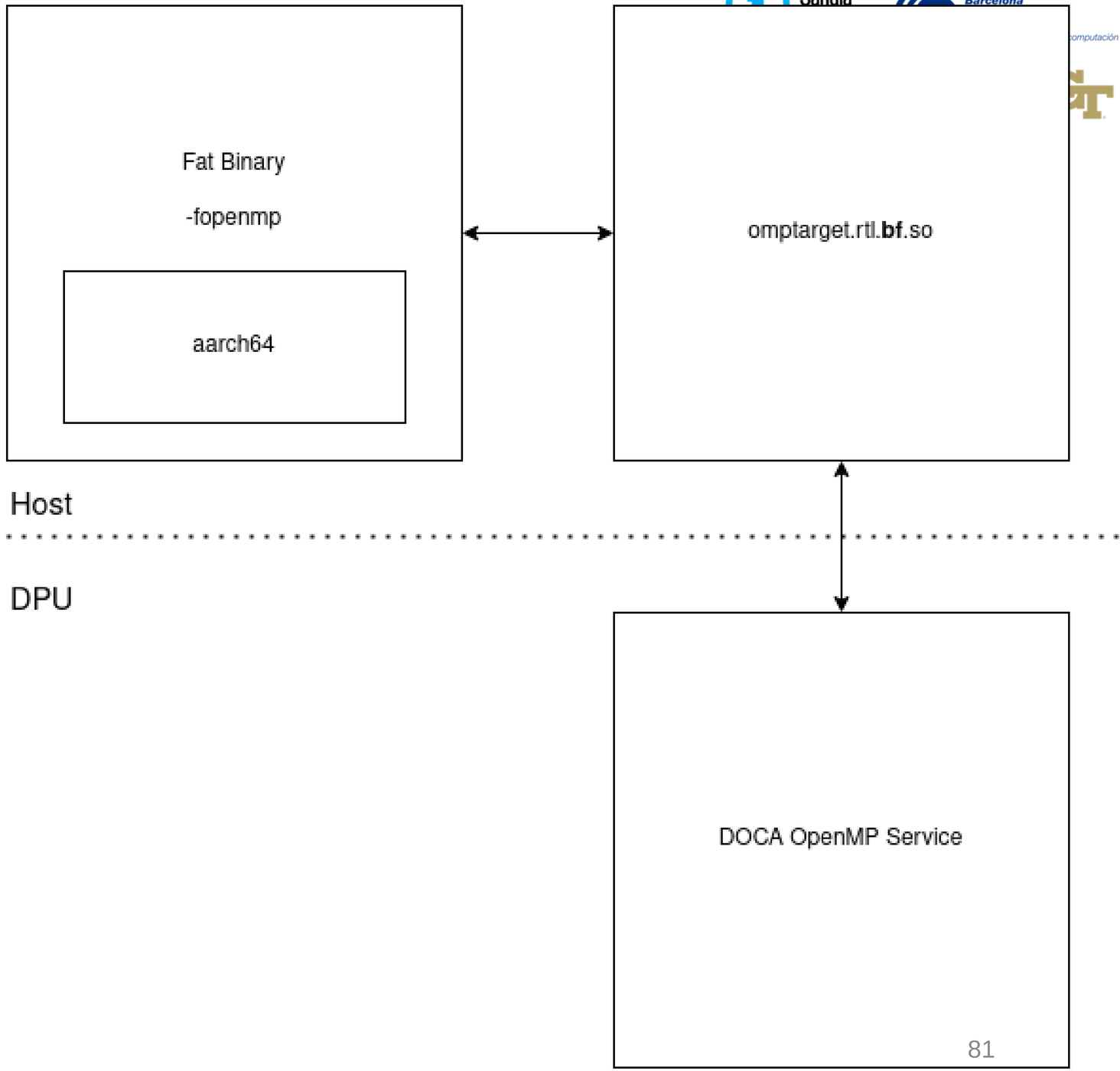
ODOS Architecture



M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23*, https://doi.org/10.1145/3624062.3624165

ODOS Modules

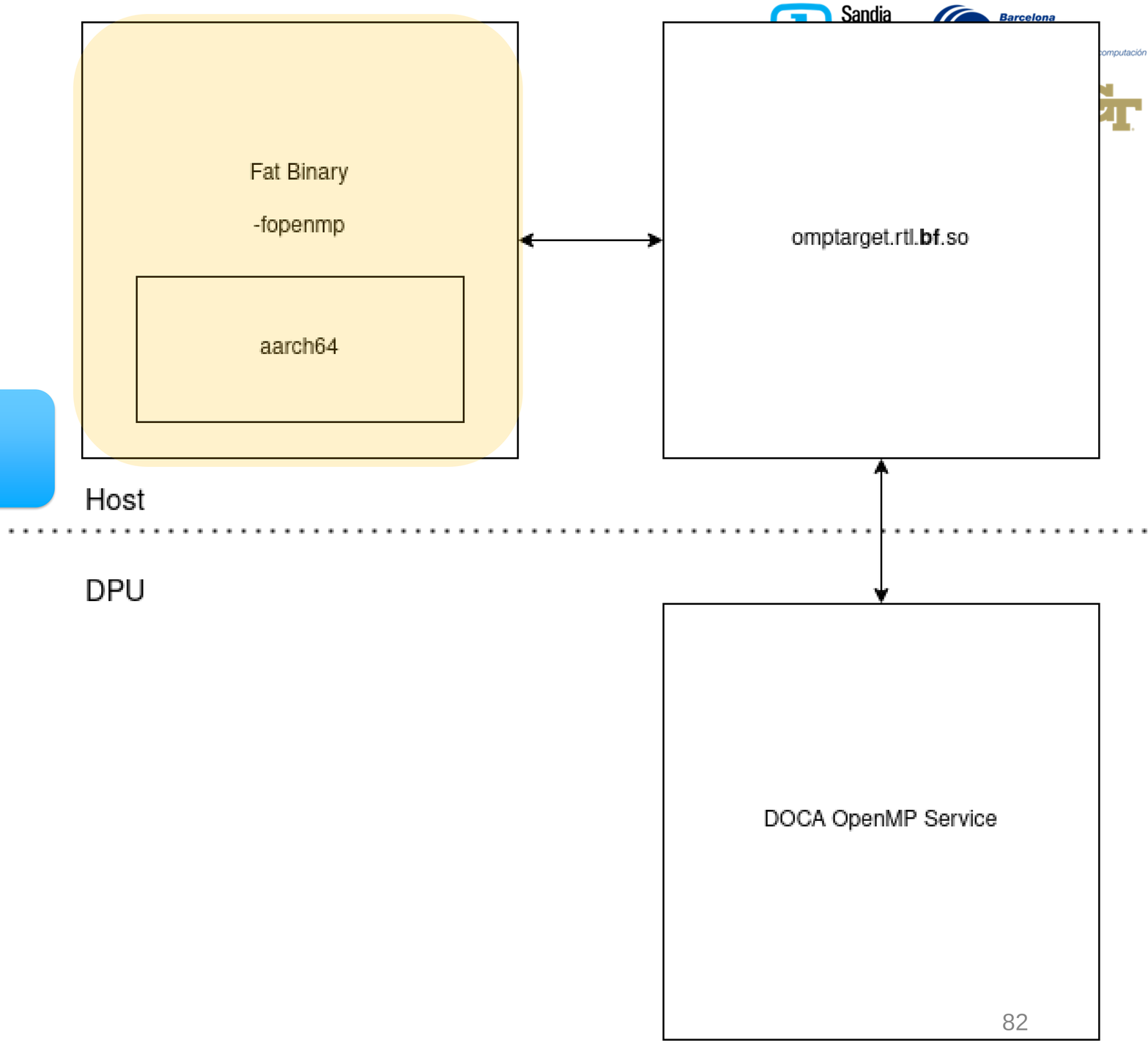
- Fat Binary Generation
- OpenMP Runtime (host)
- OpenMP Service (device)



Fat Binary Generation

Cross-compilation

- I.e., x86-64 & aarch64
- Default Linux GNU GCC compiler

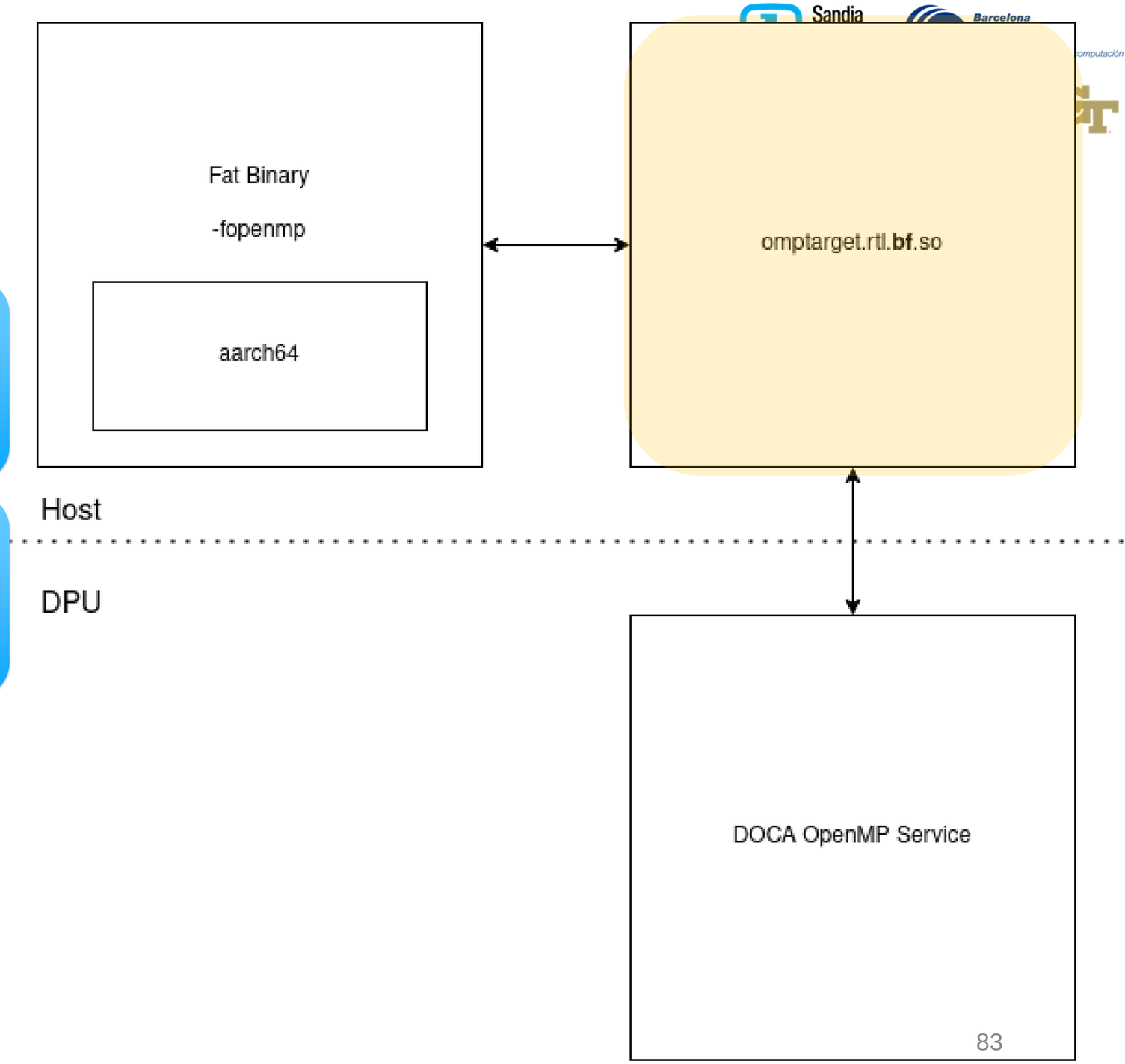


Target Plugin

Communication with the DPU using DOCA SCC

Send commands to

- Load binary
- Allocate and exchange data
- Execute target blocks



DOCA OpenMP Service

Receive commands using DOCA SCC

Load binary

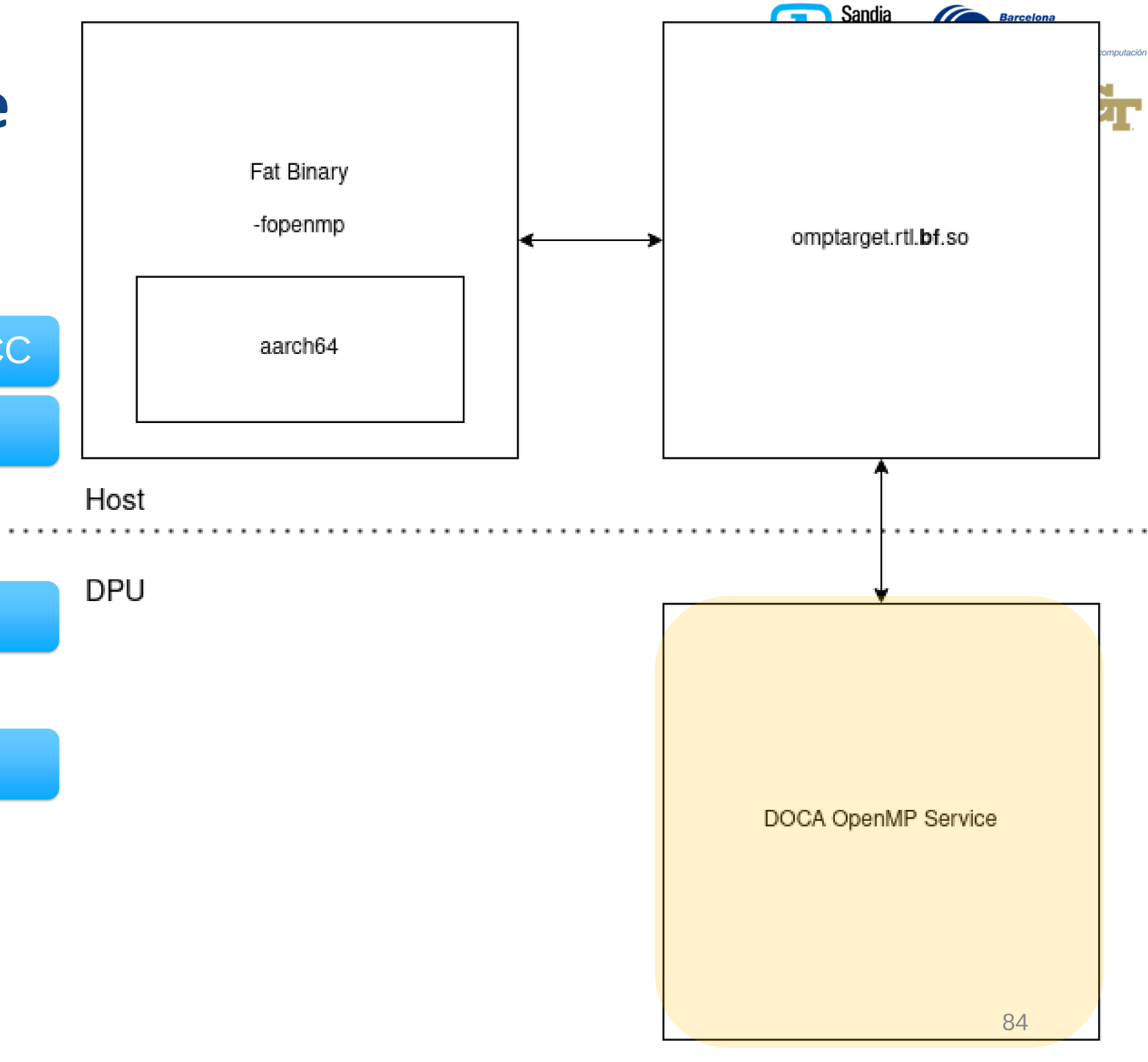
- Save image as a shared object
- Open the image [dlopen]
- Send back handles of symbols to host [dlsym]

Data operations

- Allocate/Delete [malloc/free]
- Send/Retrieve [DOCA SCC]

Target block execution

- Run target handle [ffi_prep_cif, ffi_call]



DEMO #1

- How to compile ODOS-enabled applications
- Basic example
- OpenMP "Parallel" feature
- Shared libraries
- Asynchronous TCP

How to Compile ODOS-enabled Applications

ODOS-enabled LLVM

Enable OpenMP and specify offloading targets

```
$ clang -fopenmp -fopenmp-targets=aarch64-unknown-linux app.c -o app
```

```c
#include <omp.h>
#include <stdio.h>

int main()
{
#pragma omp target
    puts("Hi Folks!");

    return 0;
}
```

Host

```
uthmanhere@thor013:~/omp_exp/labs_sc23/task_a/build$
```

DPU

```
uthmanhere@thorbf3a013:~$
```

```c
#include <omp.h>
#include <stdio.h>

int main()
{
#pragma omp target
#pragma omp parallel
        puts("Hey! OpenMP even work in DPU...");

        return 0;
}
```

Host

```
uthmanhere@thor013:~/omp_exp/labs_sc23/task_b/build$
```

DPU

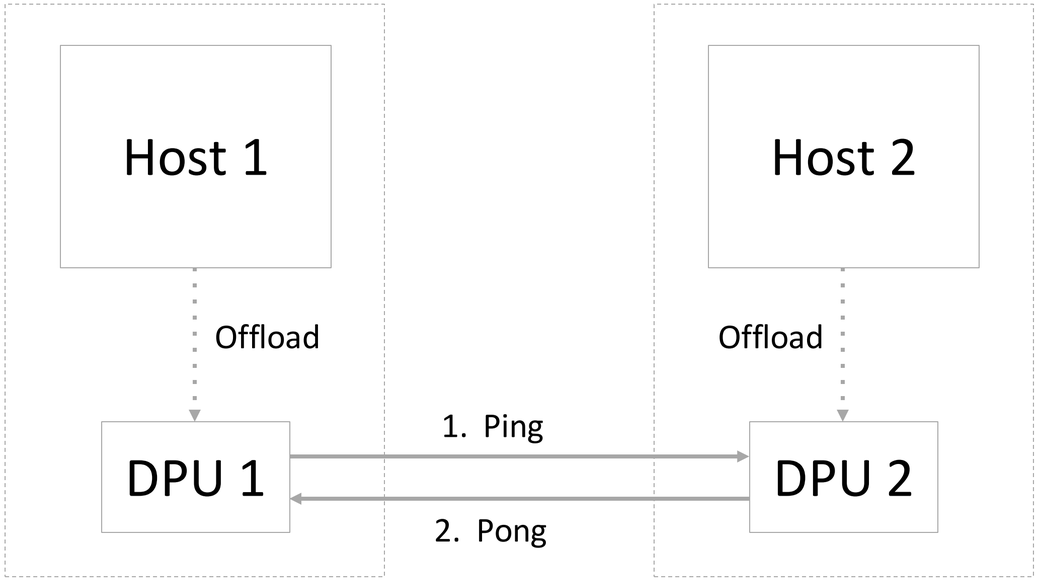
```
uthmanhere@thorbf3a013:~
```

Shared Libraries

```
$ #Compile shared object
$ LLVM/bin/clang log.c -shared -o liblog_x86.so

$ #Cross-compile shared object
$ LLVM/bin/clang log.c -shared -target aarch64-unknown-linux -o liblog_aarch64.so

$ #Link the objects and generate the fat binary
$ LLVM/bin/clang -fopenmp -fopenmp-targets=aarch64-unknown-linux  code.c -o code -L.
-llog_x86 -Wl,--device-linker=-L.,--device-linker=-llog_aarch64
```

Asynchronous TCP I/O in DPU using OpenMP (1/2)

```
#pragma omp target nowait
{
    // initialize tcp sockets, listen, bind, connect, and accept.
    fd = tcp_init(mode);

    // compute and communicate asynchronously on DPU
    ping_pong(fd, mode);

    close(fd);
}
#pragma omp taskwait
```



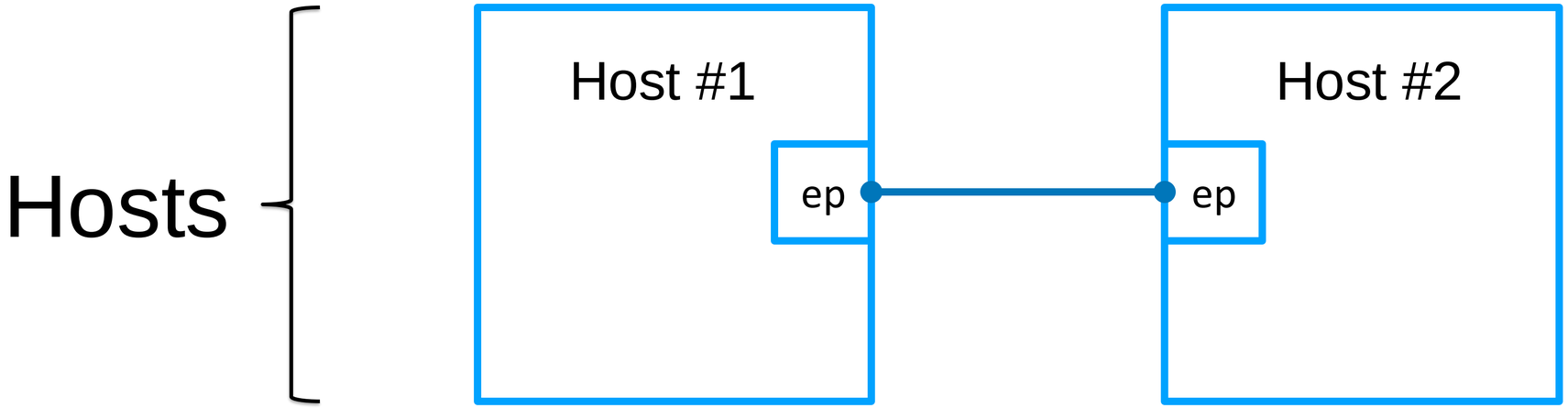
Asynchronous TCP I/O in DPU using OpenMP (2/2)

```
void ping_pong(int fd, char mode) {
        int i, m = 0;
        for (i = 0; i < _ITERATIONS_; ++i) {
                if (mode == 'c') {
                        printf("to   server > %02d\n", m);
                        send(fd, &m, sizeof(m), 0);
                        recv(fd, &m, sizeof(m), 0);
                        ++m;
                } else {
                        recv(fd, &m, sizeof(m), 0);
                        ++m;
                        printf("to   client > %02d\n", m);
                        send(fd, &m, sizeof(m), 0);
                }
        }
}
```

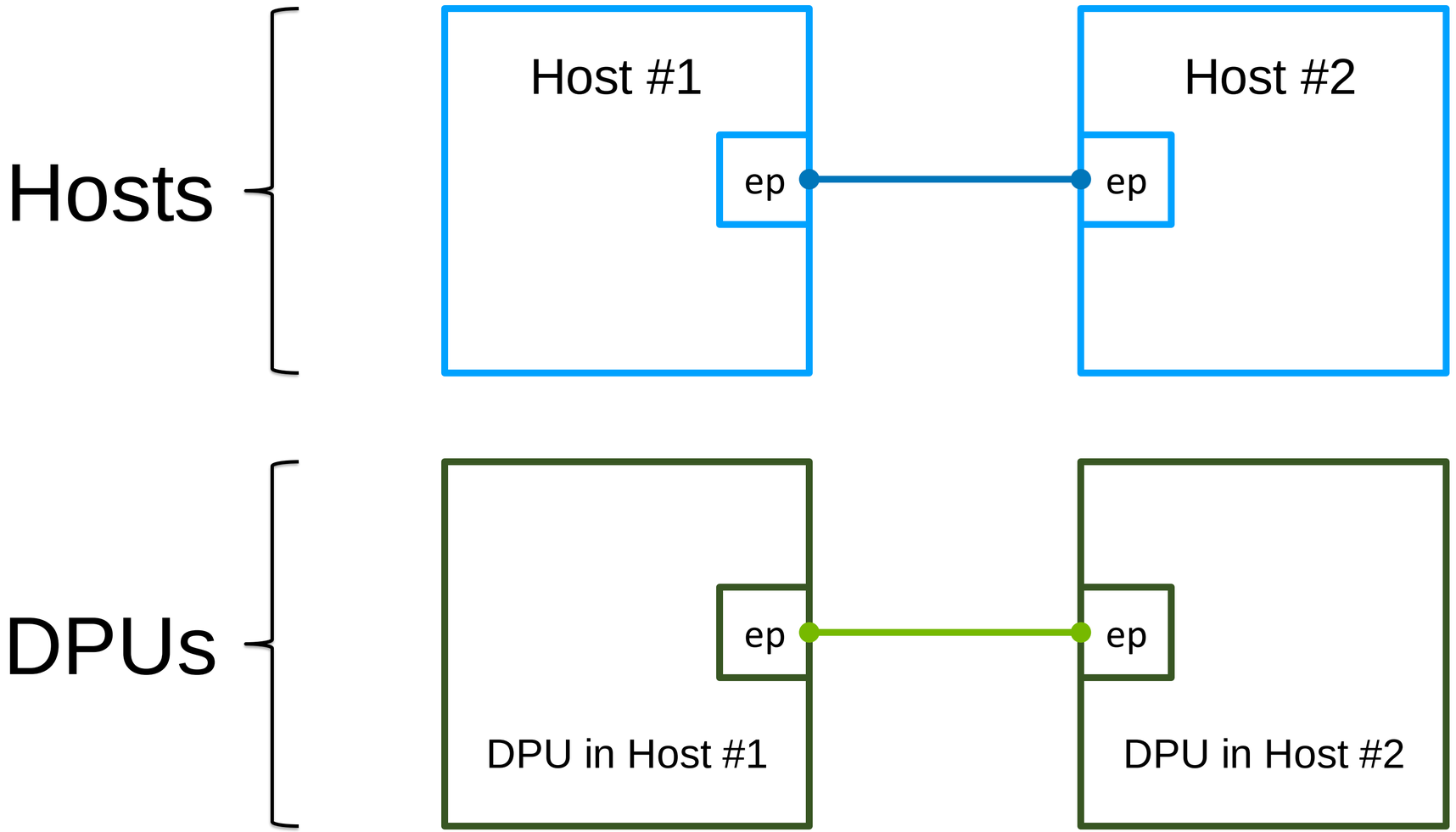

uthmanhere@thor013:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thor013:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thorbf3a013:~$ ./doca_openmp_service
connection established.
1
Server (Pong)
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thorbf3a014:~$ ./doca_openmp_service
connection established.
Client (Ping)
2

ODOS MPI

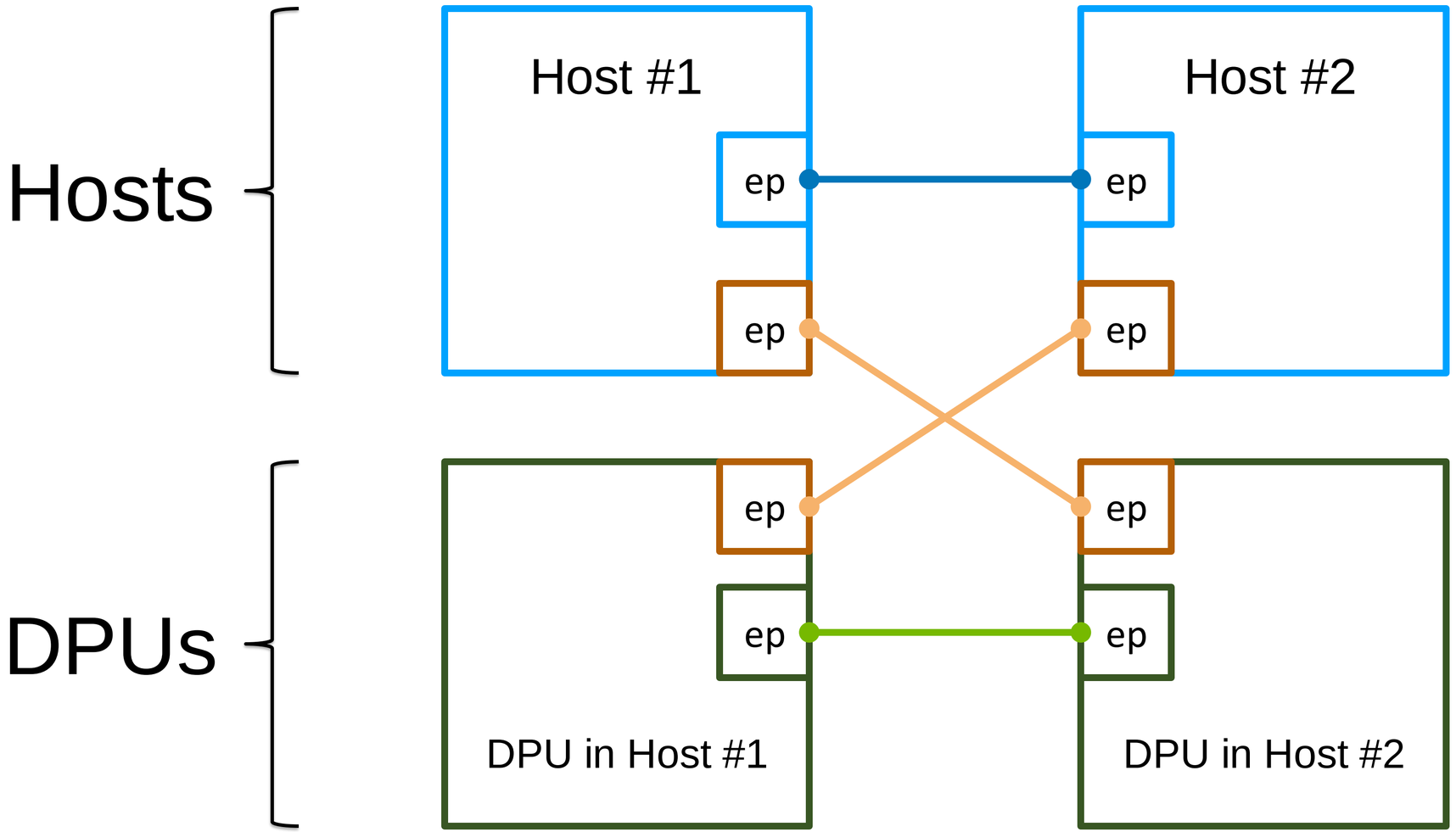
End-Points Links (1/3)



End-Points Links (2/3)



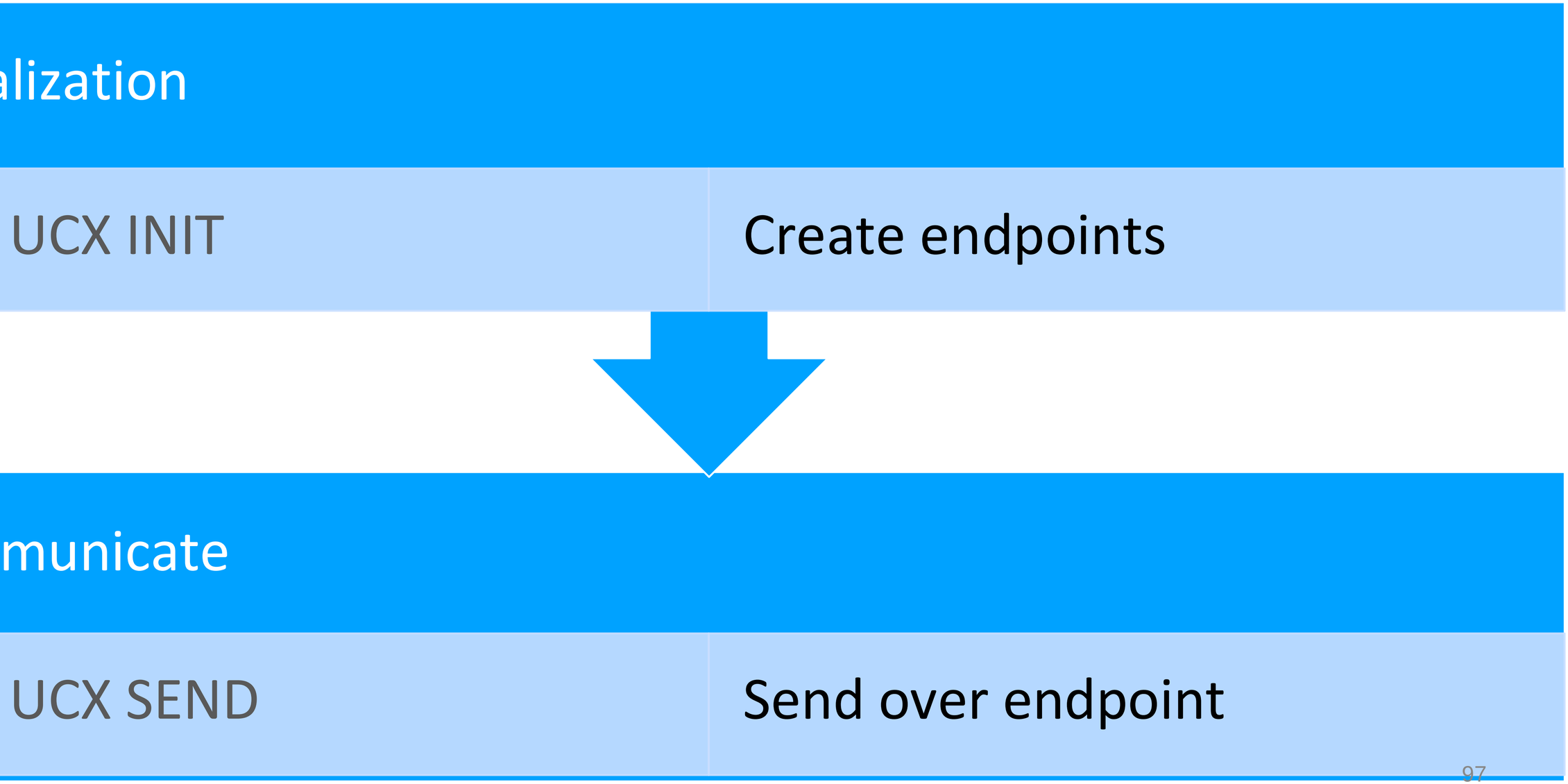
End-Points Links (3/3)



DPU as a local co-processor

- DPUs don't have their own MPI rank number ☺

Open MPI for ODOS



DEMO #2

- MPI P2P
- MPI RMA

- **2 Nodes**
- **1 rank per node**
- **Ranks can execute commands in the host and in the DPU**

MPI P2P DPU-to-DPU

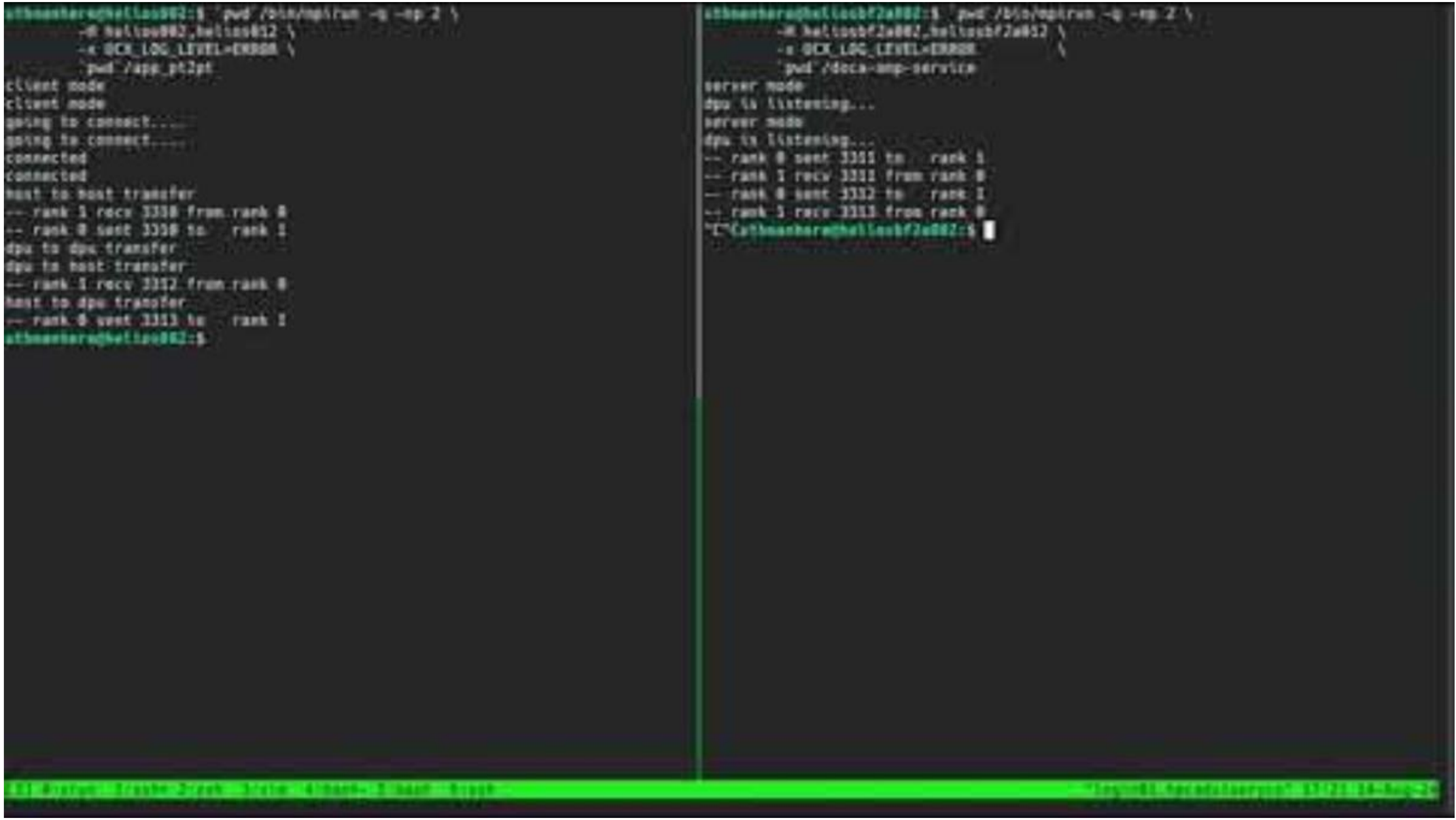
```
#pragma omp target
{
  if ( rank == 0 ) {
   MPI_Send( &buf, 1, MPI_INT, dst, 0, MPI_COMM_WORLD );
  } else {
   MPI_Recv( &recv_data, 1, MPI_INT, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
  }
}
```

MPI P2P DPU-to-Host

```
if ( rank == 0 ) {
    #pragma omp target
    {
        MPI_Info info;
        MPI_Info_create( &info );
        MPI_Info_set( info, "REMOTE_DEST_HETERO", "1" );
        MPI_Comm_set_info( MPI_COMM_WORLD, info );
        MPI_Send( &buf, 1, MPI_INT, dst, 0, MPI_COMM_WORLD );
        MPI_Info_delete( info, "REMOTE_DEST_HETERO" );
        MPI_Info_free( &info );
    }
} else {
    MPI_Recv( &recv_data, 1, MPI_INT, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
```

MPI P2P Example

- Host to host: "3310"
- Device to device: "3311"
- Device to host: "3312"
- Host to device: "3313"



```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_pt2pt
client mode
client mode
going to connect....
going to connect....
connected
connected
host to host transfer
-- rank 1 recv 3310 from rank 0
-- rank 0 sent 3310 to   rank 1
dpu to dpu transfer
dpu to host transfer
-- rank 1 recv 3312 from rank 0
host to dpu transfer
-- rank 0 sent 3313 to   rank 1
uthmanhere@helios002:$
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
-- rank 0 sent 3311 to   rank 1
-- rank 1 recv 3311 from rank 0
-- rank 0 sent 3312 to   rank 1
-- rank 1 recv 3313 from rank 0
^C^Cuthmanhere@heliosbf2a002:$
```

MPI Collectives across DPUs

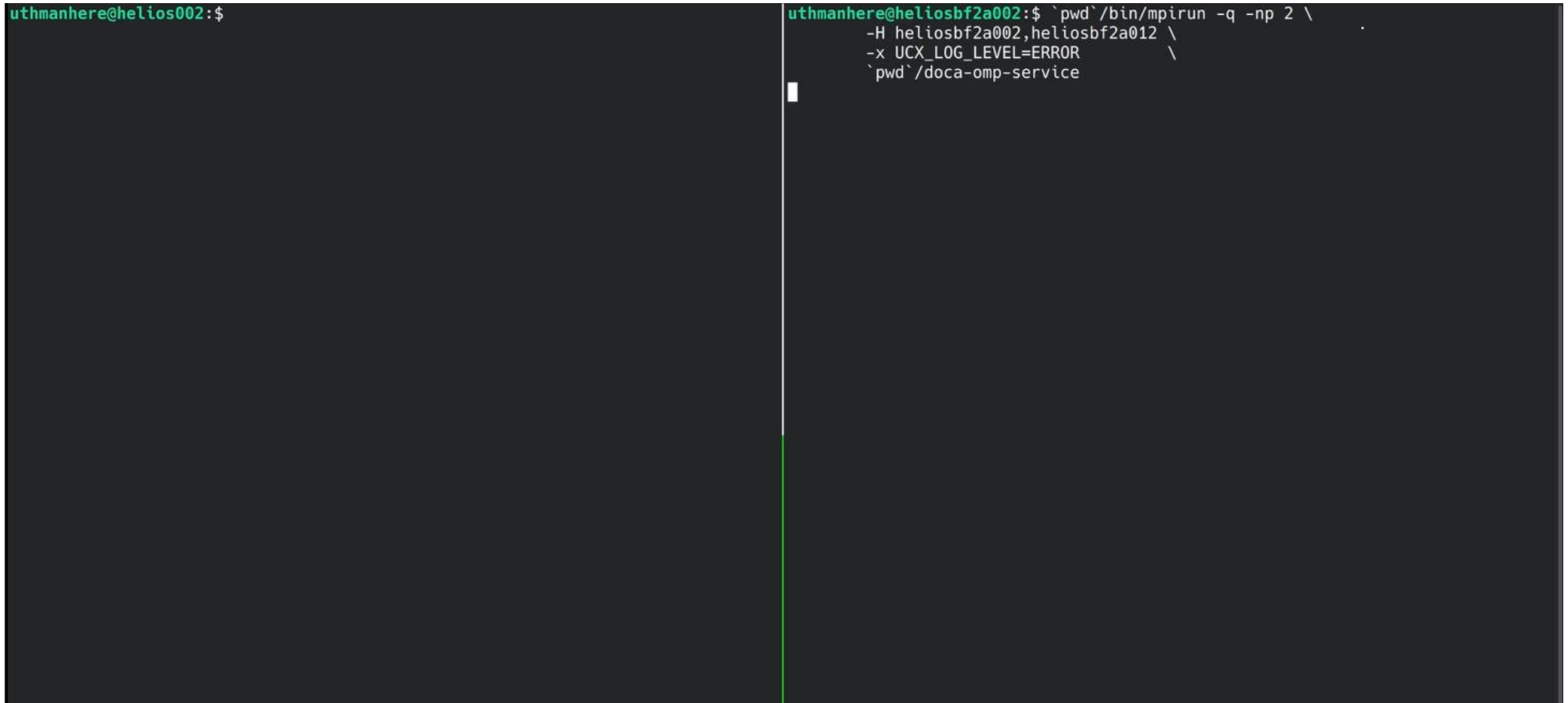
```
#pragma omp target
{
    MPI_Alltoall(send_buffer, 1, MPI_INT, recv_buffer, 1, MPI_INT, MPI_COMM_WORLD);
}
```

MPI RMA Host-to-DPU

```
#pragma omp target nowait
{
  MPI_Win_create(&window_data_dpu, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win);
  MPI_Win_fence(0, win);
  MPI_Win_free(&win);
}

MPI_Info info;
MPI_Info_create(&info);
MPI_Win_create(&window_data, sizeof(int), sizeof(int), info, MPI_COMM_WORLD, &win);

if (rank == src) {
    MPI_Info_set(info, "REMOTE_DEST_HETERO", "1");
    MPI_Win_set_info(win, info);
    MPI_Put(&data, 1, MPI_INT, dst, 0, 1, MPI_INT, win);
    MPI_Info_delete(info, "REMOTE_DEST_HETERO");
 }

 MPI_Win_fence(0, win);
 MPI_Win_free(&win);
 MPI_Info_free(&info);
```

Host-DPU Simultaneous RMA Communication (1/4)

- Simultaneous one-sided cross-communication:
  - Rank#0_Host to Rank#1_DPU
  - Rank#0_DPU to Rank#1_Host



Host-DPU Simultaneous RMA Communication (2/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
```

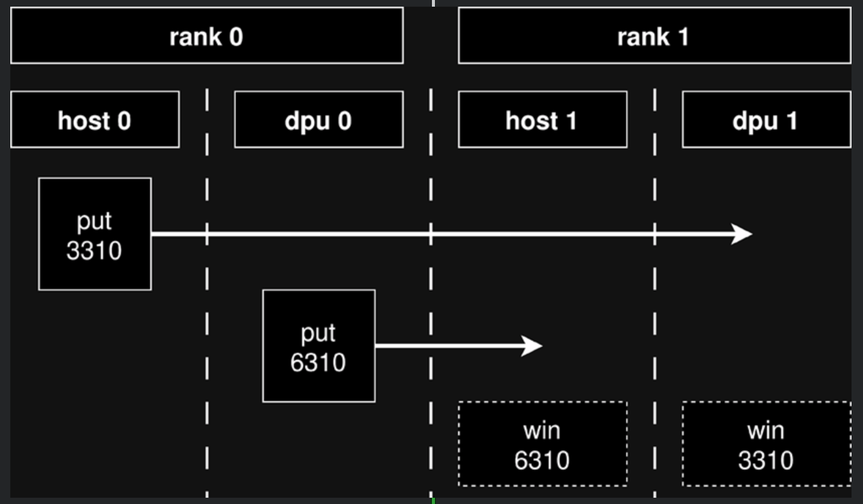
```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
```

```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim  6:bash                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Host-DPU Simultaneous RMA Communication (3/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
>>> process 1 got 6310 from rank 0 by put
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
>>> process 1 got 3310 from rank 0 by put
```

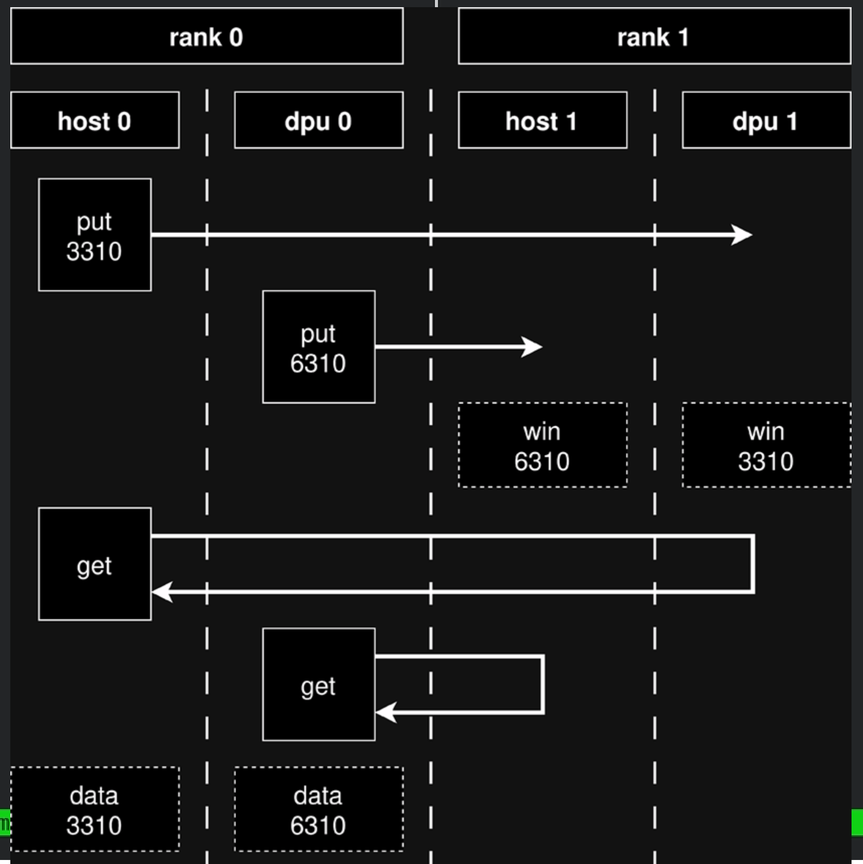



```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim  6:bash                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Host-DPU Simultaneous RMA Communication (4/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
>>> process 1 got 6310 from rank 0 by put
>>> process 0 got 3310 from rank 1 by get
uthmanhere@helios002:$
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
>>> process 1 got 3310 from rank 0 by put
>>> process 0 got 6310 from rank 1 by get
```

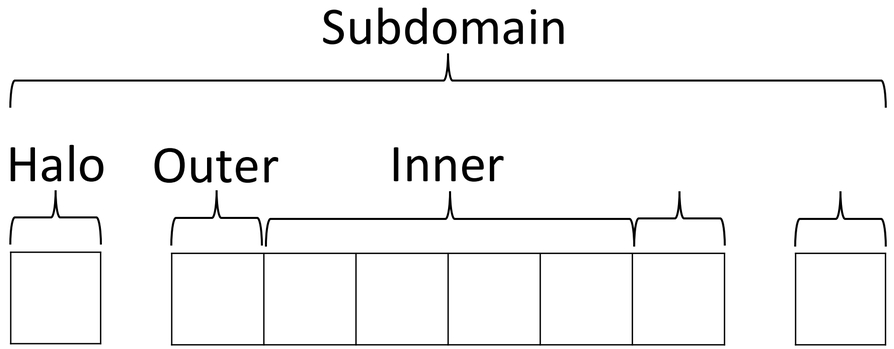


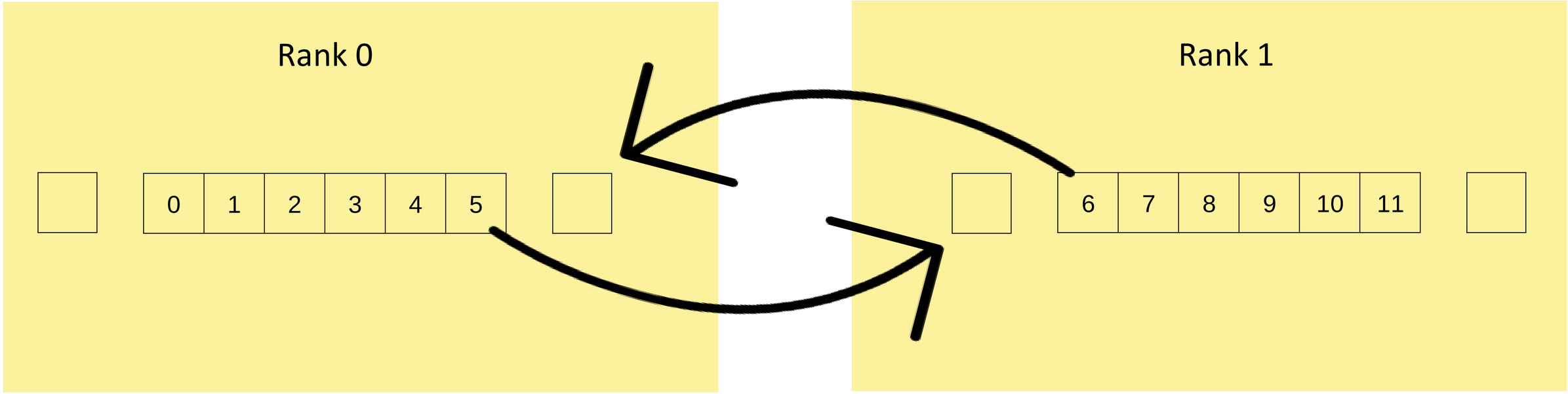

```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Example: Halo Exchange

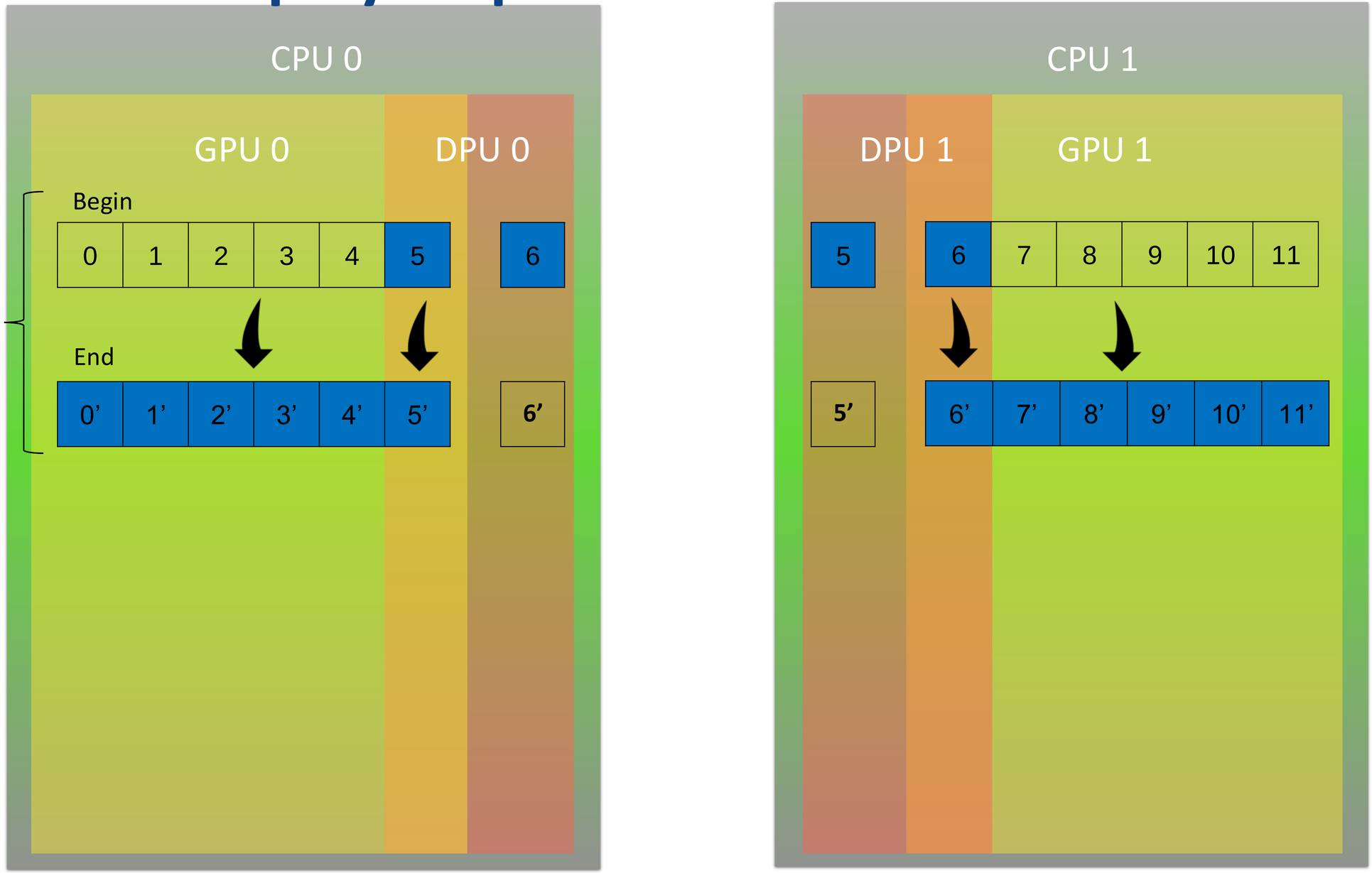
Halo Exchange Offloading

SIZE = 8
THICKNESS = 1

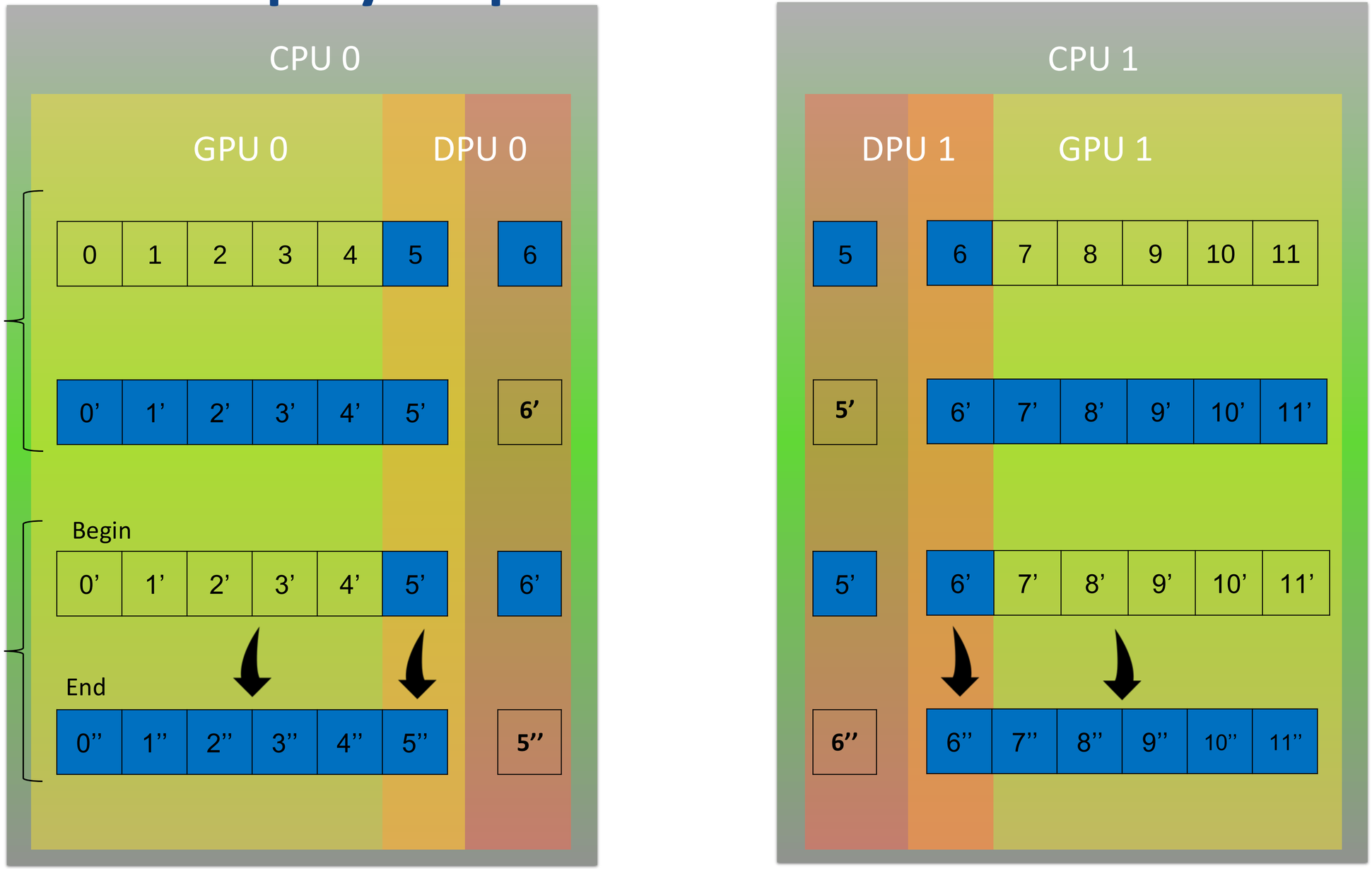




Execution step by step



Execution step by step



```
int curr_subdomain[SIZE + THICKNESS * 2];
int next_subdomain[SIZE + THICKNESS * 2];
#pragma omp target data map(to: curr_subdomain[THICKNESS:SIZE-THICKNESS])
                        map(from: next_subdomain[THICKNESS:SIZE-THICKNESS])
                        device (GPU_ID) nowait
{
    #pragma omp target teams num_teams(2)
    {
        #pragma omp parallel
        {
            next_subdomain[THICKNESS] = compute_inner(curr_subdomain[THICKNESS:SIZE-THICKNESS])
        }
    }
}
#pragma omp target data map(to: curr_subdomain[0:THICKNESS*2], curr_subdomain[SIZE-THICKNESS*2:SIZE]
                        map(from: next_subdomain[0:THICKNESS*2], next_subdomain[SIZE-THICKNESS*2:SIZE])
                        device (DPU_ID) nowait
{
    #pragma omp parallel
    #pragma omp single
    {
        #pragma omp task
        {
            next_subdomain[THICKNESS] = compute_outer(curr_subdomain[0:THICKNESS*2], left);
        }
        #pragma omp task
        {
            next_subdomain[SIZE-THICKNESS*2] = compute_outer(curr_subdomain[SIZE-THICKNESS*2:SIZE], right);
        }
    }
}
#pragma omp barrier
curr_subdomain = copy_domain(next_subdomain);
```

GPU

DPU

Offloaded Function to the DPU

```
int* compute_outer(int* data, int rank_id)
{
    int halo[THICKNESS], outer_domain[THICKNESS], tag_id = 0;
    MPI_Request request;
    MPI_Status status;

    MPI_Irecv(halo, THICKNESS, MPI_INT, rank_id, tag_id, MPI_COMM_WORLD, &request);
    outer_domain = compute(data);
    MPI_Send(outer_domain, THICKNESS, MPI_INT, rank_id, tag_id, MPI_COMM_WORLD);
    MPI_Wait(&request, &status);
    return outer_domain;
}
```

Performance Evaluation

Testbed

1. Jupiter
   - Intel Xeon 10-core E5-2680, 64GB DD3
   - NVIDIA BlueField-2 HDR 100Gbps: 8 ARMv8 A72 cores, 16GB DDR4
2. Thor
   - 2x Intel Xeon 16-core E5-2697, 256GB DD4
   - NVIDIA BlueField-3 NDR 200Gbps: 16 ARMv8.2 A78 cores, 16GB DDR5

Software Stack

OpenMP 5.0

LLVM 14.0.6

DOCA 2.0.2

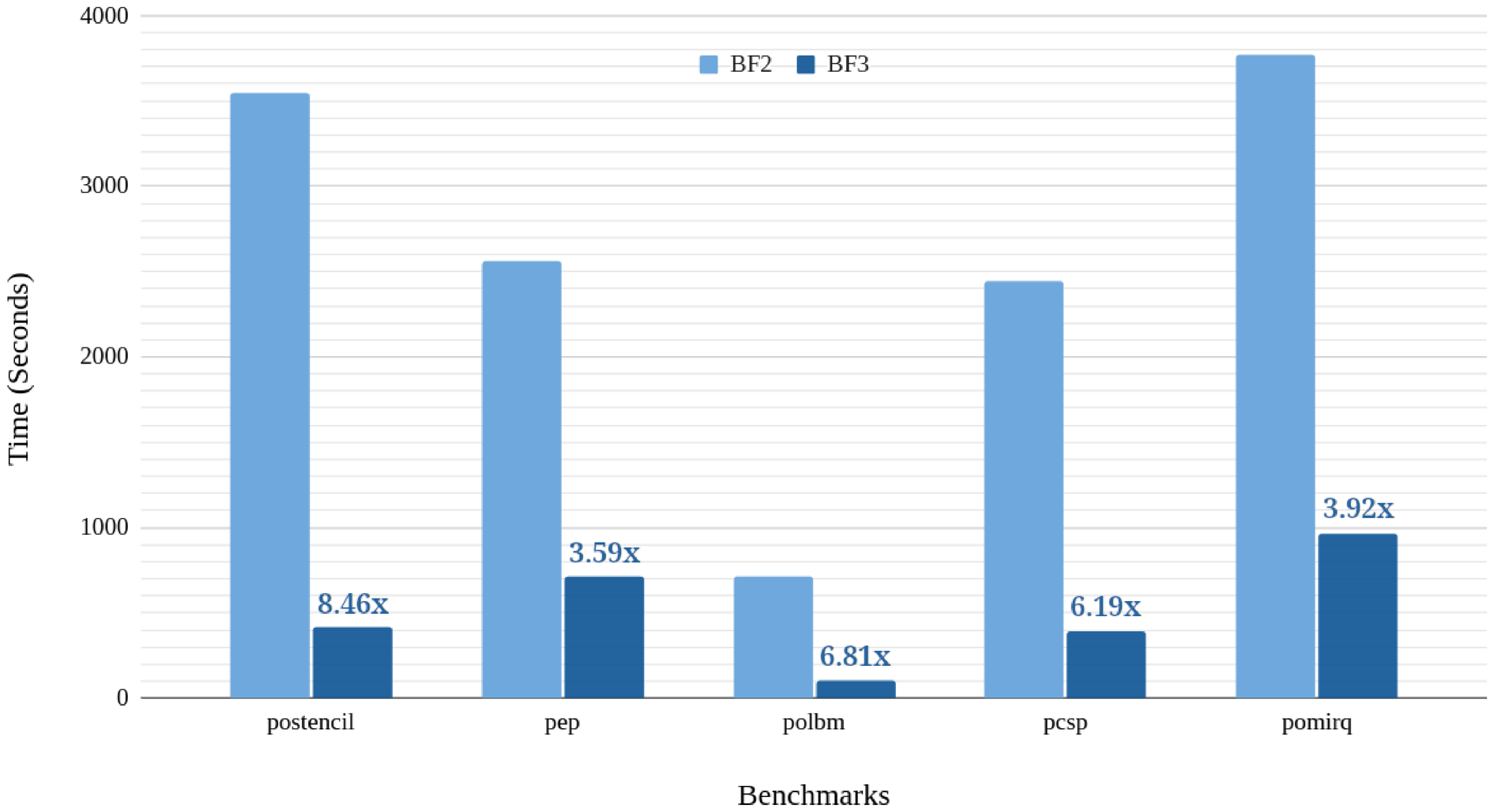
OpenMPI 4.1.6

SPEC ACCEL Benchmark Suite (OpenMP target Offloading)

- Removed OpenMP “distribute” construct for GPUs
- Added “simd” construct for enabling SIMD instructions

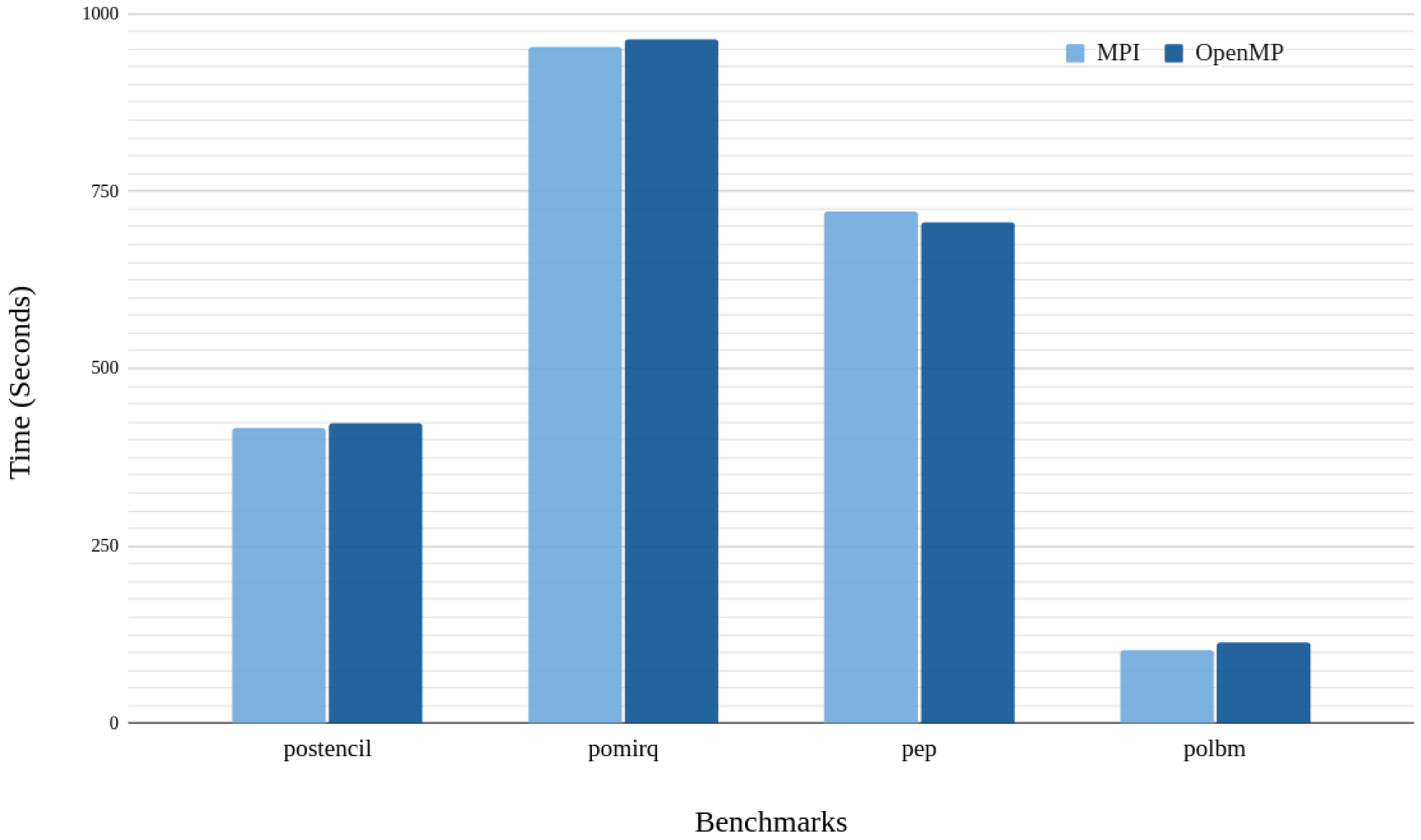
OSU Microbenchmarks (variation supporting DOCA)

BlueField 2 and BlueField 3 Performance Comparison



M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23,* https://doi.org/10.1145/3624062.3624165

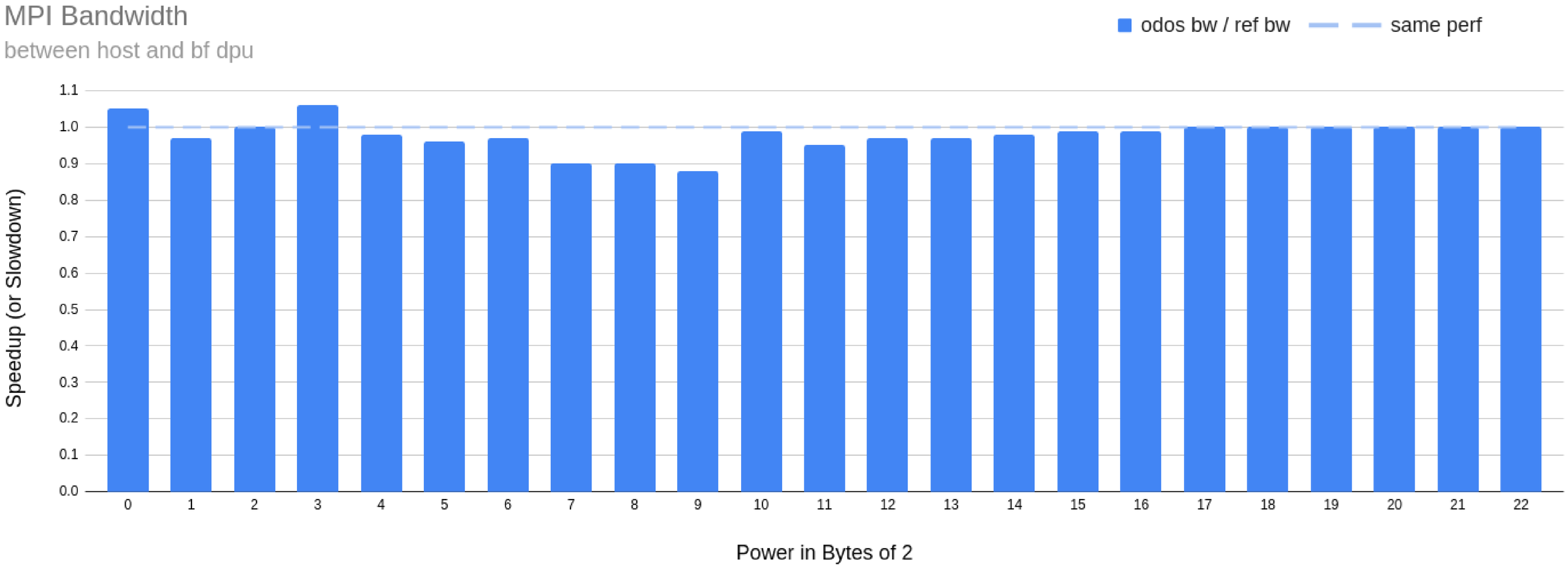
OpenMP and MPI Performance Comparison



M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23,* https://doi.org/10.1145/3624062.3624165

MPI vs ODOS MPI Offloading

Custom version of OSU to offload MPI to DPU



Latest version of ODOS with MPI offloading support unreleased – Contact us: accelcom@bsc.es

Hands On Section – Application Experiences

Users can test the following codes. ODOS, miniMD, and P3DFFT++

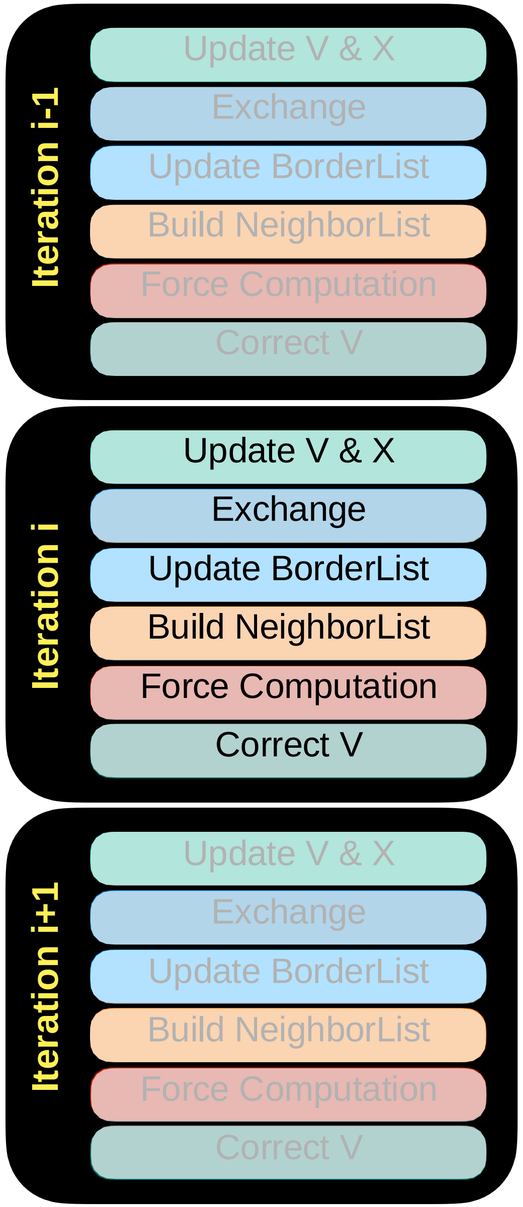
Octopus will be shown as a demo.

Please check the instructions for accessing the testbed from our public GitHub repo at

https://github.com/gt-crnch-rg/smartnic-tutorial-sc24/

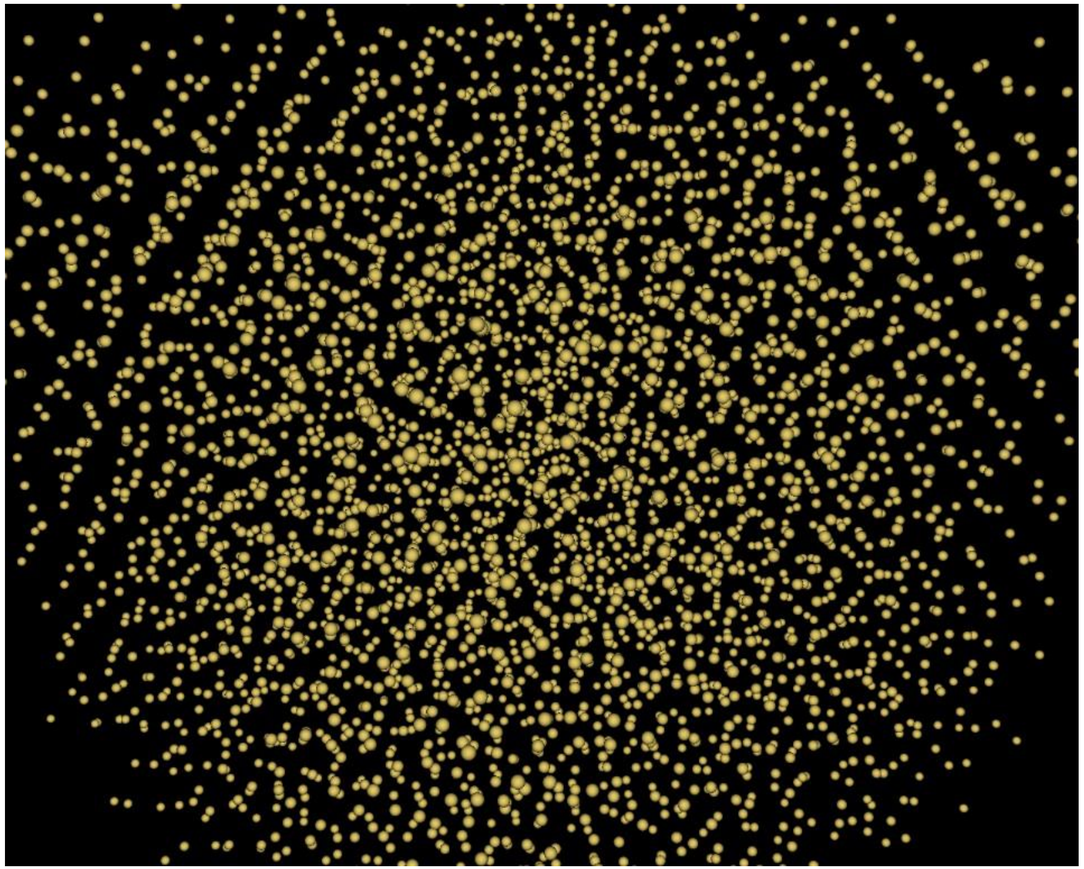
Heterogeneous Transformation with MPI: MiniMD case study

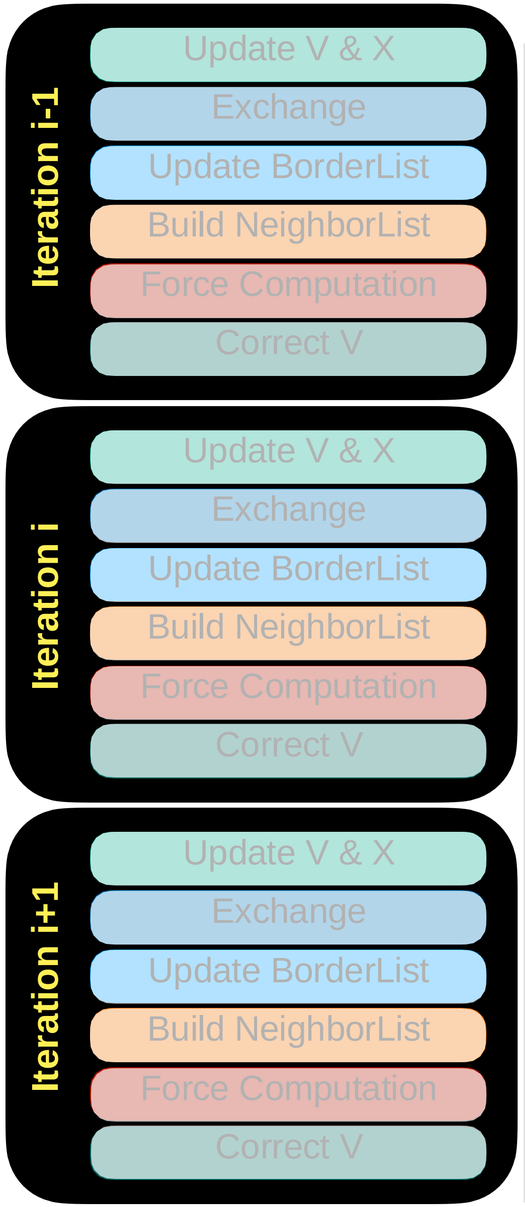
MiniMD (S. Karamati et al., “Smarter” NICs for faster molecular dynamics: a case study, IPDPS, 2022)



Baseline MiniMD

MiniMD is a molecular dynamics proxy-app. It calculates the position and velocity of a set of interacting particles in discrete time steps (iterations).

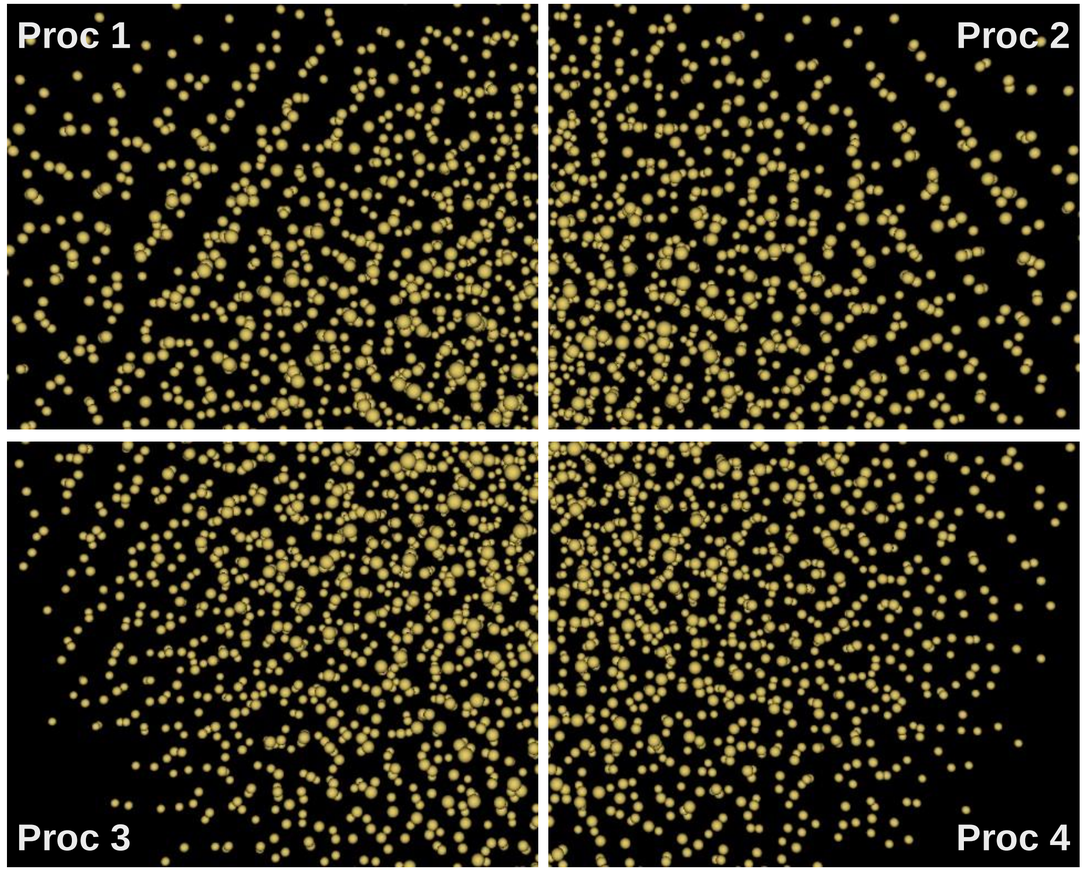


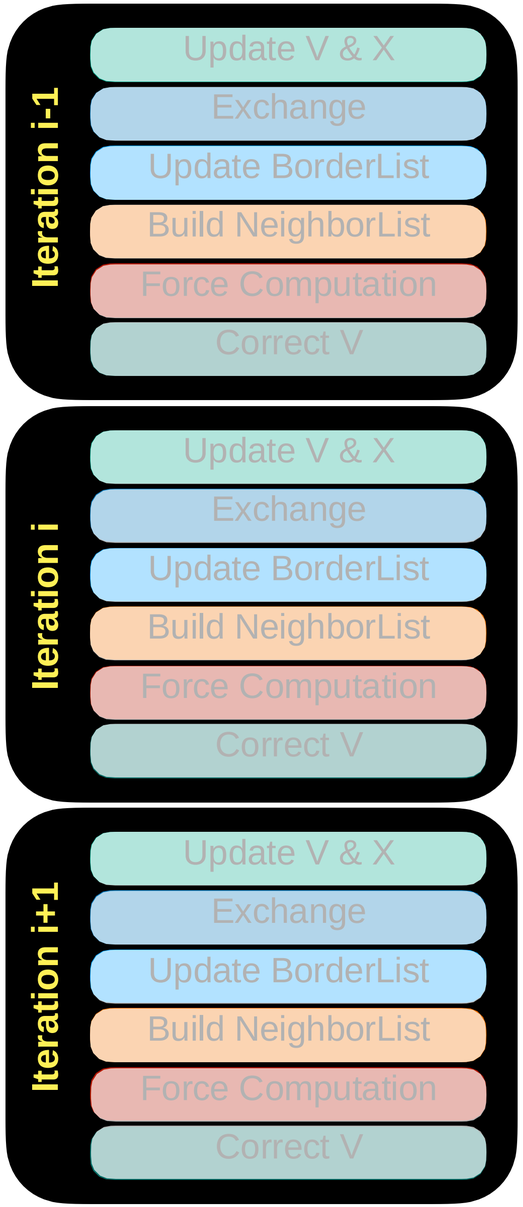


Baseline MiniMD

MiniMD is a molecular dynamics proxy-app. It calculates the position and velocity of a set of interacting particles in discrete time steps (iterations).

In the distributed-memory setting, the simulation domain is divided spatially among MPI processes.

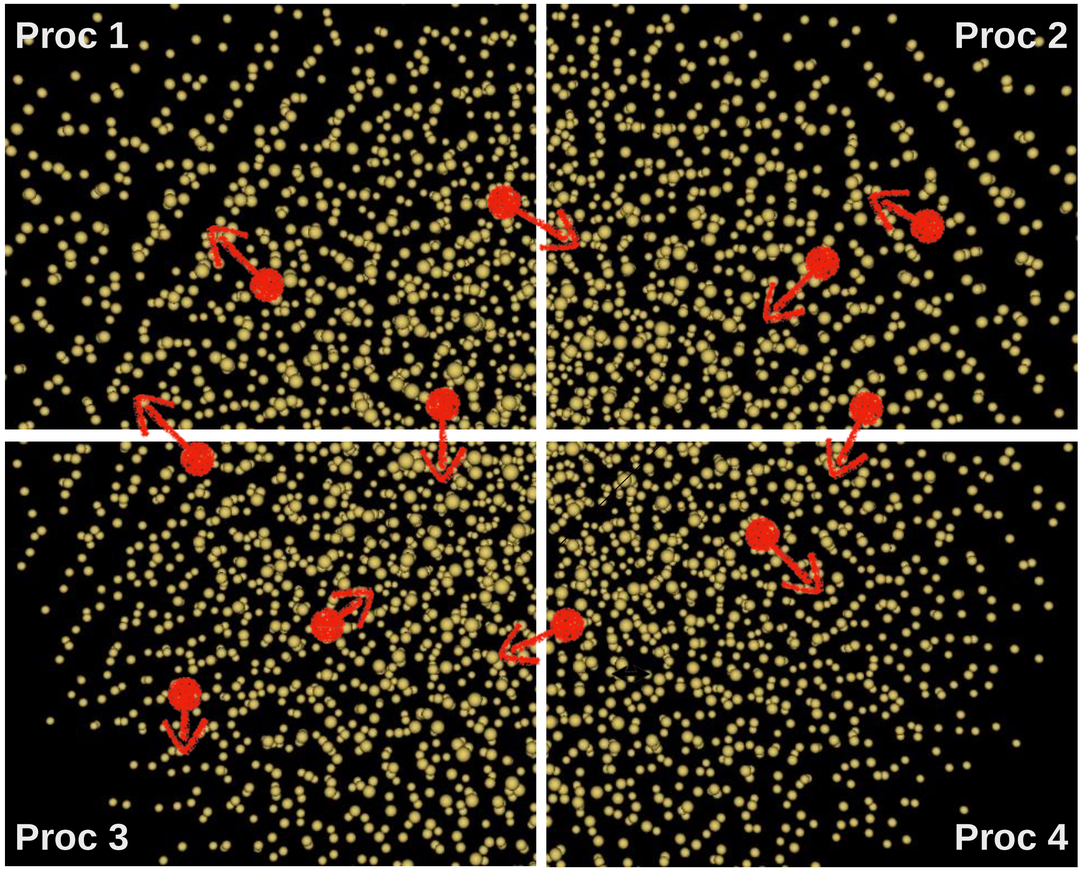


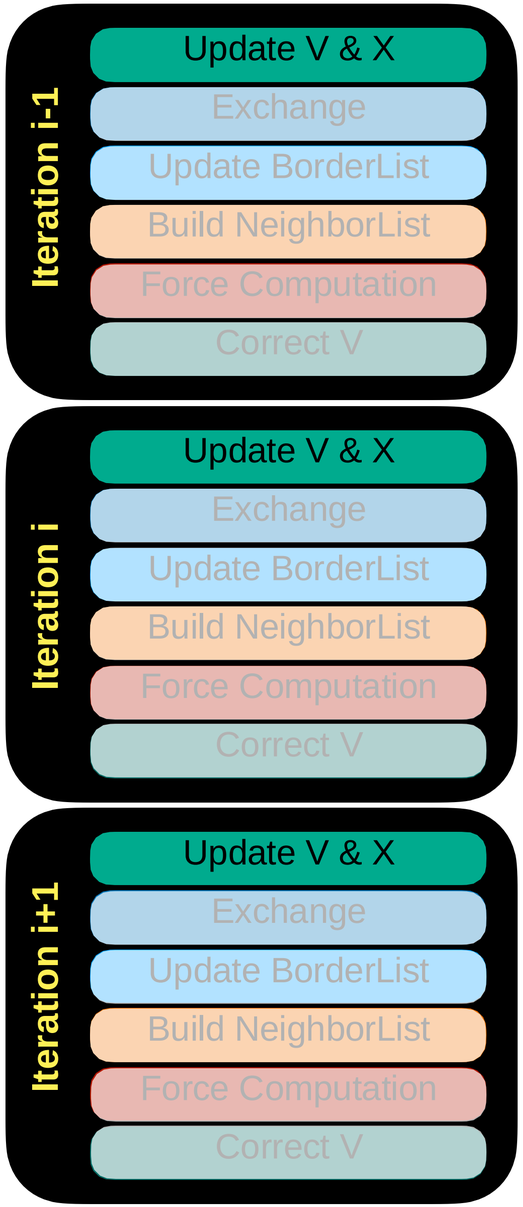


Baseline MiniMD

In each Iteration, every process:

- Update velocity and position for each particle.

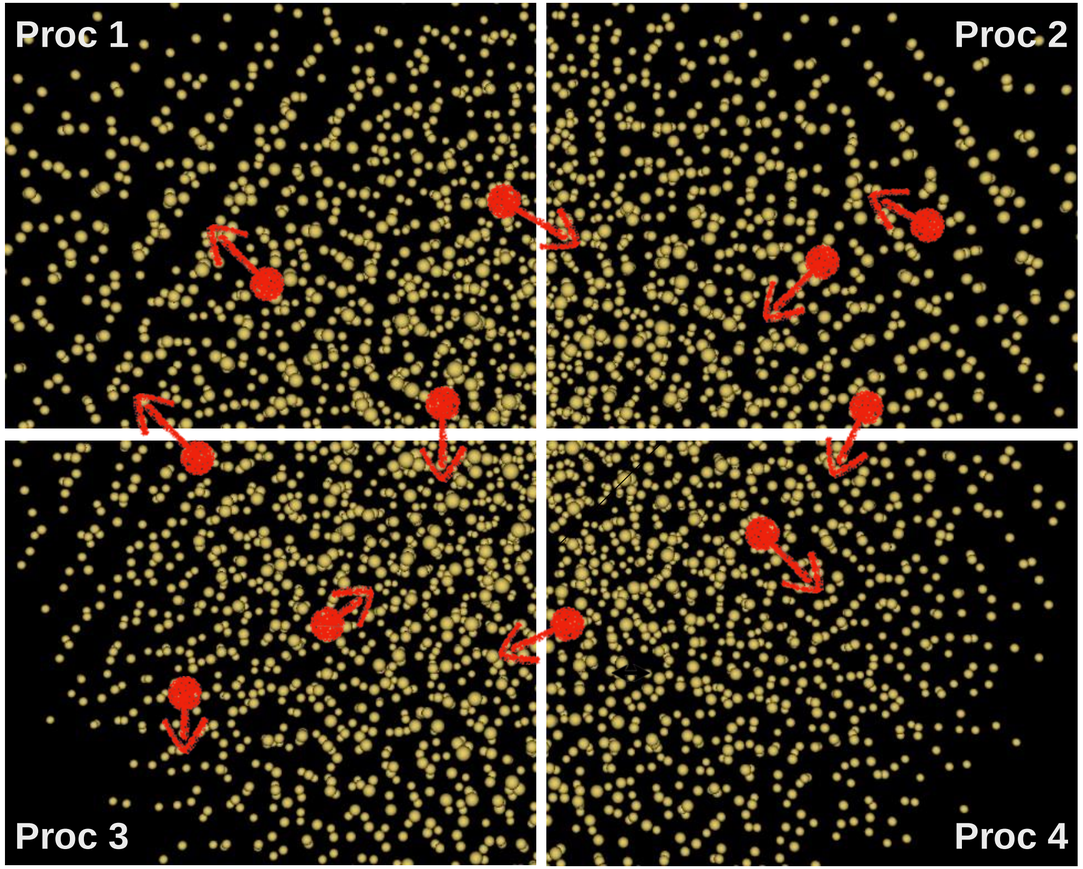


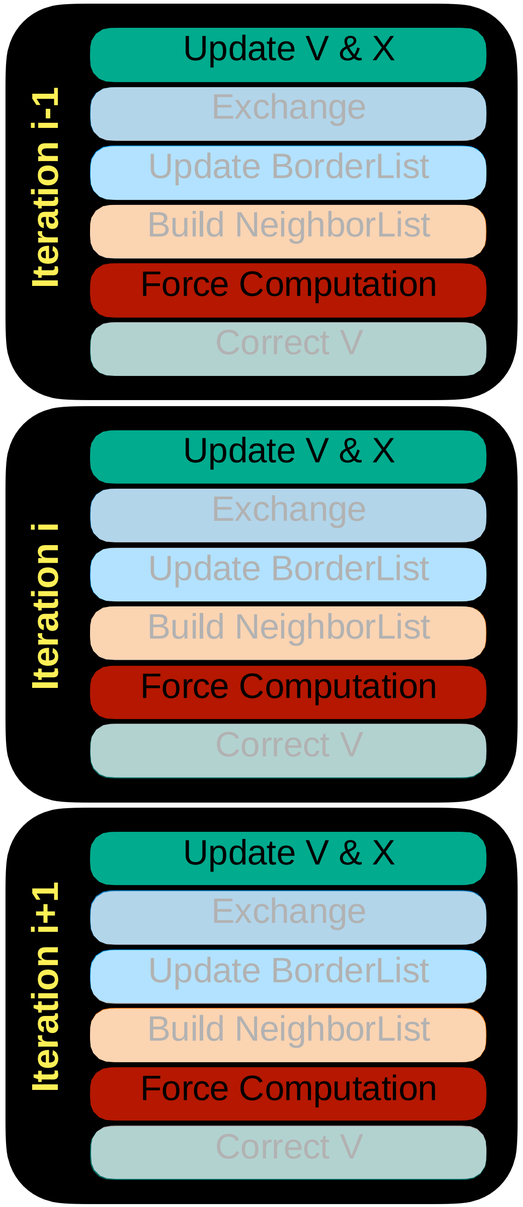


Baseline MiniMD

In each Iteration, every process:

- Update velocity and position for each particle.
- Calculate pairwise forces on each particle.

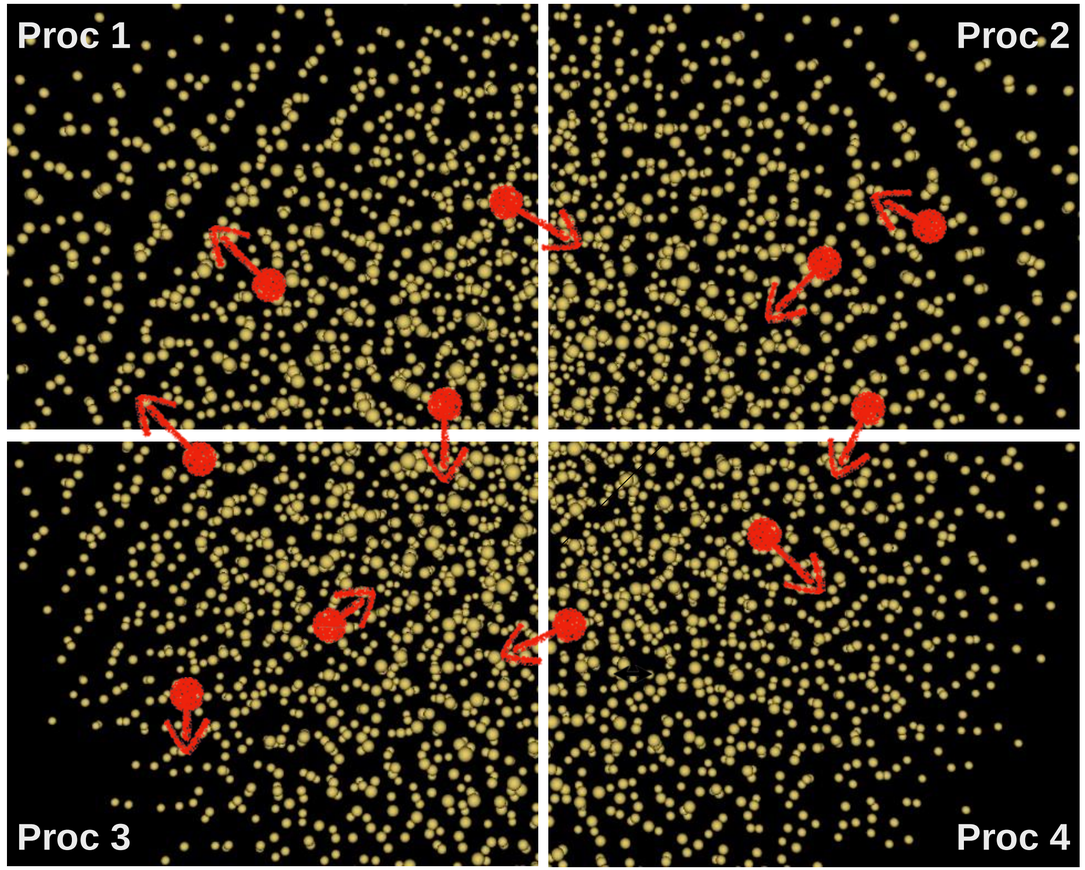


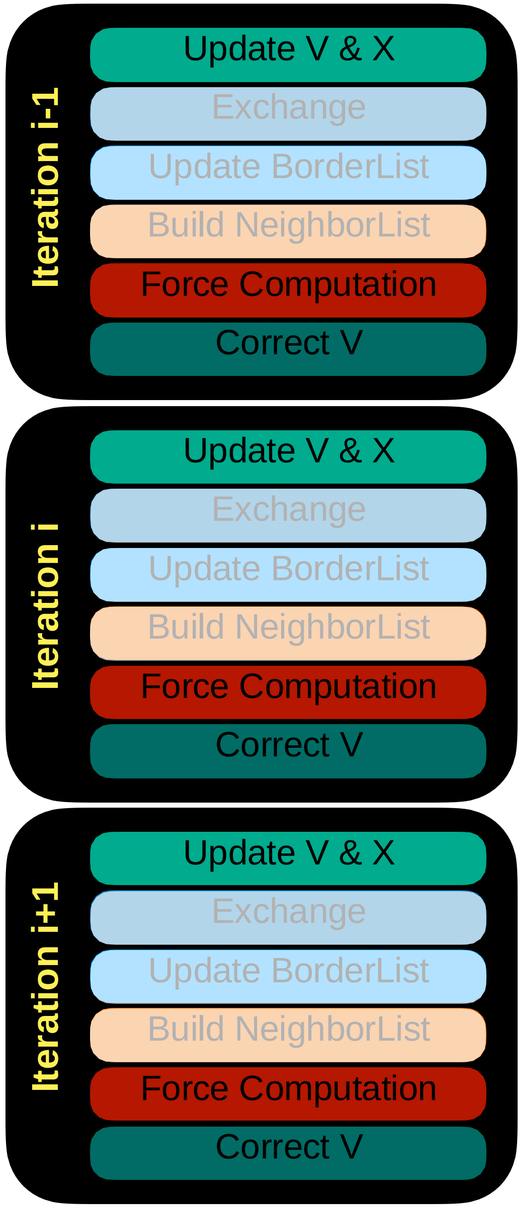


Baseline MiniMD

In each Iteration, every process:

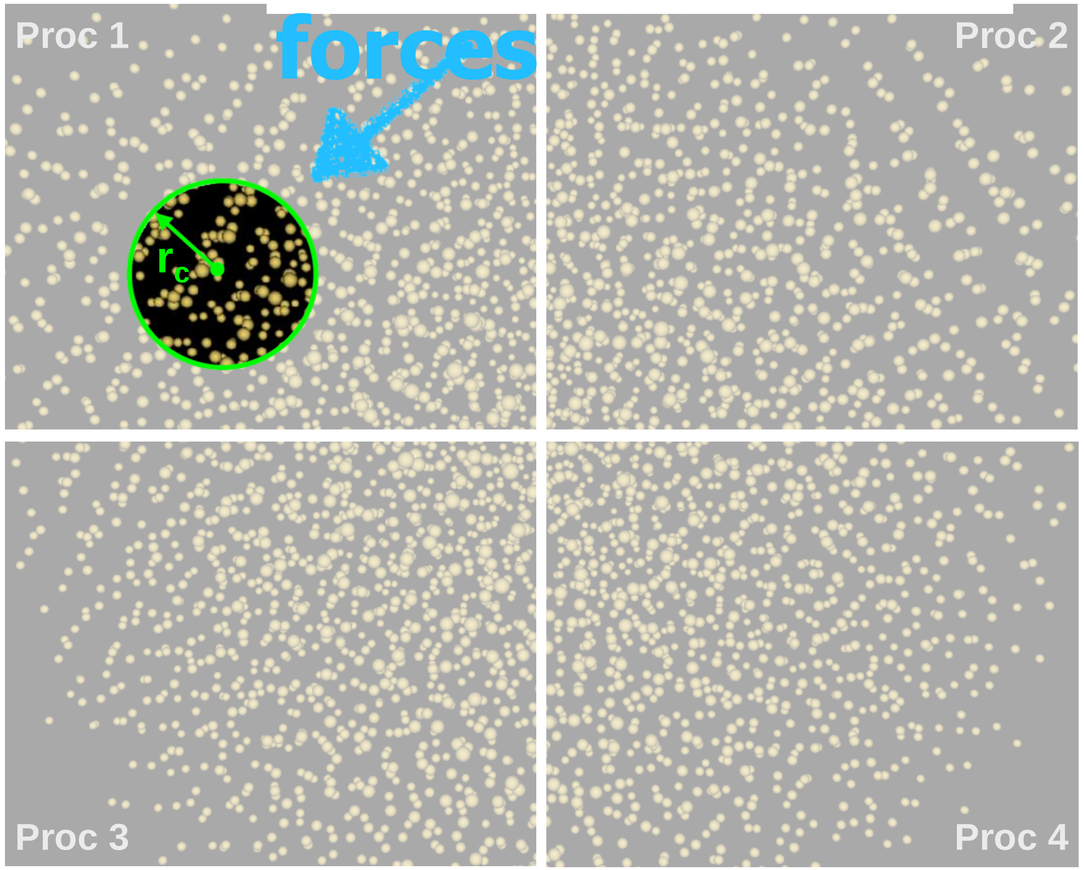
- Update velocity and position for each particle.
- Calculate pairwise forces on each particle.
- Correct velocity for each particle using the newly calculated forces.

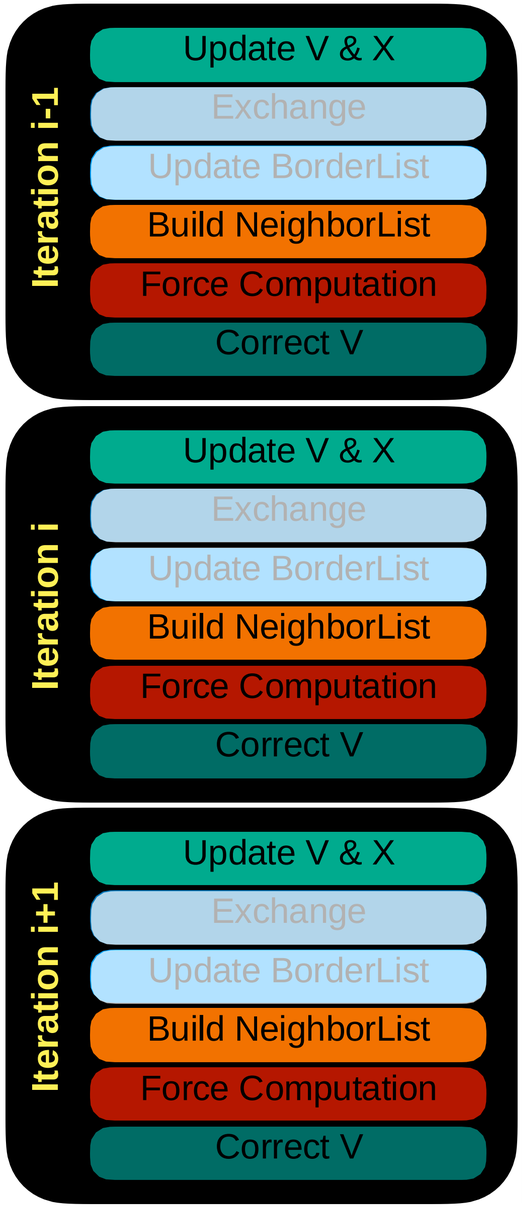




Baseline MiniMD

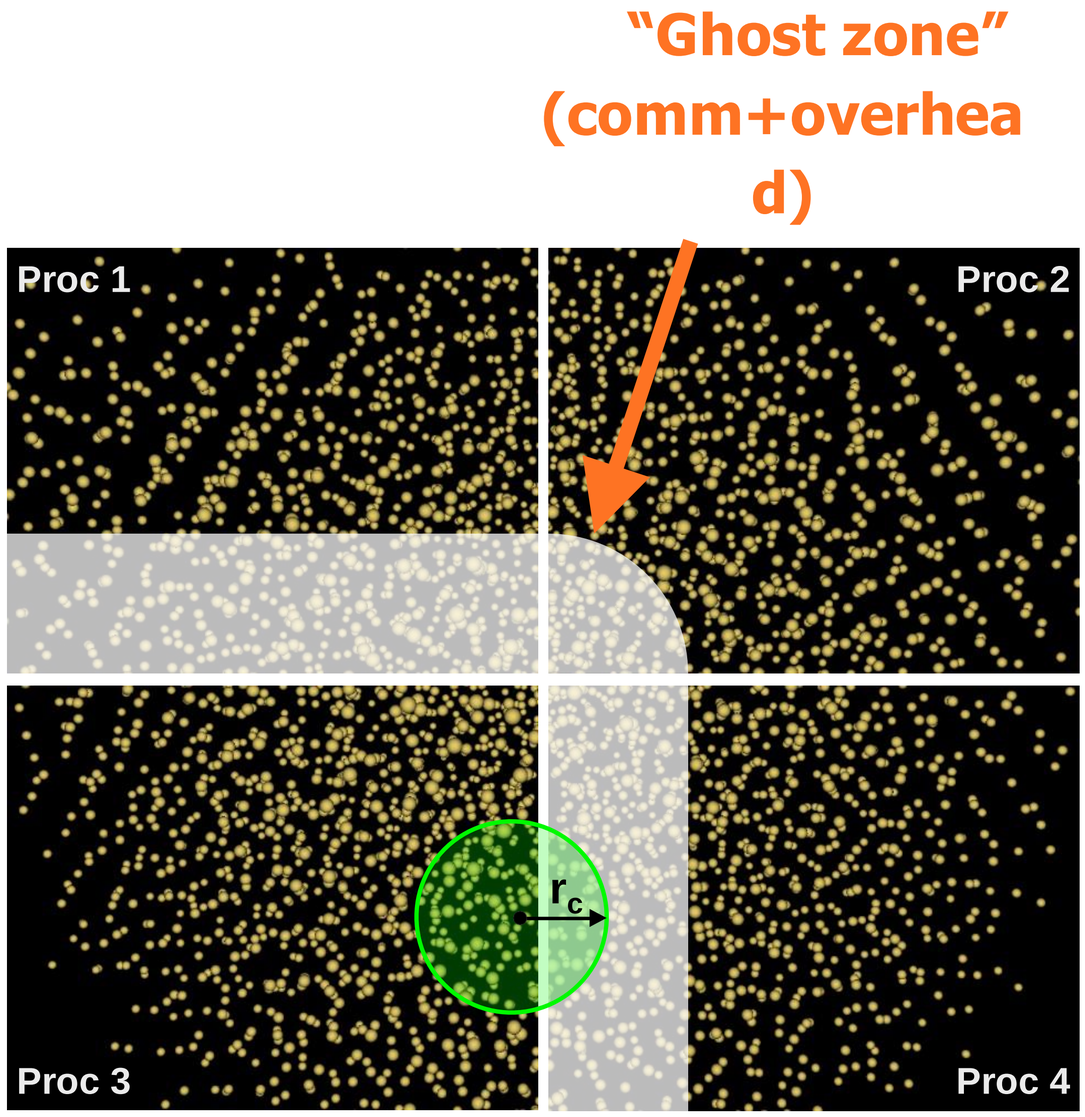
In each iteration, every particle interacts with others that lie within a some **cutoff distance** (r_c). A particle's **neighbor list** stores references to them.

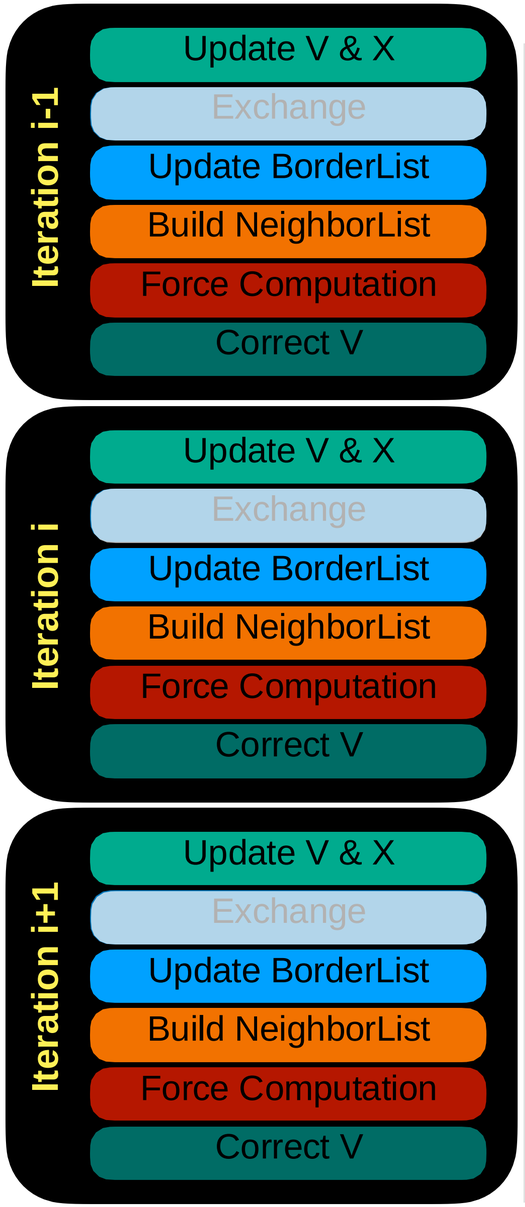




Baseline MiniMD

Each process keeps a copy of particles in the interaction region just outside the boundaries of each process.



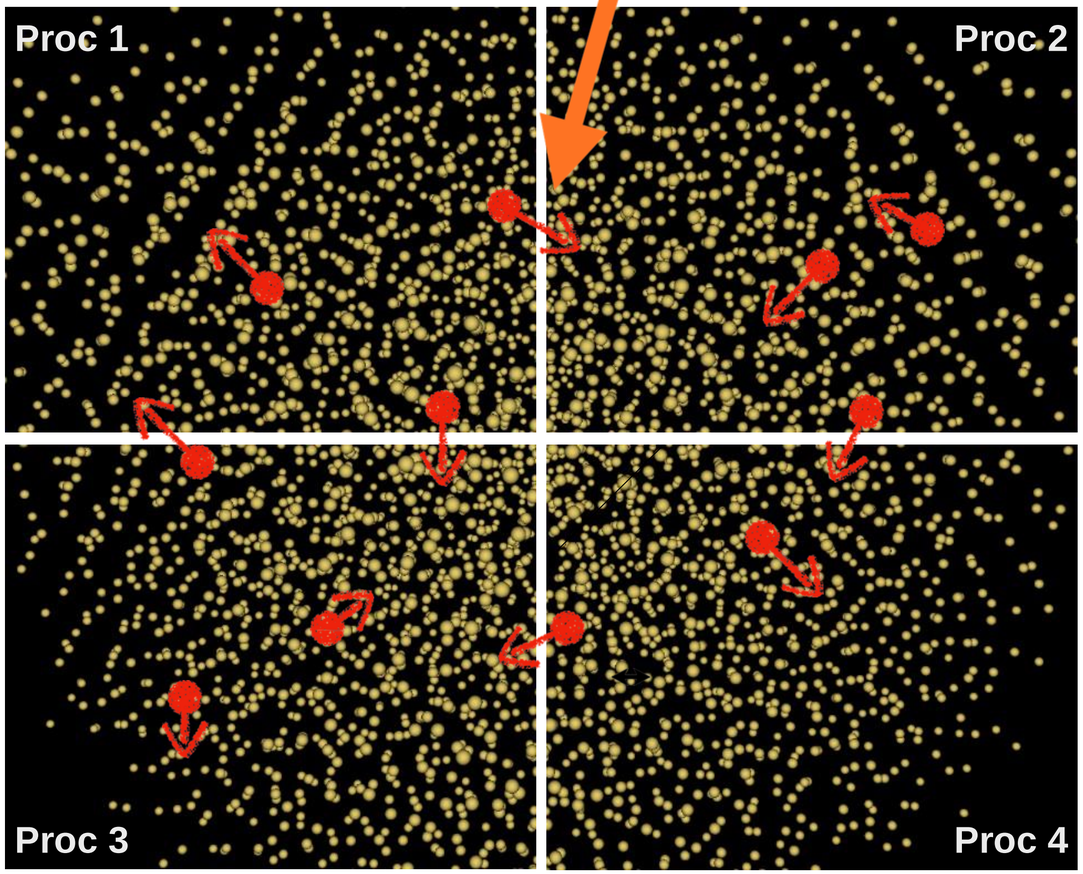


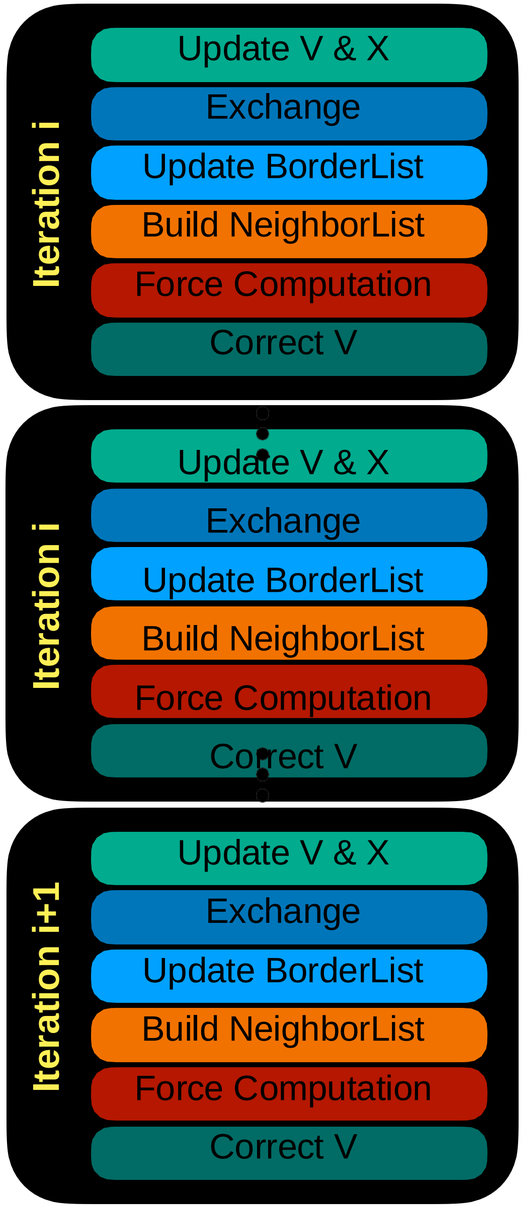
Baseline MiniMD

Particles are reassigned to new processes as they move through the spatial domain.

Neighbor list updates, boundary region exchanges, and particles reassignment to processes are triggered every so often via a user-selected parameter (e.g., every k iterations).

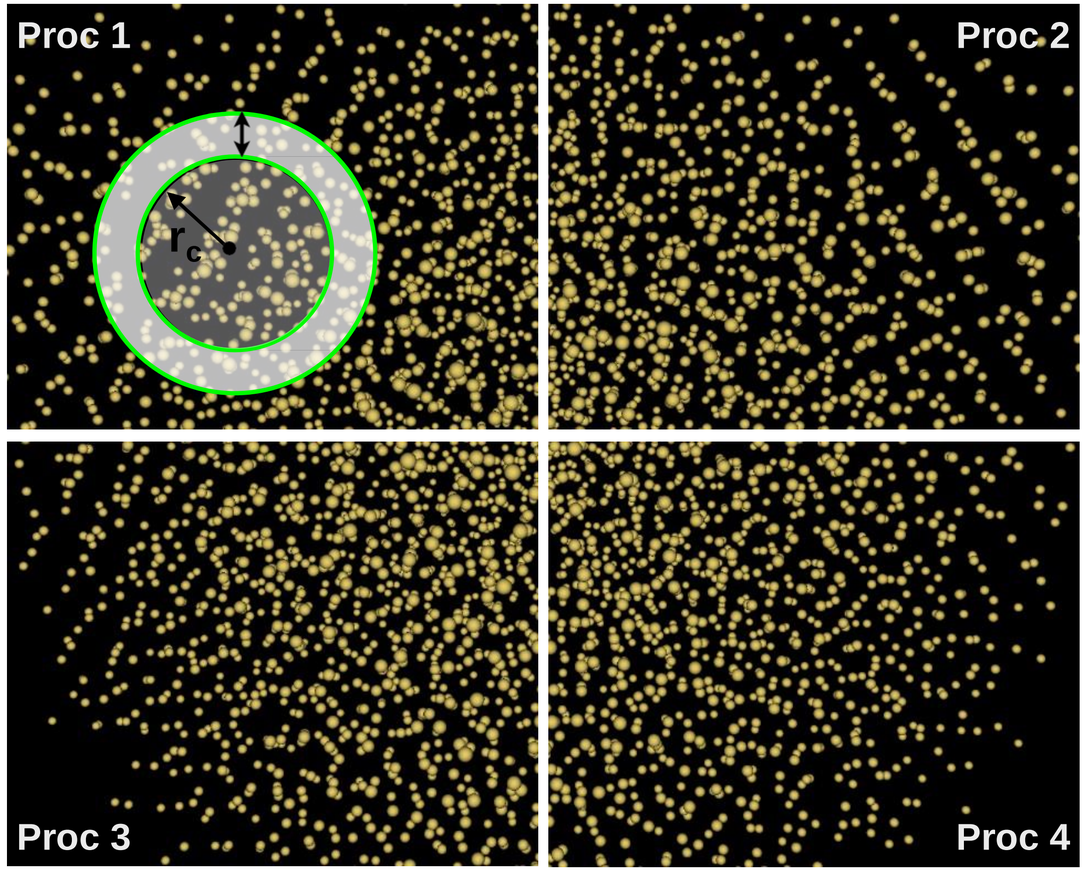
Particles move through subdomains (comm+overhead)





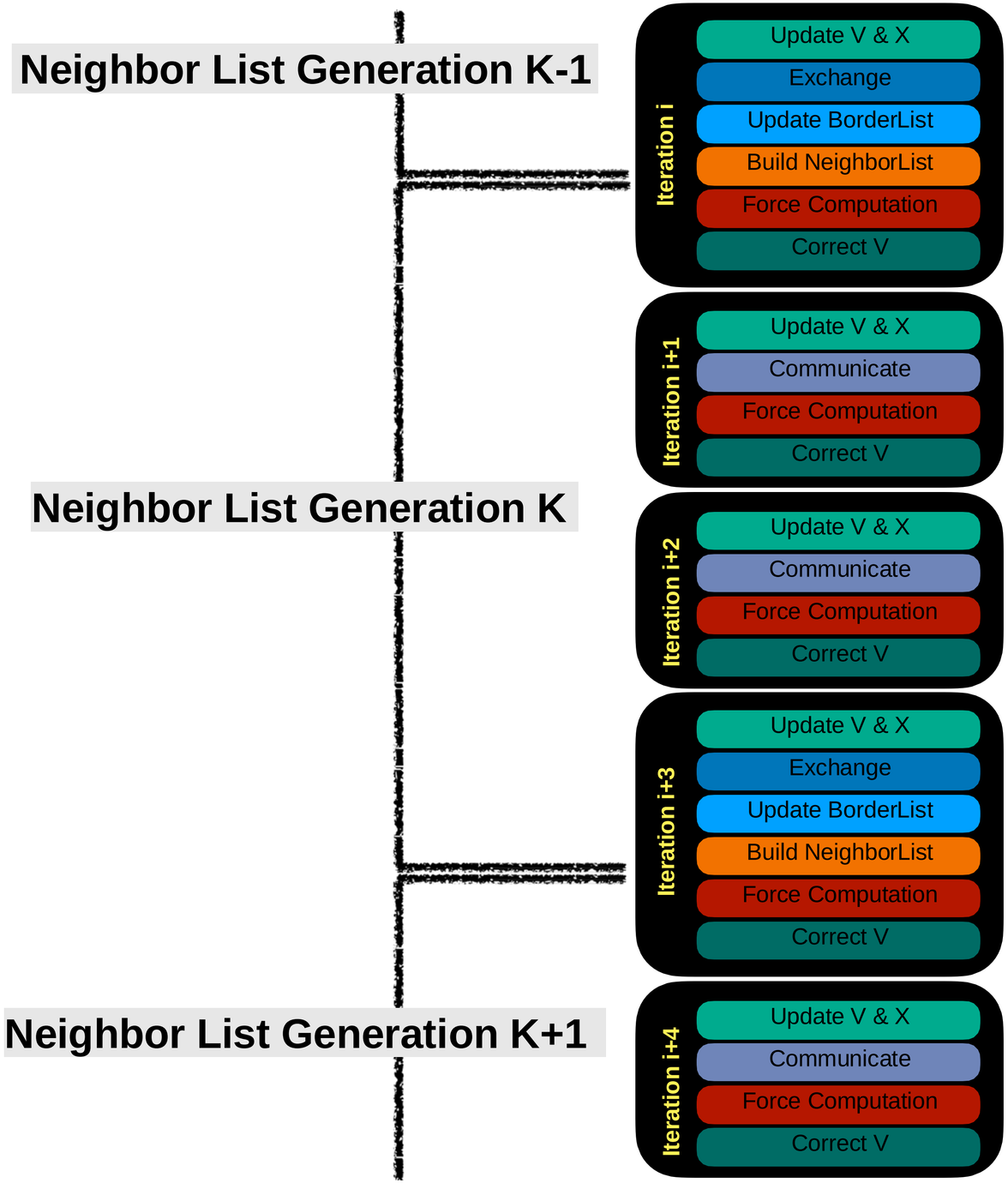
Baseline MiniMD

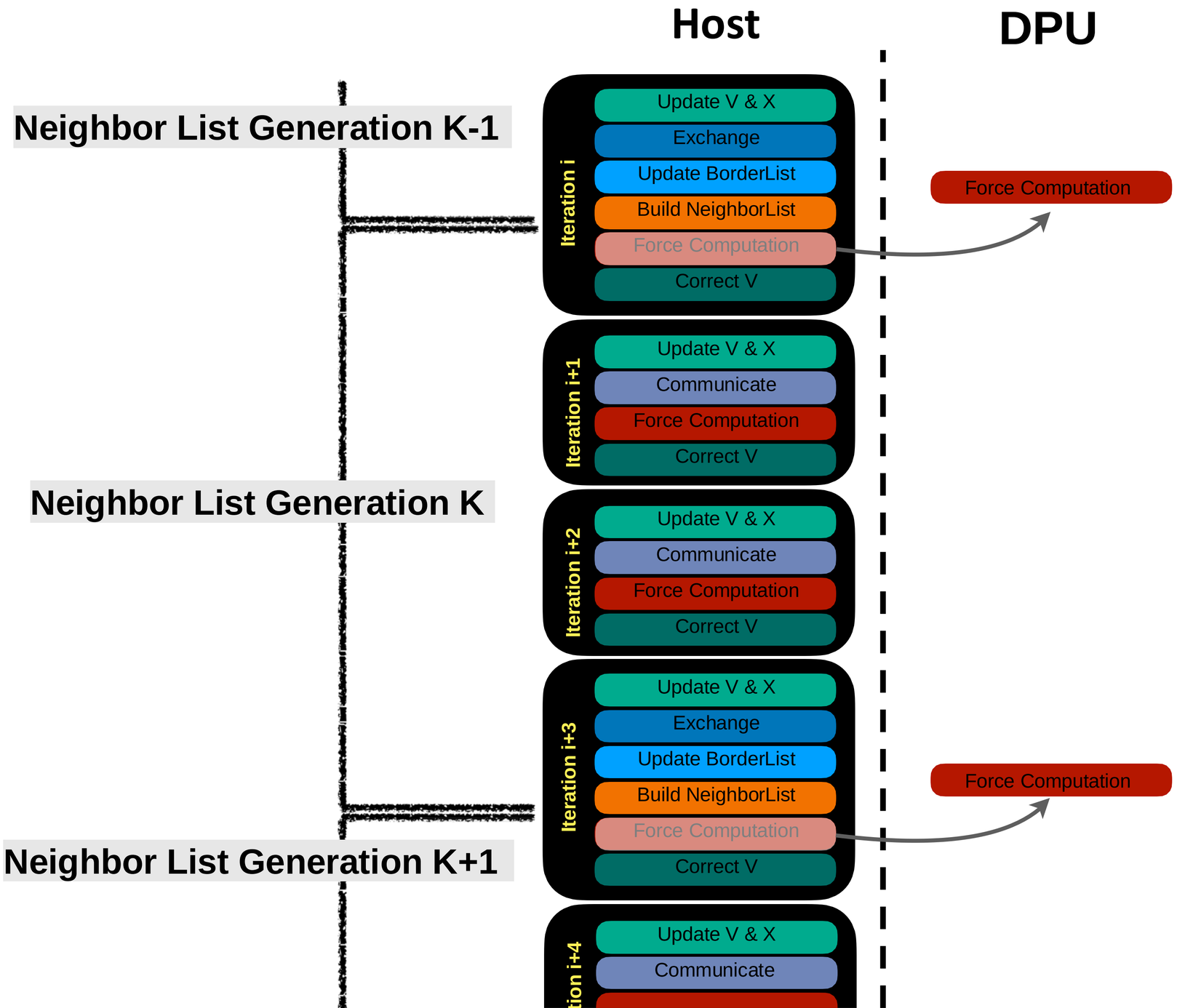
The neighbor list must be updated as particles move. But such **updates are expensive**! So every list includes a buffer of “extra” particles that lie within a surrounding annulus, or “**skin**,” parameterized by its thickness (Δ).

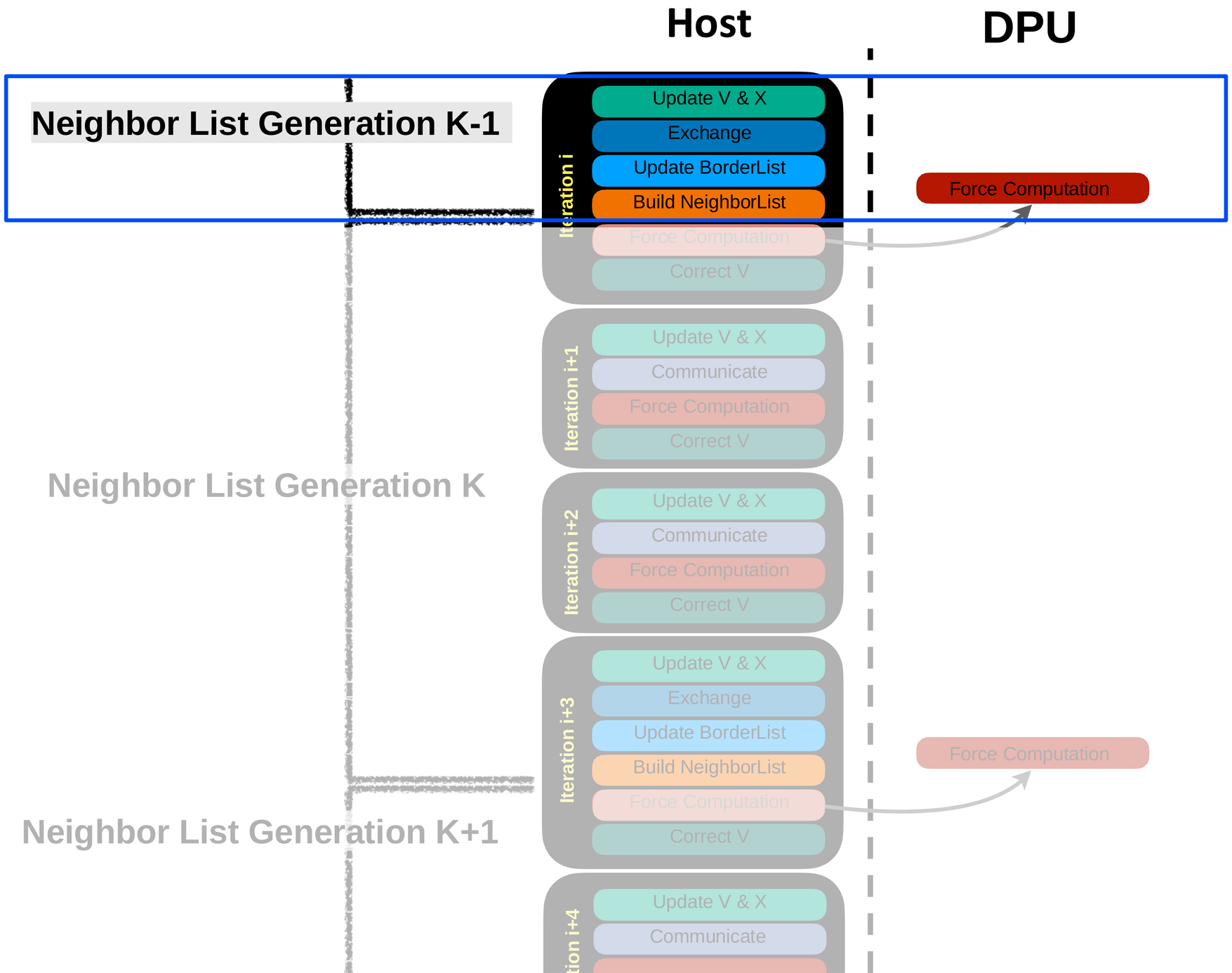


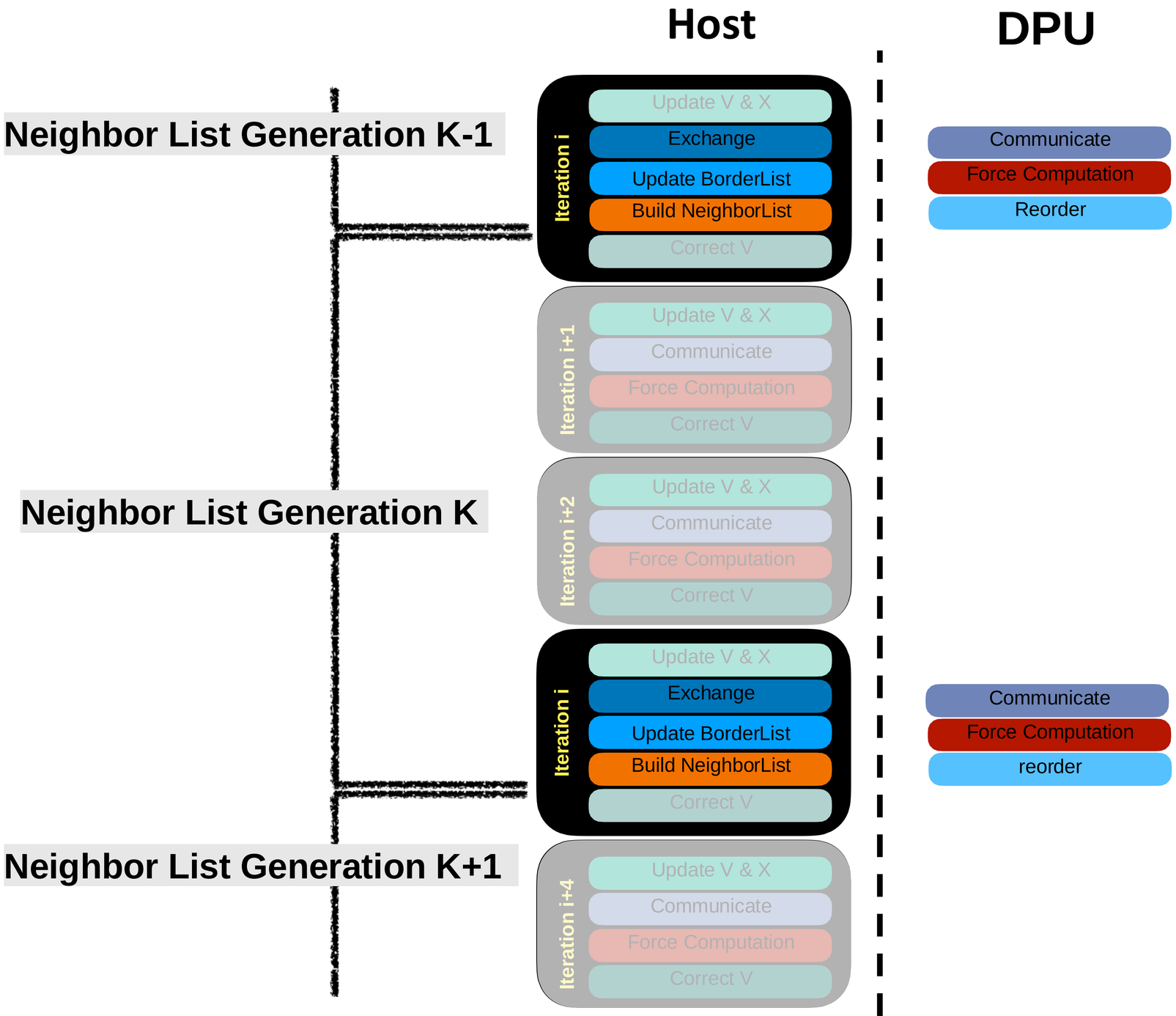
Serial dependencies

Each task is **parallelizable** but the sequence is **sequential** as shown by edges




Host
DPU
Neighbor List Generation K-1
Neighbor List Generation K
Neighbor List Generation K+1
Iteration i
Update V & X
Exchange
Update BorderList
Build NeighborList
Force Computation
Correct V
Iteration i+1
Update V & X
Communicate
Force Computation
Correct V
Iteration i+2
Update V & X
Communicate
Force Computation
Correct V
Iteration i+3
Update V & X
Exchange
Update BorderList
Build NeighborList
Force Computation
Correct V
Iteration i+4
Update V & X
Communicate
Force Computation
Force Computation
Force Computation


Host
DPU
Neighbor List Generation K-1
Iteration i
Update V & X
Exchange
Update BorderList
Build NeighborList
Force Computation
Correct V
Force Computation
Iteration i+1
Update V & X
Communicate
Force Computation
Correct V
Neighbor List Generation K
Iteration i+2
Update V & X
Communicate
Force Computation
Correct V
Iteration i+3
Update V & X
Exchange
Update BorderList
Build NeighborList
Force Computation
Correct V
Force Computation
Neighbor List Generation K+1
Update V & X
Communicate


Host
DPU
Neighbor List Generation K-1
Neighbor List Generation K
Neighbor List Generation K+1
Iteration i
Update V & X
Exchange
Update BorderList
Build NeighborList
Correct V
Iteration i+1
Update V & X
Communicate
Force Computation
Correct V
Iteration i+2
Update V & X
Communicate
Force Computation
Correct V
Iteration i
Update V & X
Exchange
Update BorderList
Build NeighborList
Correct V
Iteration i+4
Update V & X
Communicate
Force Computation
Correct V
Communicate
Force Computation
Reorder
Communicate
Force Computation
reorder

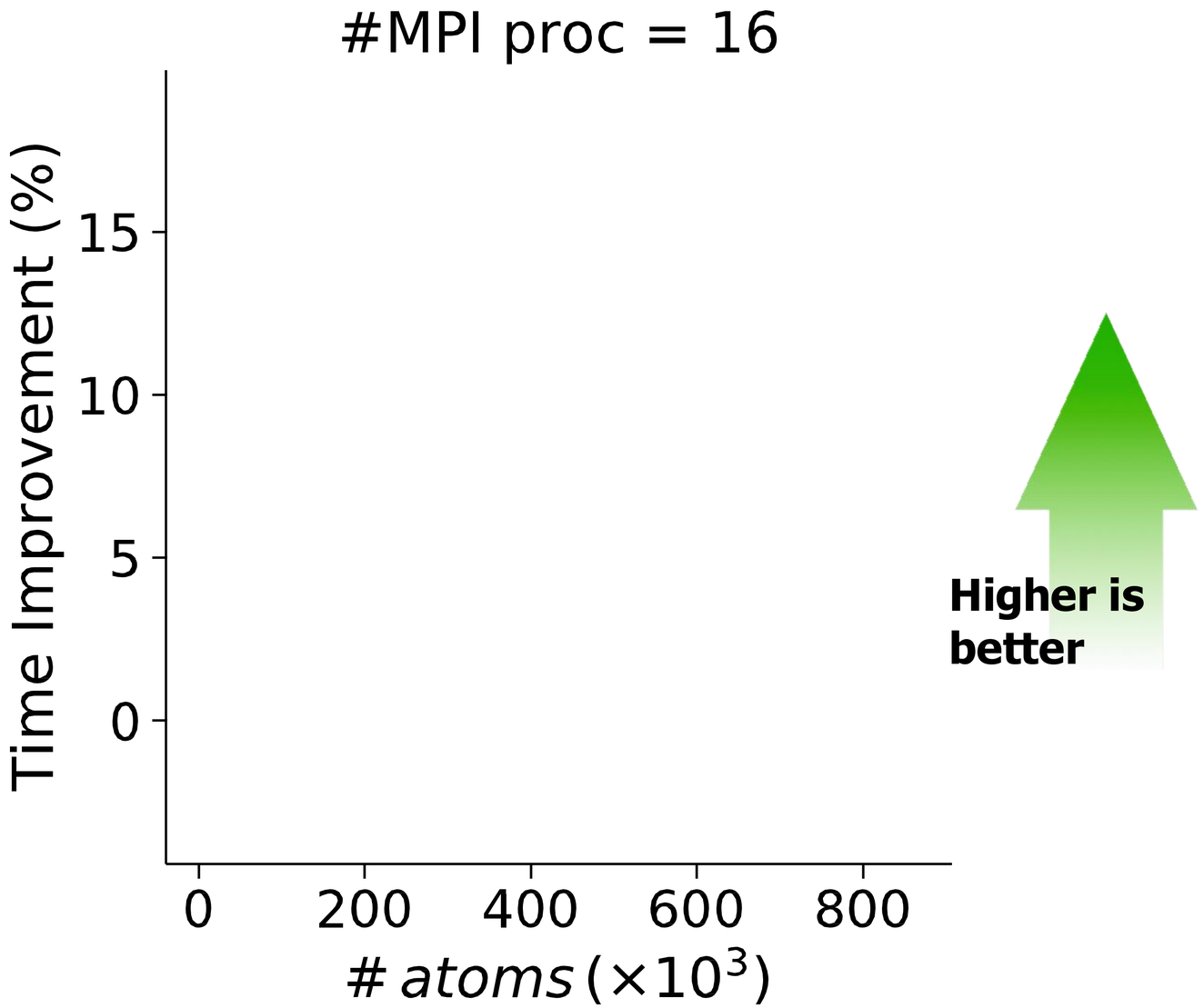
Baseline experiments

- **System**: 16 nodes, Infiniband HDR (100 Gbps)
- **Hosts**: (2-socket) x (16-core Intel Broadwell E5-2697A, 2.6 GHz) + (256 GiB DDR4 RAM, 2400 MHz)

"THOR" CLUSTER, MAINTAINED BY THE HPC·AI ADVISORY COUNCIL [LINK]

- **NICs per node**
  - 1 x NVIDIA **ConnectX-6 HDR100** (100 Gbps) InfiniBand/VPI adapters
  - 1 x NVIDIA **BlueField-2 SoC** — (8-core ARMv8 A72, 2.5 GHz) + (16 GiB DDR4 RAM) + (HDR100)

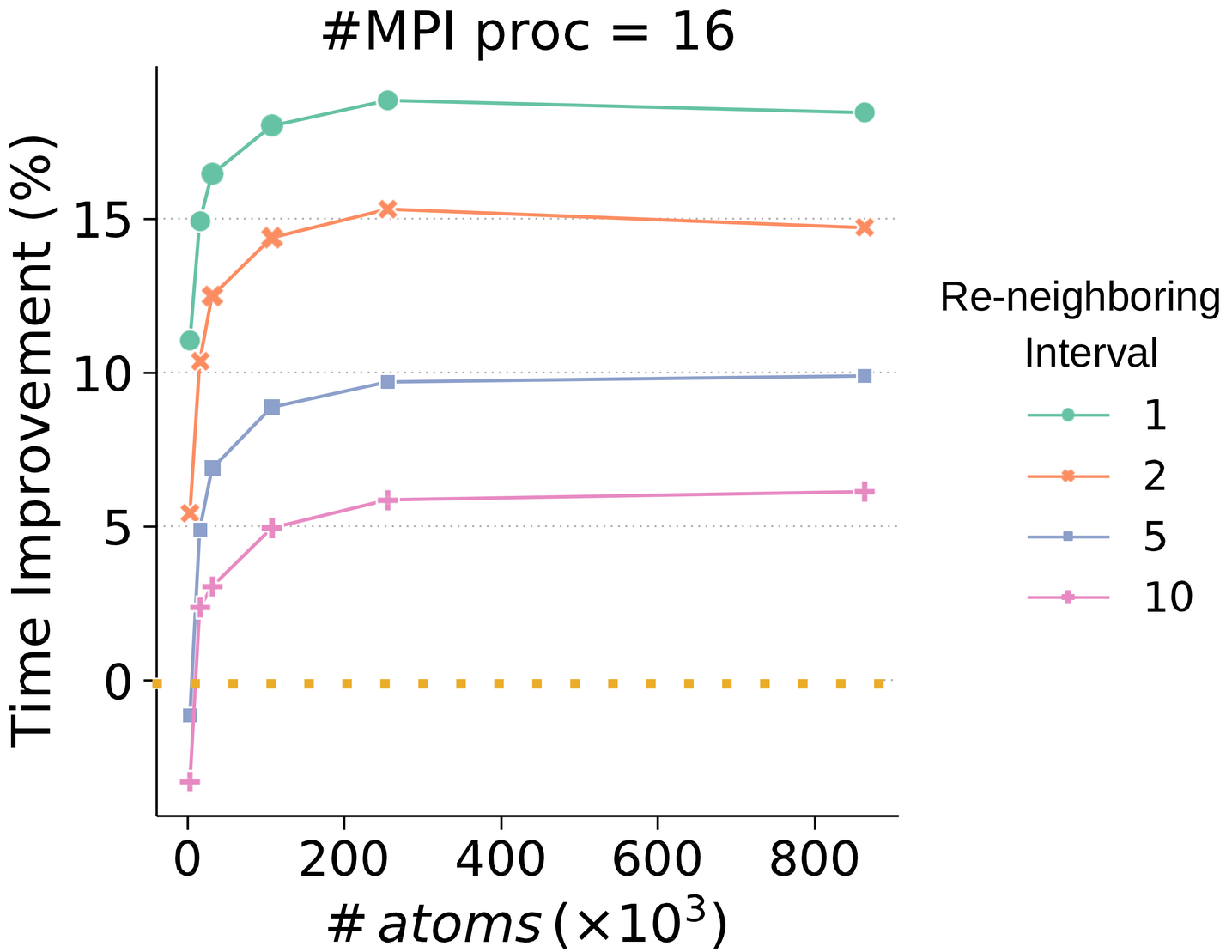
Restructured method …



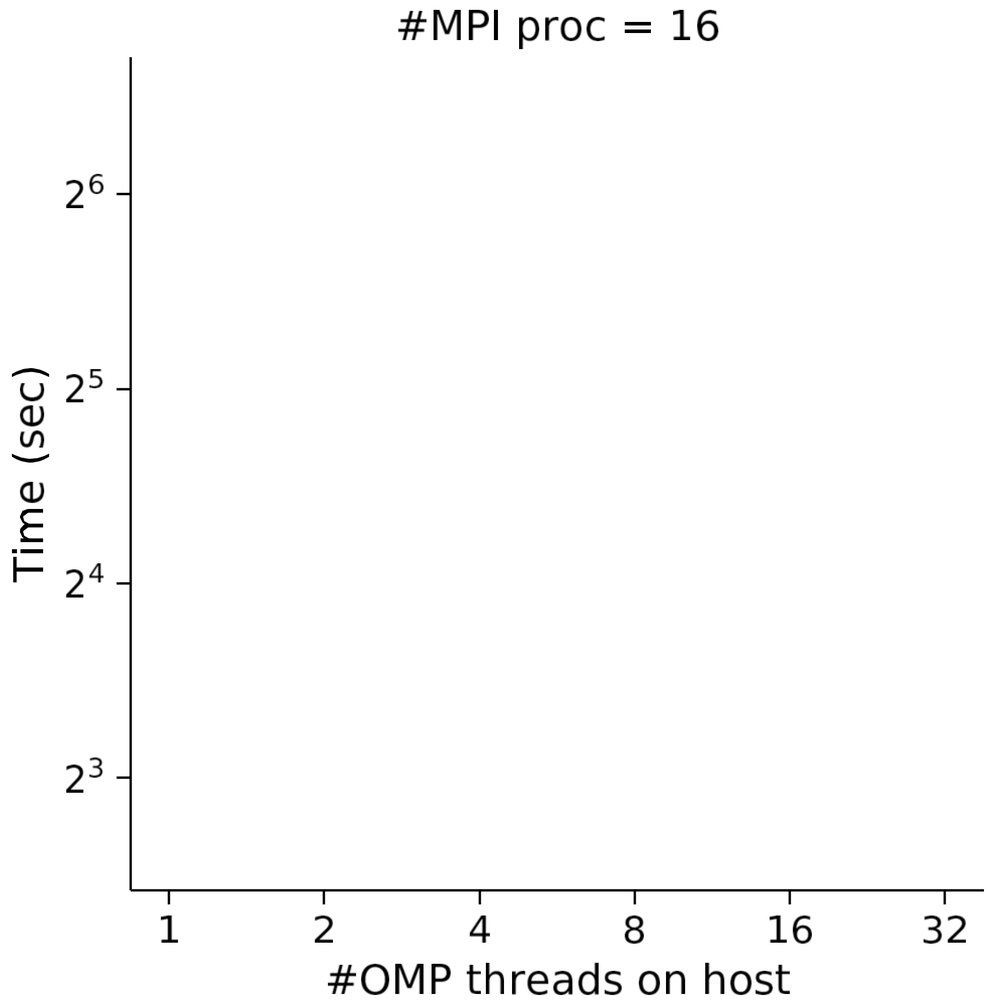
Restructured method is faster

We observe small, but largely uniform, **speedups of up to 20%** compared to host-only execution with conventional NICs.

This improvement compares favorably with the **power increase** on each node due to BF2, which we estimate from sensors to be **as little as 6%**.



Hybrid MPI/OpenMP performance results



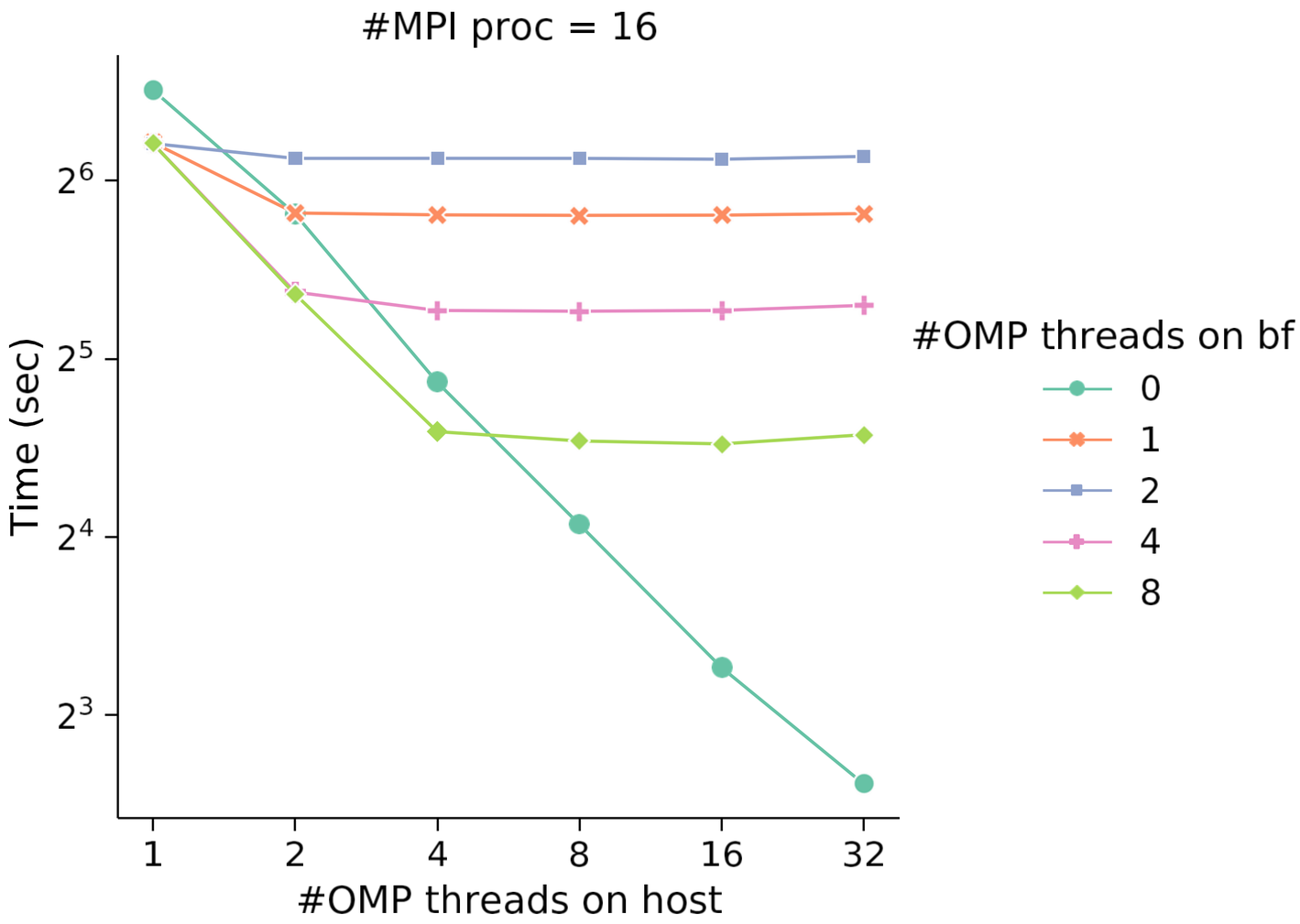
Lower is better

Hybrid MPI/OpenMP performance results

Our algorithm works best when it can completely hide the force computation time on BlueField.

The degree of achievable overlap depends on the relative computational power of the host and BlueField.

The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

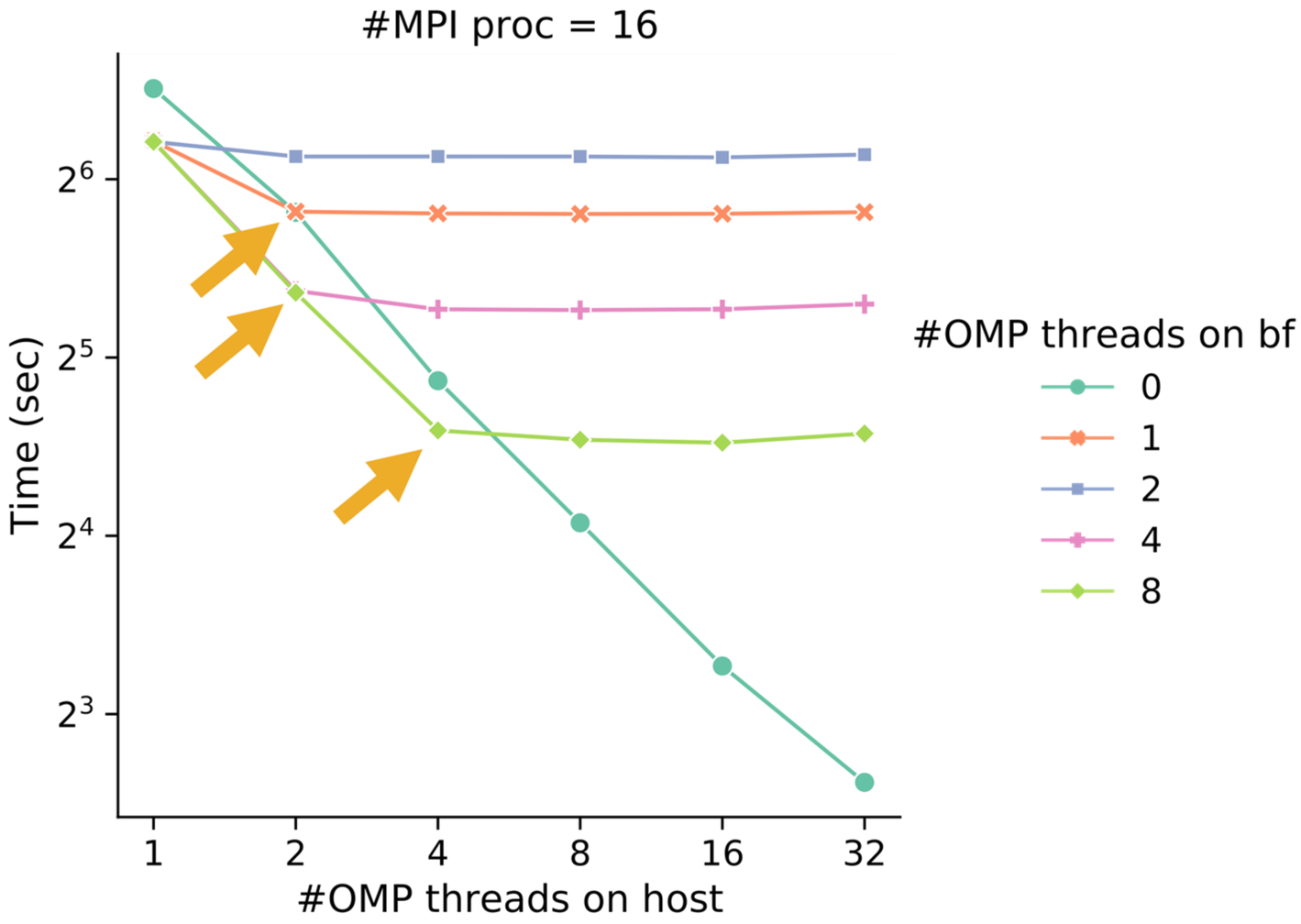


Hybrid MPI/OpenMP performance results

Our algorithm works best when it can completely hide the force computation time on BlueField.

The degree of achievable overlap depends on the relative computational power of the host and BlueField.

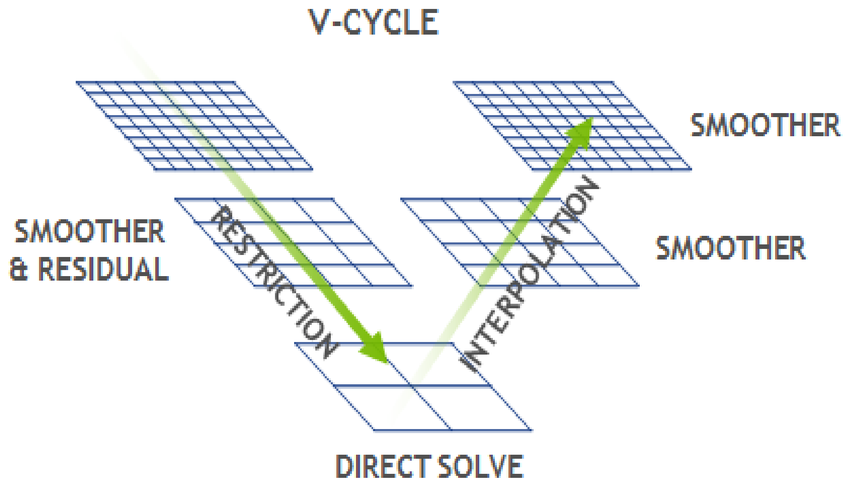
The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

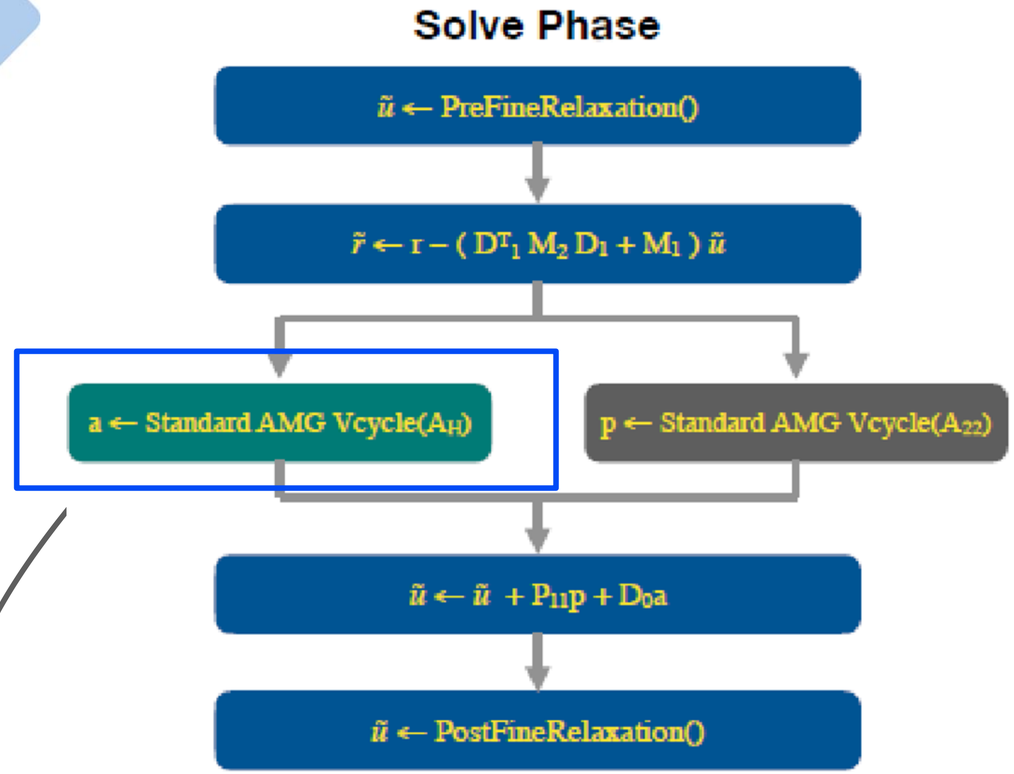


Current DPU Offload Work

- We believe that other HPC applications may also show benefits from offloading to the DPU

- For example, Maxwell Equation Solvers as implemented in the MueLu library represents a compelling distributed and shared memory parallel algorithm. These implementations can can be formulated using adaptive multigrid approaches and possible also parallelized to DPUs!





Possible Offload Candidate

Bochev, Pavel B., et al. "An algebraic multigrid approach based on a compatible gauge reformulation of Maxwell's equations." *SIAM Journal on Scientific Computing* 31.1 (2008)

MiniMD Demo – MPI Version

Running the miniMD Demo

Run the host application (built with x86) on the host

```
@thorbf3a008 ref]$ ./run_minimd_bf_mixed.sh

Running miniMD with NP=8, nThrHost=8, nThrBF=4
Running miniMD split across the host and BlueField device:
mpirun -np 8 -hostfile hostfile.in -bind-to cpu-list -map-by node ~/hotint_2023/miniMD/ref/miniMD_openmpi_x86_64 -t 8
: -np 8 -hostfile hostfile.bf.in -bind-to cpu-list -map-by node ~/hotint_2023/miniMD/ref/miniMD_openmpi_aarch64 -t 4
```

Run the BlueField (aarch64) application on the DPU

MiniMD Demo

- Find the code for this demo at https://github.com/gt-crnch-rg/smartnic-tutorial-sc24/tree/main/code/03-algorithm-restructuring-minimd
  - This modified version of MiniMD moves the force calculation to the BlueField
  - Execution starts on the BlueField and uses heterogenous MPI to launch across the Arm and x86 environments

MiniMD Demo – Offload Version

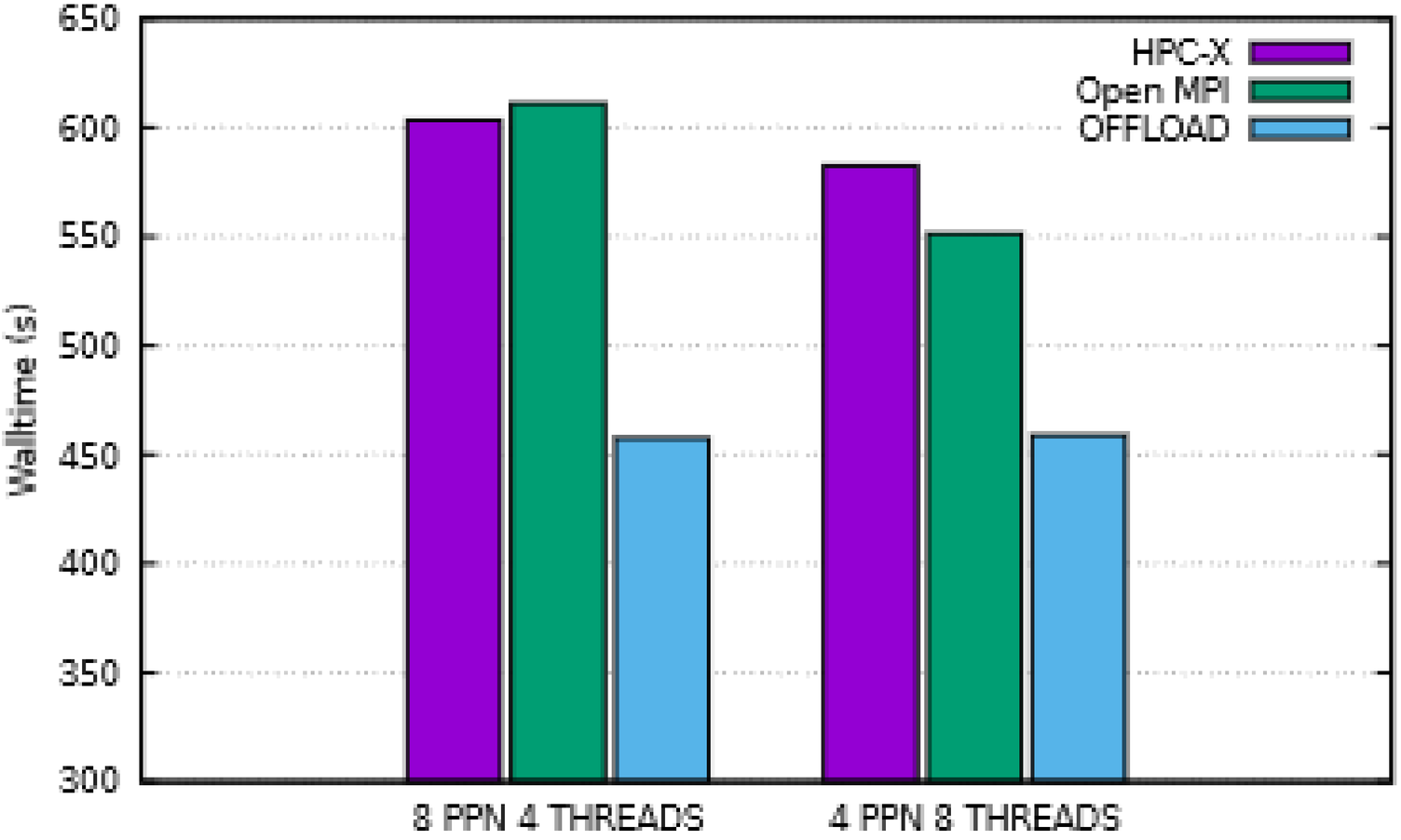
MiniMD Offload Demo

- Find the code for this demo at https://github.com/gt-crnch-rg/smartnic-tutorial-sc24/tree/main/code/04-offload-minimd
  - This modified version of MiniMD moves the force calculation to the BlueField using OpenMP offfload

Octopus

Octopus Performance*

- https://octopus-code.org/
- No modification to the code
- 32 nodes, 100 time steps



*R. Graham *et al*., "Optimizing Application Performance with BlueField: Accelerating Large-Message Blocking and Nonblocking Collective Operations," *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, Hamburg, Germany, 2024, pp. 1-12, doi: 10.23919/ISC.2024.10528935

Octopus Performance – MPI Time

- 32 nodes, 100 time steps, Average total time per process in seconds

Function	8P4T Host	8P4T Offloaded	4P8T host	4P8T Offloaded
Application	637.28	455.49	585.14	461.04
All Communications	245.99	138.15	230.66	142.37
MPI_Allgatherv	126.38	82.74	102.17	76.26
MPI_Waitall	57.60	32.02	73.65	41.40
MPI_Alltoall	37.89	13.87	30.03	17.45
MPI_Allreduce	7.44	4.96	2.88	4.05
Other	16.68	4.56	21.93	3.21

P3DFFT+

Introducing P3DFFT++

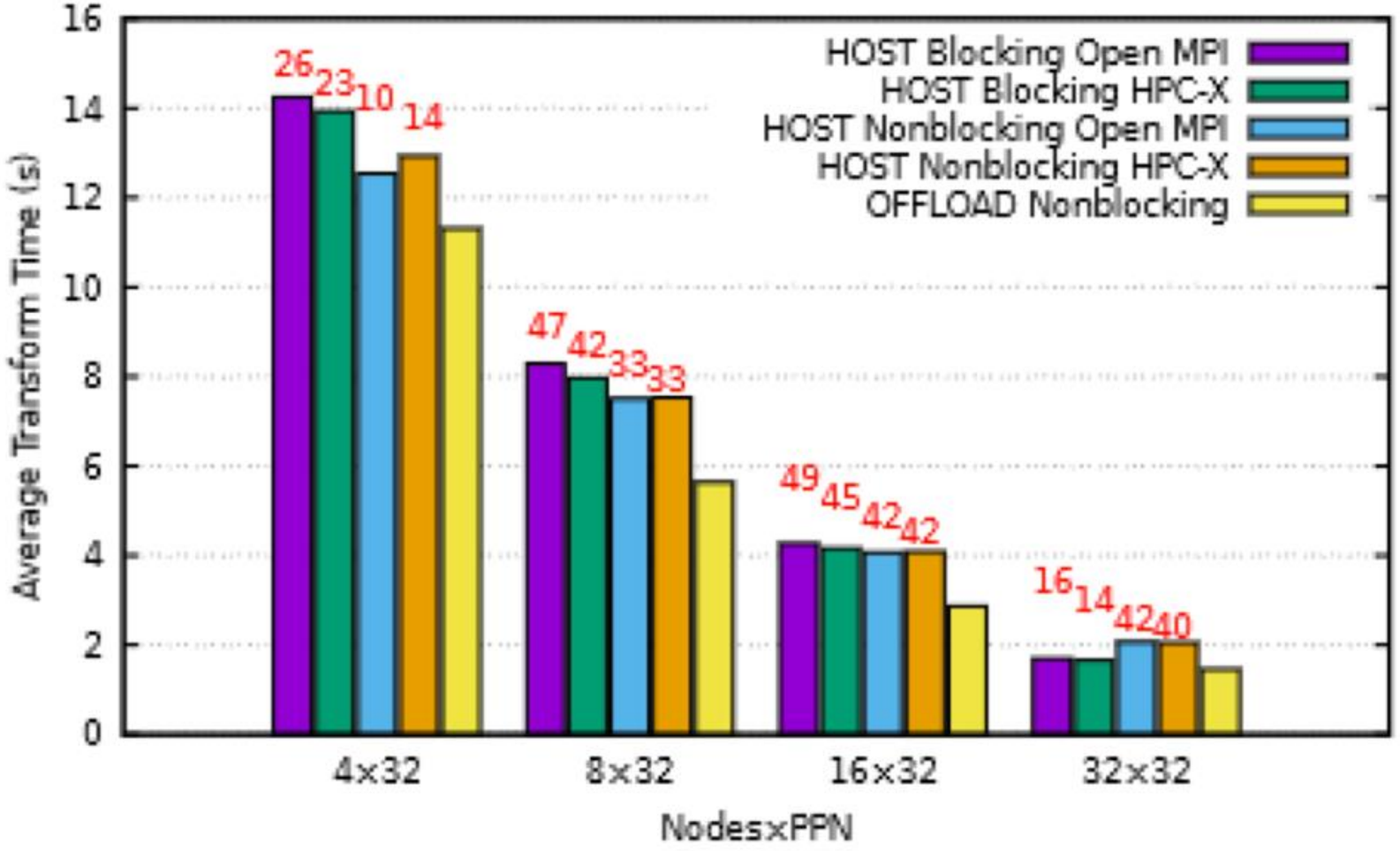
- Open source package implementing Fast Fourier Transforms in 3D based on 2D domain decomposition.
- Used in many areas of computational science, for purposes such as spectral analysis, solving non-linear differential equations with high precision, convolution and more
- Information about P3DFFT++ can be found at http://www.p3dfft.net.
  - The package can be downloaded from https://github.com/sdsc/p3dfft.3
- Implemented in C++ with MPI. Built on top of FFTW for 1D FFT.
- P3DFFT++ is an evolution of previous generation package P3DFFT, introducing innovative data layout and a modular design.
  - Package includes the FFT library and examples for using it in 3 languages (C, C++ and Fortran). These examples focus on /sample/C++/test3D_c2c_cpp.

P3DFFT++ and overlap

- 3D FFT algorithm with 2D decomposition consists of 5 stages: 3 compute stages (1D FFT/memory transpose) and 2 communication stages (all-to-all exchange within cartesian subcommunicators).
- At large scale communication becomes substantial and in some cases can even dominate the total execution time.
- Focus on overlapping communication with computation, using NVIDIA's DPUs
- The overlap happens when we pipeline the batch algorithm (multivariable execution, for example when we need to transform several independent variable, such as 3 velocity components). We can overlap communication phase of one variable with computation phase of another. (See branch **nb-mv** in P3DFFT++ github repository).

P3DFFT++ FFT TRANSFORM

4-variable transform



P3dFFT++ FFT Transform – MPI Time

- 4-variable transform, Average total time per process in seconds

Function	16 Nodes Host	16 Nodes Offload	32 Nodes Host	32 Nodes Offload
Application	49.53	36.39	26.07	20.57
All Communications	17.58	7.54	10.15	5.46
MPI_Waitany	10.20	4.88	6.32	3.22
MPI_Ialltoallv	2.98	1.38	1.65	0.77
Other	4.40	1.28	2.19	1.47

Wrap-Up

Summary

In this tutorial we have covered the following:

- Described how SmartNICs can be used as HPC/AI accelerators
- Described how users can benefit from acceleration engines in their current and future applications
- Described how one can use one family of SmartNICs to accelerate applications and communication libraries
- Shared demo experiences using SmartNICs with canned examples

SmartNICs and DPUs for HPC/AI is a wide open area for future research and engineering!

How to continue working with SmartNICs and DPUs

You can access SmartNICs via a variety of general access testbeds

- CRNCH Rogues Gallery testbed
  - BlueField 1-3, Alveo FPGAs, NapaTech SmartNICs
- ORNL Wombat
  - BlueField 2-3
- Xilinx HACC centers
  - AMD Alveos for OpenNIC work

Testbed Links

https://crnch-rg.cc.gatech.edu/

https://olcf.ornl.gov/olcf-resources/compute-systems/wombat/

https://www.xilinx.com/support/university/xup-hacc.html

Questions and Attendee Feedback

Please fill out the Mentimeter Survey and any SC24 surveys!