SC23
Denver, CO | i am hpc.

TUTORIAL:
Leveraging SmartNICs for HPC and Data Center Applications

Presenters: Richard Graham, Antonio Peña, Jeffrey Young
Helpers: Yong Qin, Richard Vuduc
Special Thanks To: Oscar Hernandez (ORNL), Muhammad Usman

NVIDIA

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Georgia Tech

Brief Audience Survey

Audience Survey

Tutorial Testbed and System Access

Tutorial Testbed and System Access

HPC-AI Advisory Council

THOR Configuration

- Dell™ PowerEdge™ R730 32-node cluster
- Dual Socket Intel® Xeon® 16-core CPUs E5-2697A V4 @ 2.60 GHz
- Memory: 256GB DDR4 2400MHz RDIMMs per node
- **NVIDIA BlueField-3 SoC, NDR200 200Gb/s InfiniBand/VPI adapter (*)**
- **NVIDIA BlueField-2 SoC, HDR100 100Gb/s InfiniBand/VPI adapter (**)**
- NVIDIA NDR Quantum-2 Switch QM9700 40-Port 400Gb/s InfiniBand
- NVIDIA HDR Quantum Switch QM7800 40-Port 200Gb/s InfiniBand
- NVIDIA HDR Spectrum-2 Switch SN3700 32-Port 200Gb/s Ethernet

- ssh rdmaworkshop##@jumpbox
- ssh gw.hpcadvisorycouncil.com
- git clone https://github.com/gt-crnch-rg/smartnic-tutorial-sc23

Tutorial Agenda

Agenda (Timing)

1:30-1:40	Introduction and setting up system access	Yong / Jeff (survey)
1:40-2:10	SmartNIC introduction and overview	Rich G.
2:10-2:20	Programming approaches for HPC with SmartNICs	Jeff
2:20-2:30	Transforming applications with MPI and SHMEM	Rich G
2:30-2:40	Heterogeneous MPI Applications: Host + SmartNIC ranks	Jeff
2:40-3:00	Using the SmartNIC as an Communicaton Accelerator	Yong
3:00-3:30	Break	
3:30-4:00	OpenMP offload to the SmartNIC	Toni
4:00-5:00	Hands on exercises	All

See the updated agenda at https://github.com/gt-crnch-rg/smartnic-tutorial-sc23

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
- Give hands-on experience using SmartNICs with canned examples

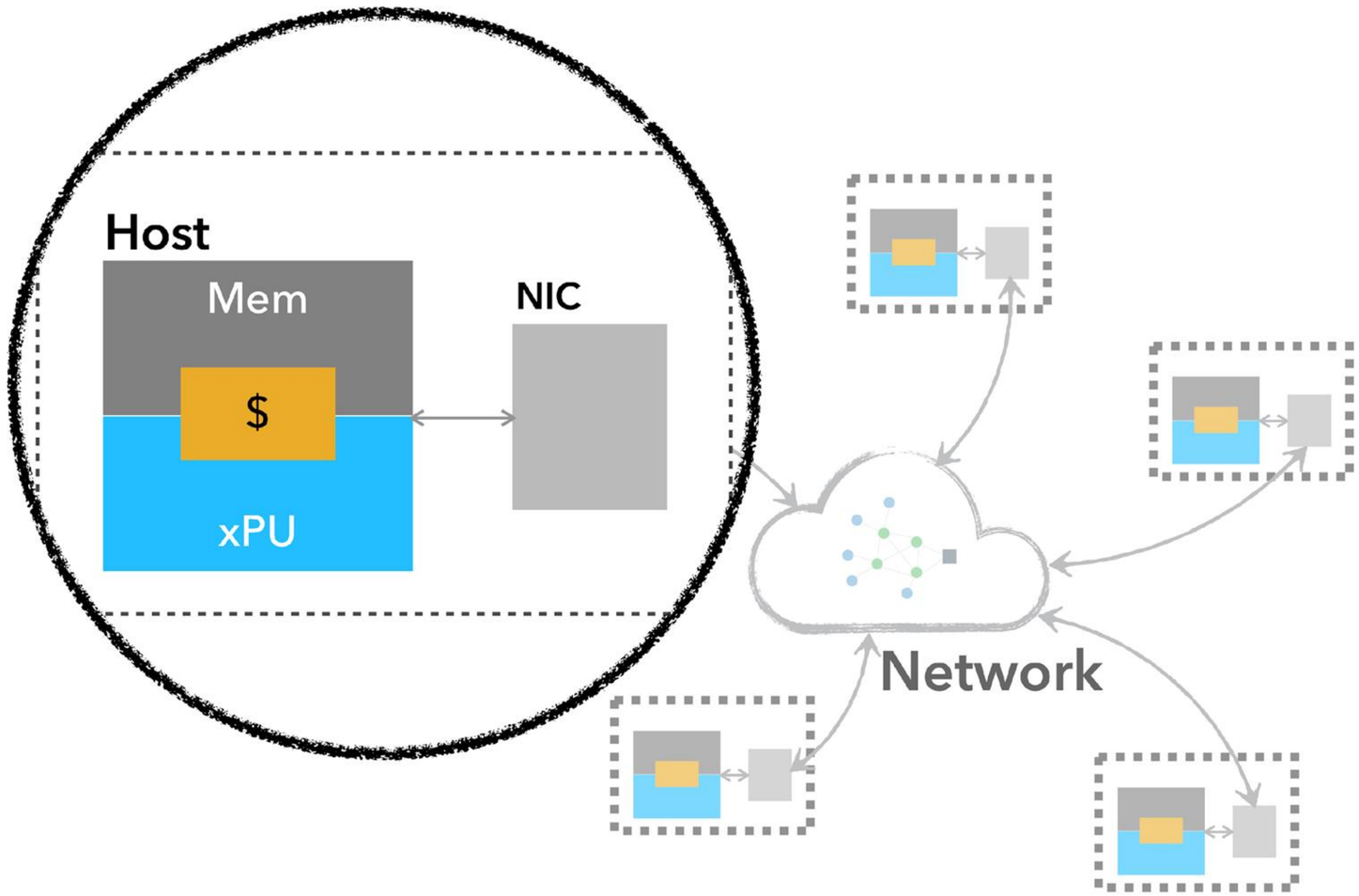
Anatomy of a supercomputer

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

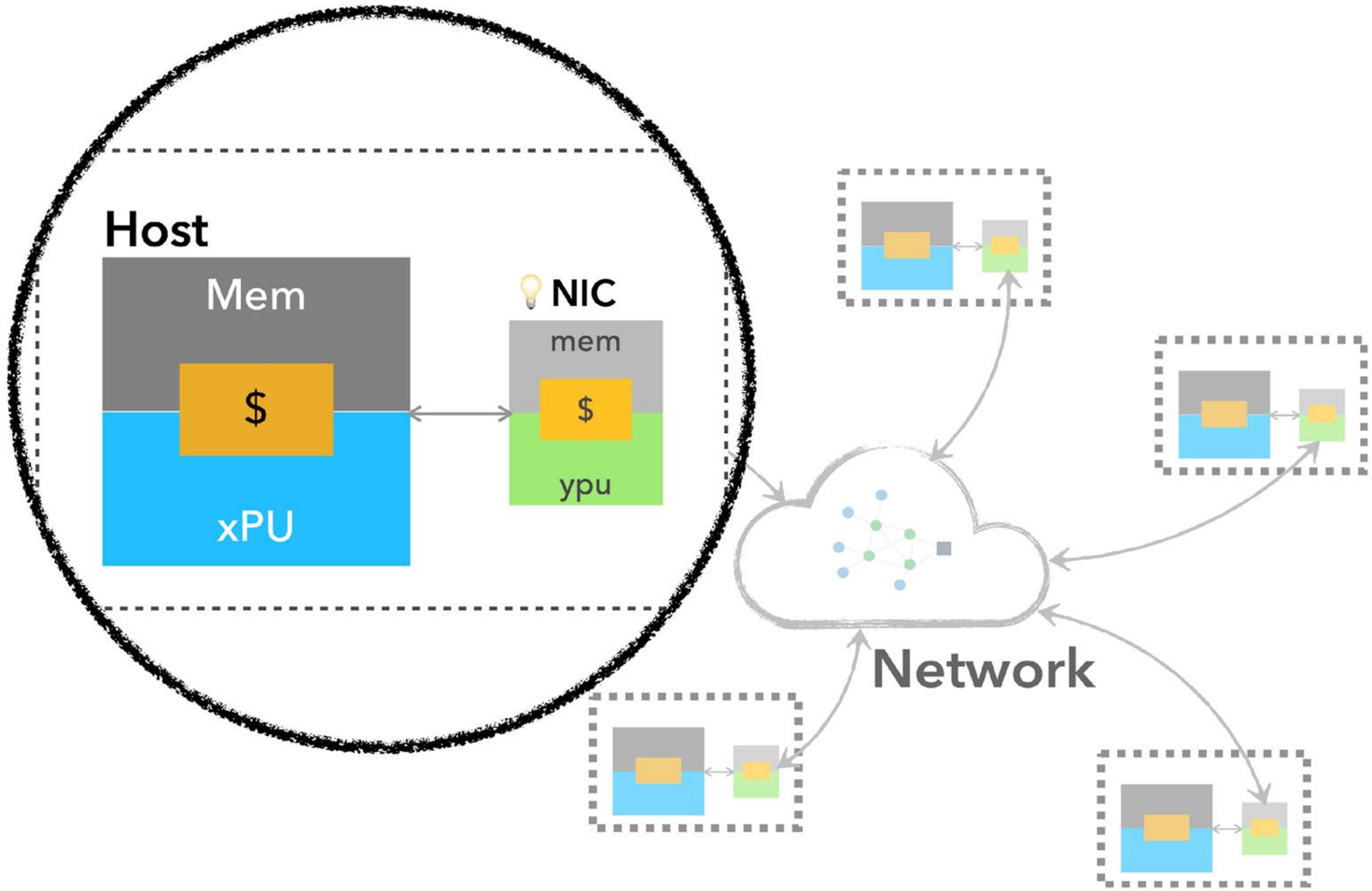
The basic building block of a distributed-memory cluster or supercomputer is a node.

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.

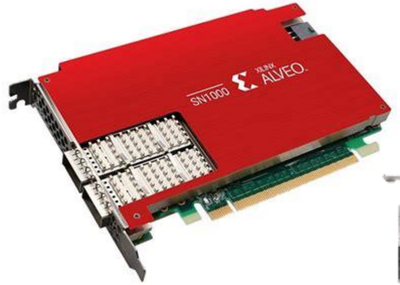
In a DPU, the NIC becomes "host-like" via the addition of processing (ypu), memory and other accelerations engines.

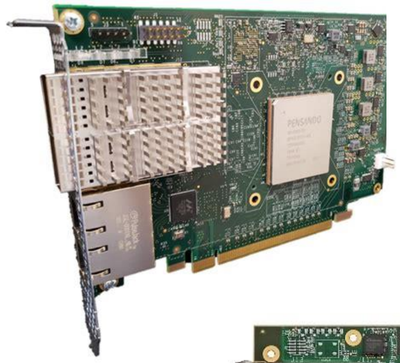


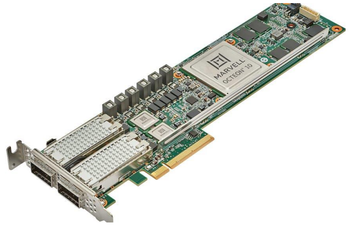
SmartNICs and Data Processing Units (DPUs)

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

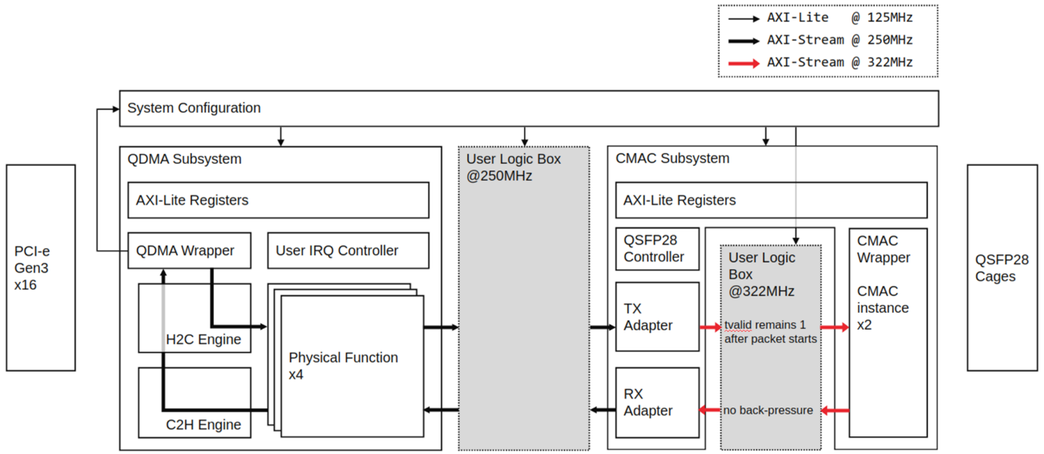






DPU Example - AMD SmartNICs

- 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.



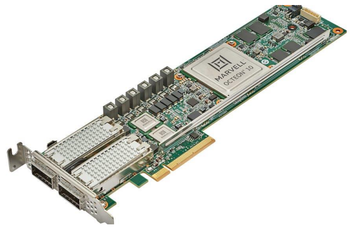
OpenNIC overview diagram from https://github.com/Xilinx/open-nic

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

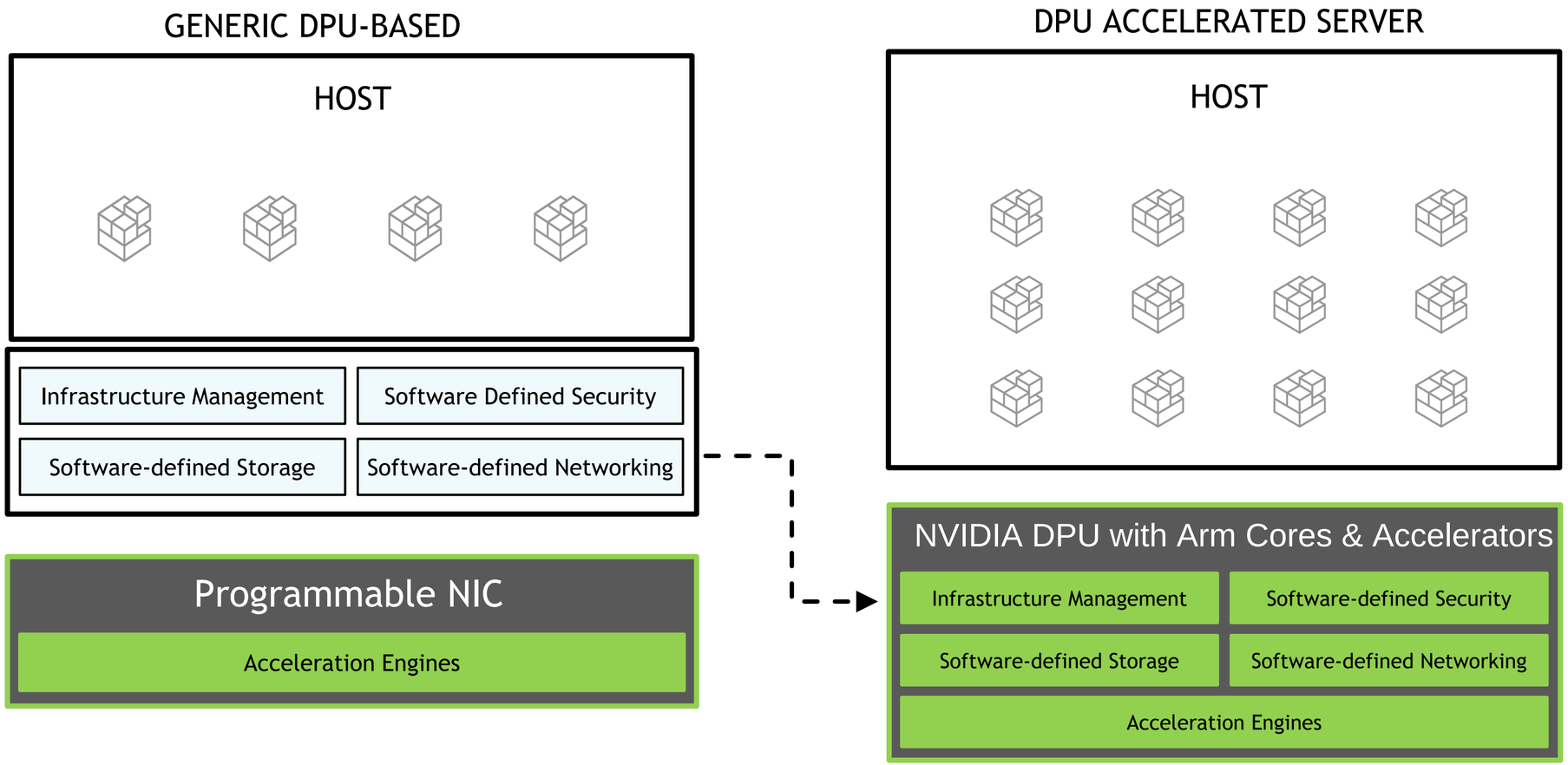
- Intel Processing Units (IPU) focuses on accelerating common network services for virtualized environments
  - Offload networking stack, storage processing, security functions and even hypervisor capabilities
  - IPUs contain a Stratix/Agilex FPGA for line-speed packet processing and a CPU for serial 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

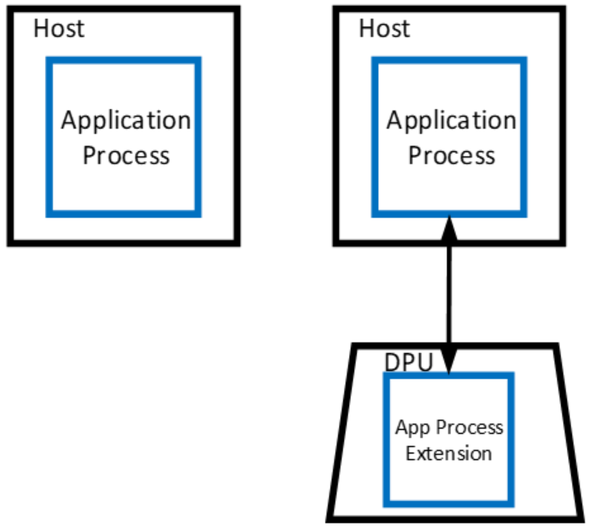
NVIDIA’s BlueField DPUs

NVIDIA’s BlueField Data Processing Unit



DPU offloading Models

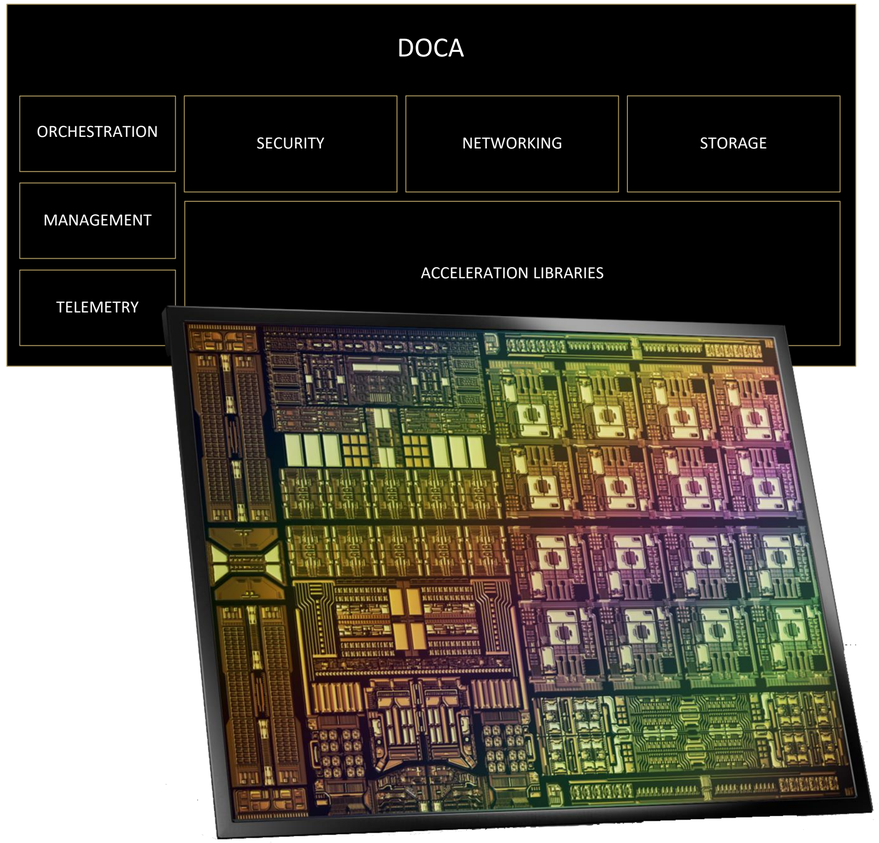
Extension of application processing – algorithms split between host and DPU



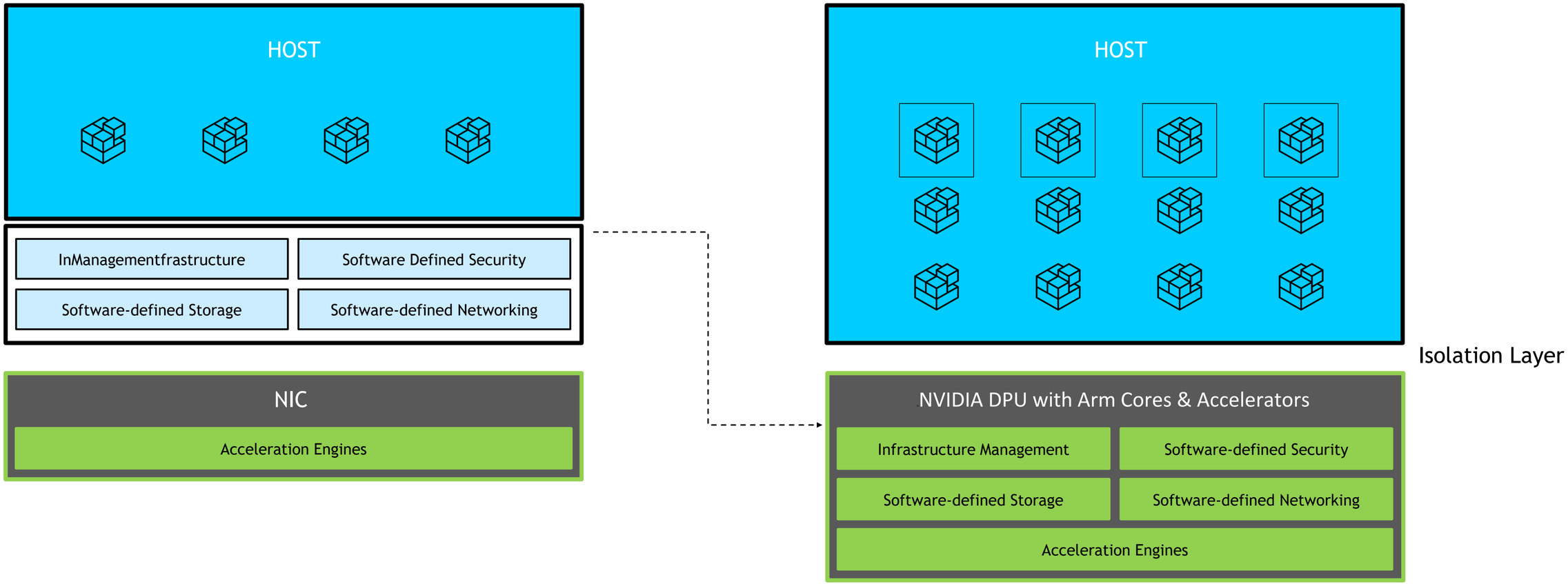
- **Offloaded system services**
  - Partial offload, such as storage – similar to application offload
  - Full offload, such as system security services
- Fully offloaded application service
  - Such as application telemetry collection

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

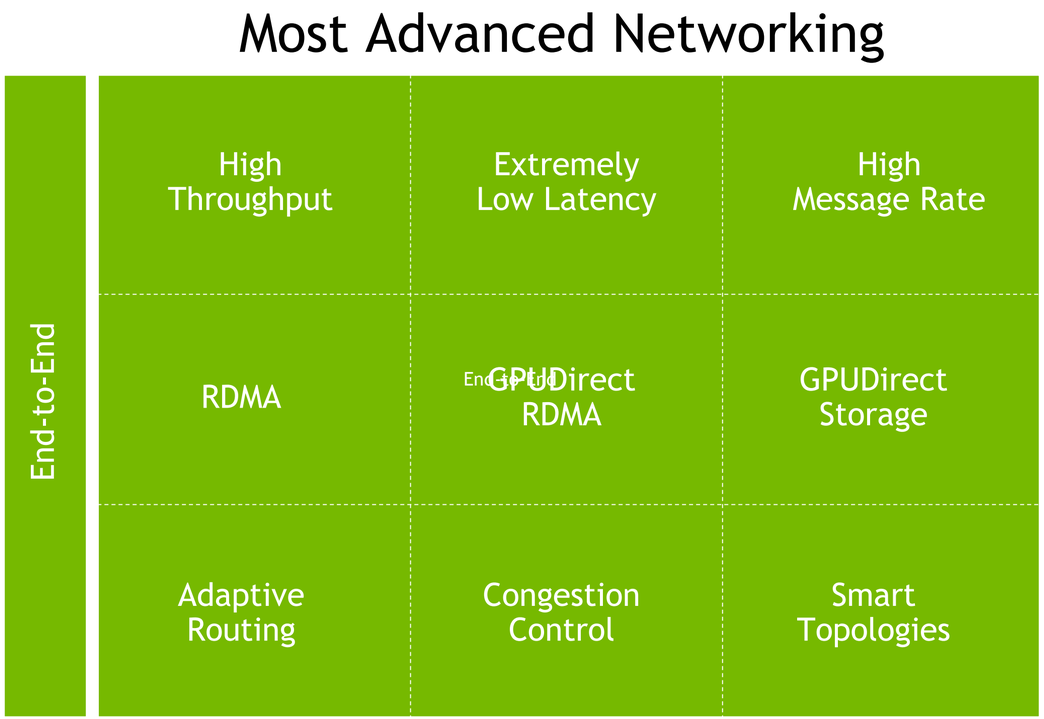


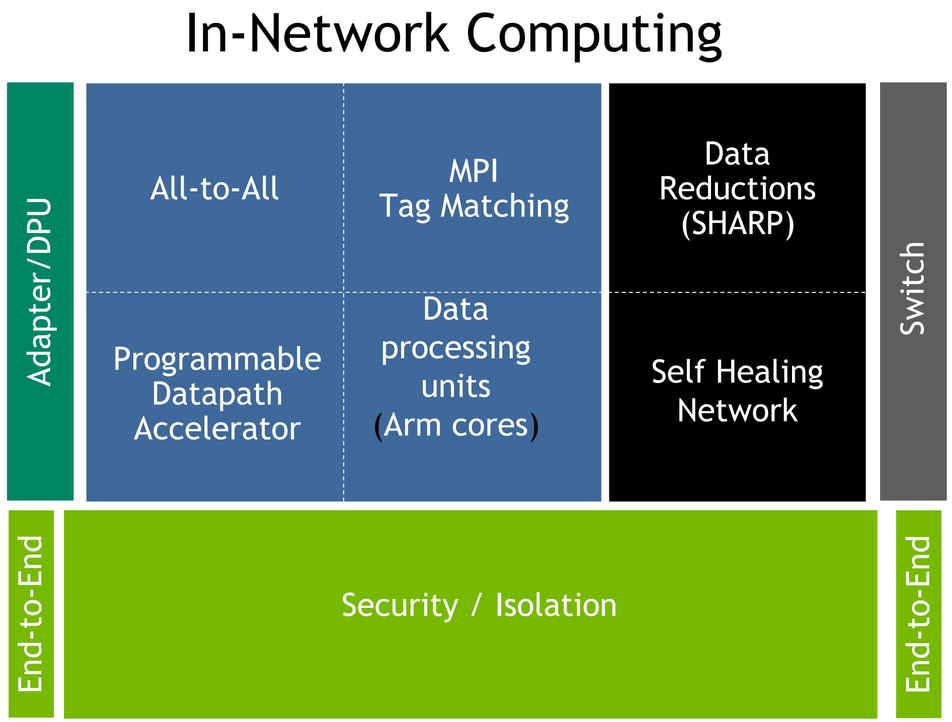
BlueField Data Processing Unit



- 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

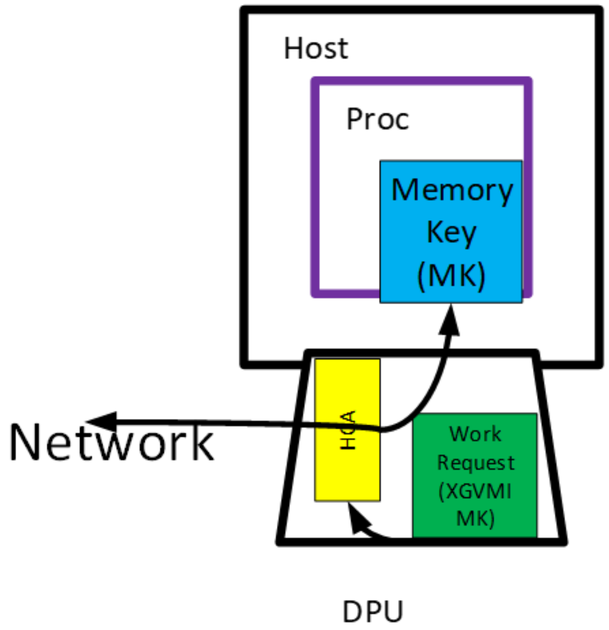
- Possess memory bandwidth independent of that of the host
  - Selectively use this memory resource to supplement what is available in the compute complex – not an all or none proposition

- 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

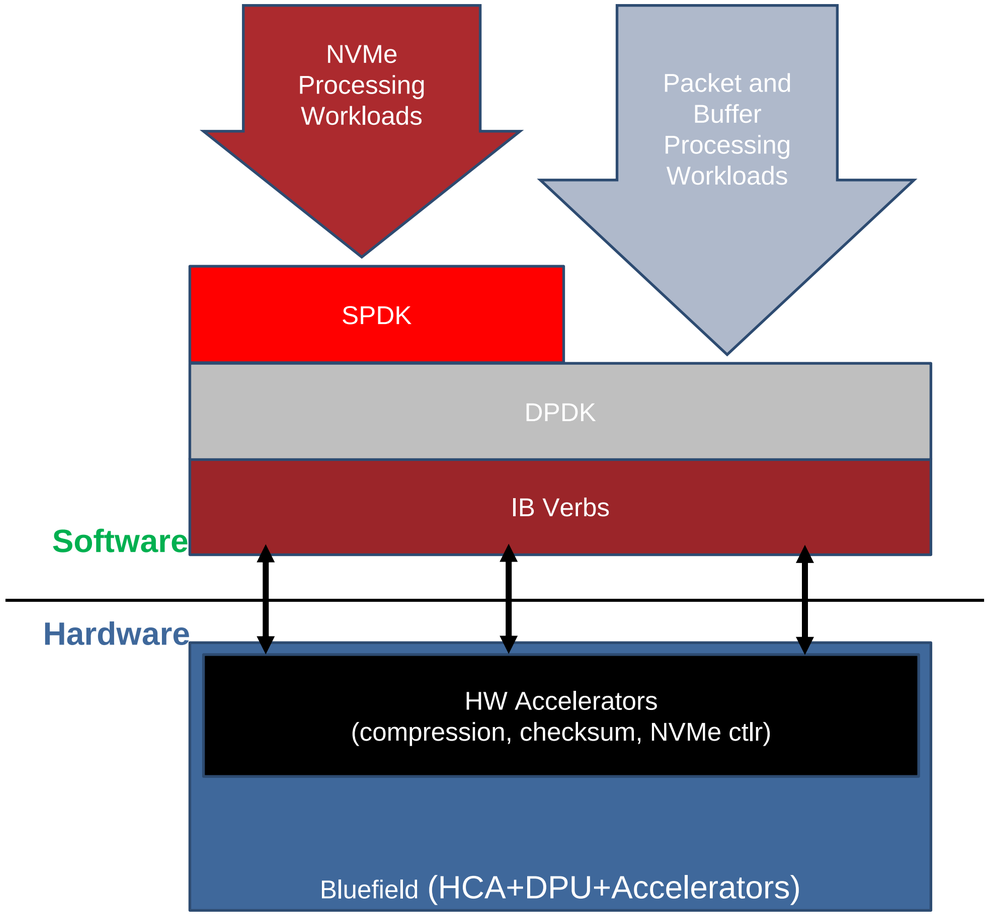
BlueField-2: Accessing Common Offloads

- 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

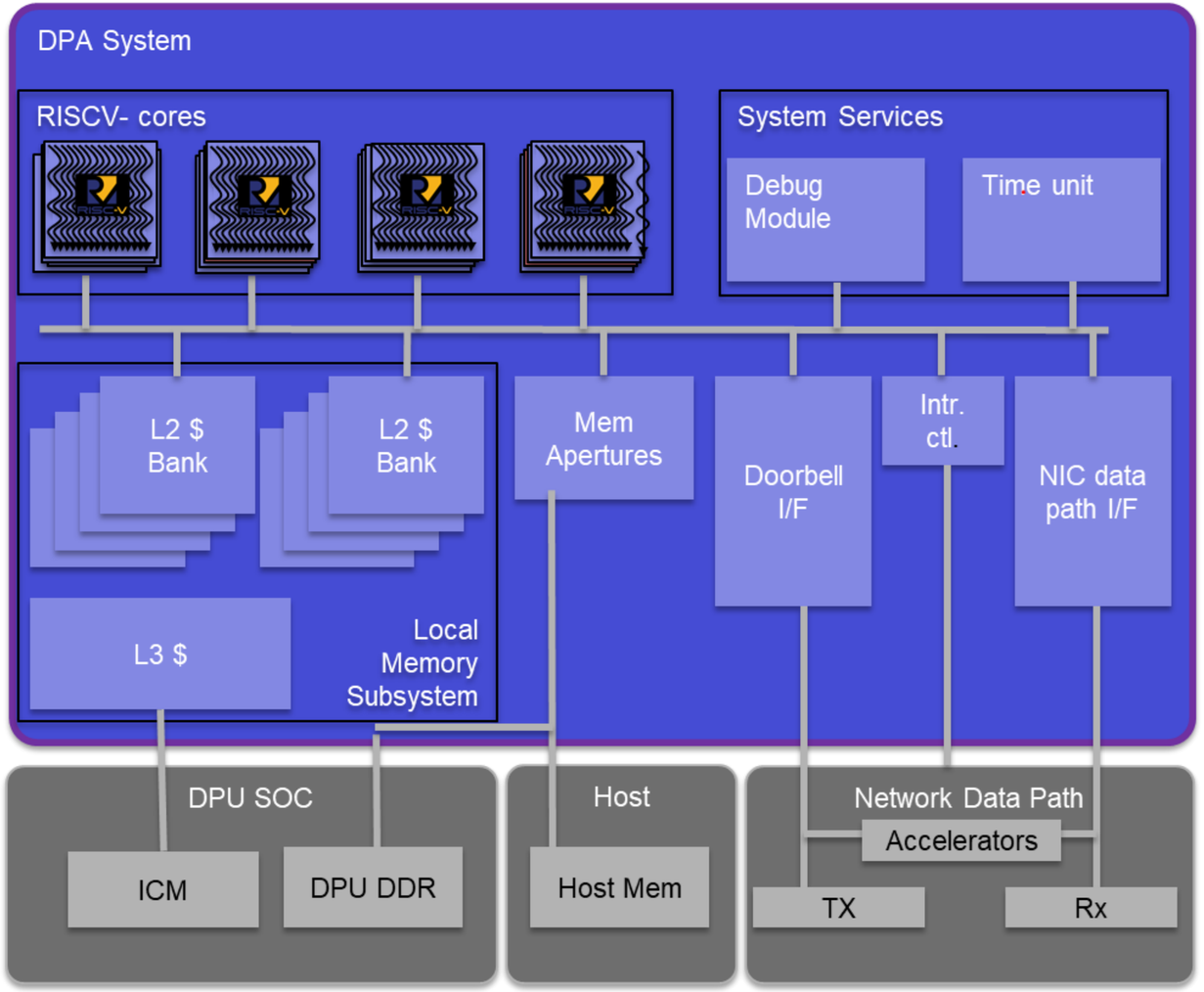
- SPDK offload example:

  Simple NVMe-oF Target Offload Benchmark (force.com)
  https://mymellanox.force.com/mellanoxcommunity/s/article/simple-nvme-of-target-offload-benchmark



Data path accelerator (DPA)

- RV64IMAC(B) + NVIDIA extensions (e.g. arithmetic ops)
- 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

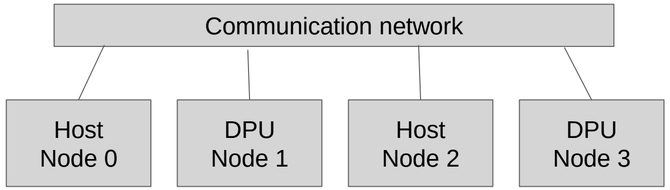
DPA			
Architecture		Standard RISC-V RV64IMAC(B)-USM	
Pipeline Width		1 Instruction / Clock	
Threads per Core		16	
Number of Cores		16	
Frequency Target		Core 1.8 GHz / System 505 MHz	
Caches		Capacity	Access / Clock
	L1 Code / Core	8 KB (2-4K inst.)	1 (per Core)
	L1 Data / Thread	1 KB	1 (per Core)
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

Programming Models and Frameworks

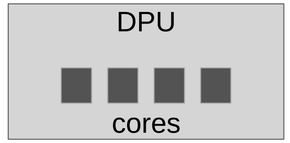
High-level HPC Programming Models

Four programming models are available to program DPUs:

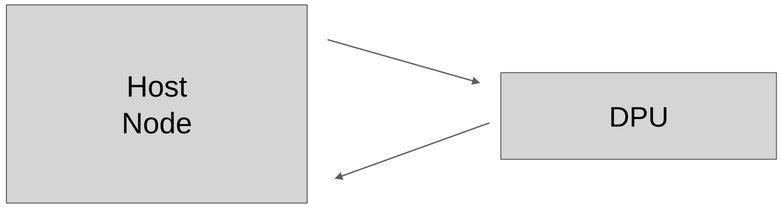
Distributed and distributed-shared memory
- MPI
- OpenSHMEM



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

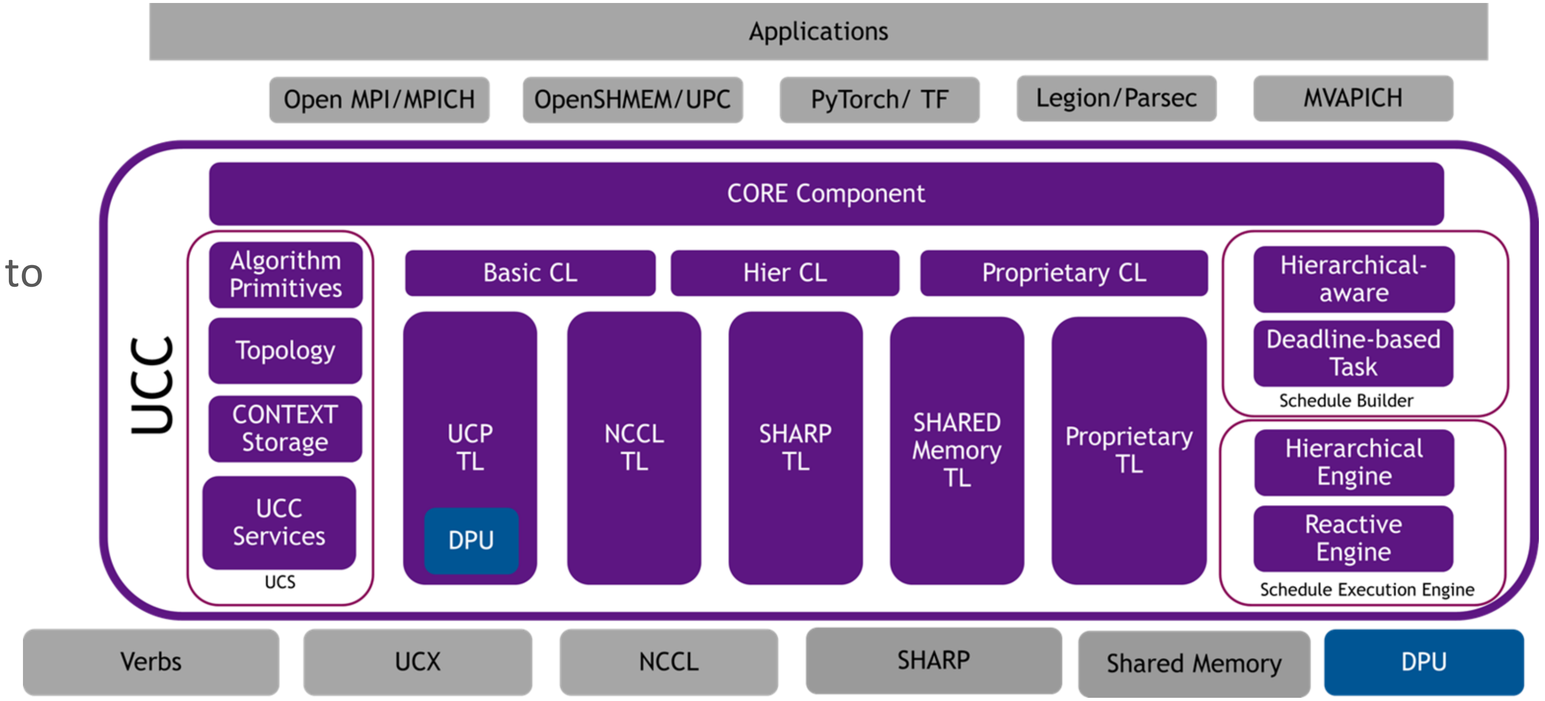
Accelerator Offload

- 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 exposes the accelerator
    - Accelerator is used by services the application uses such as MPI, OpenSHMEM and storage
    - Services explicitly interact with the accelerator

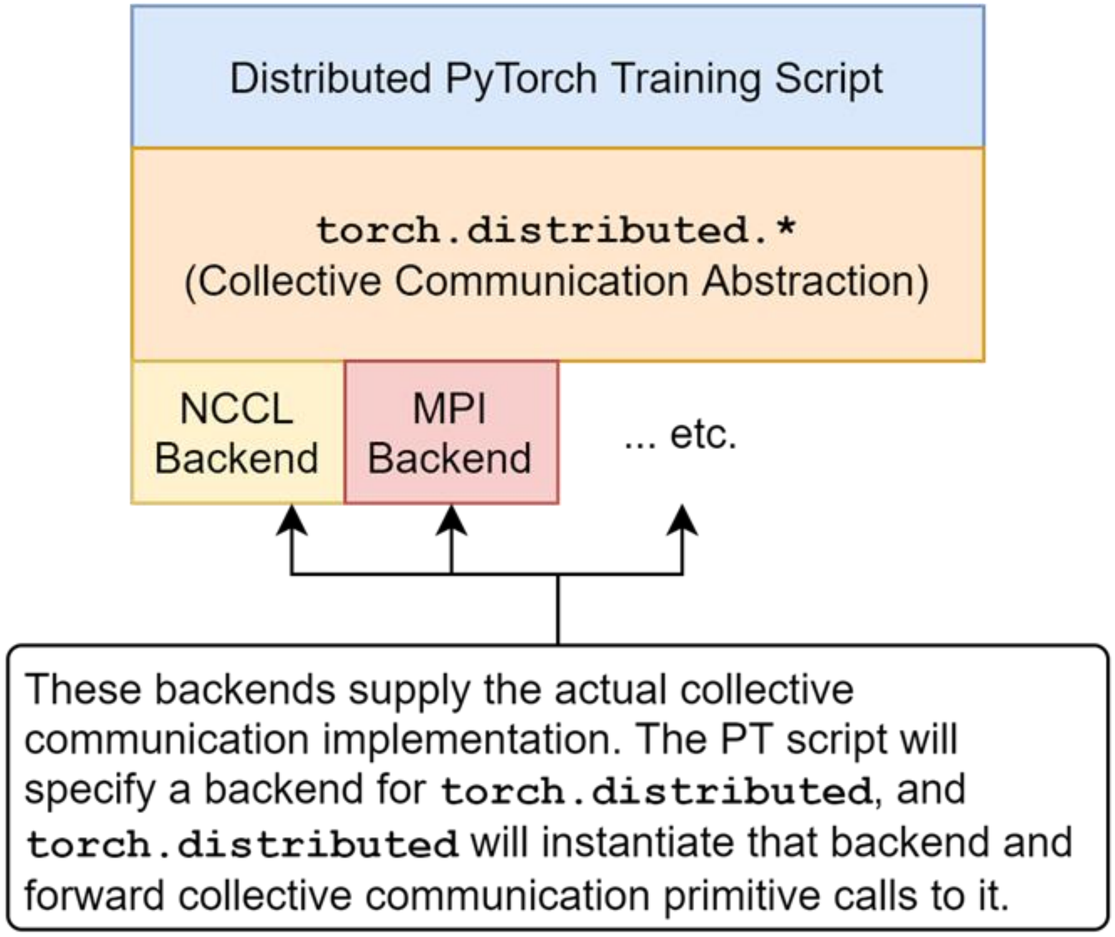
AI/PyTorch: support for low-level communication collective offload

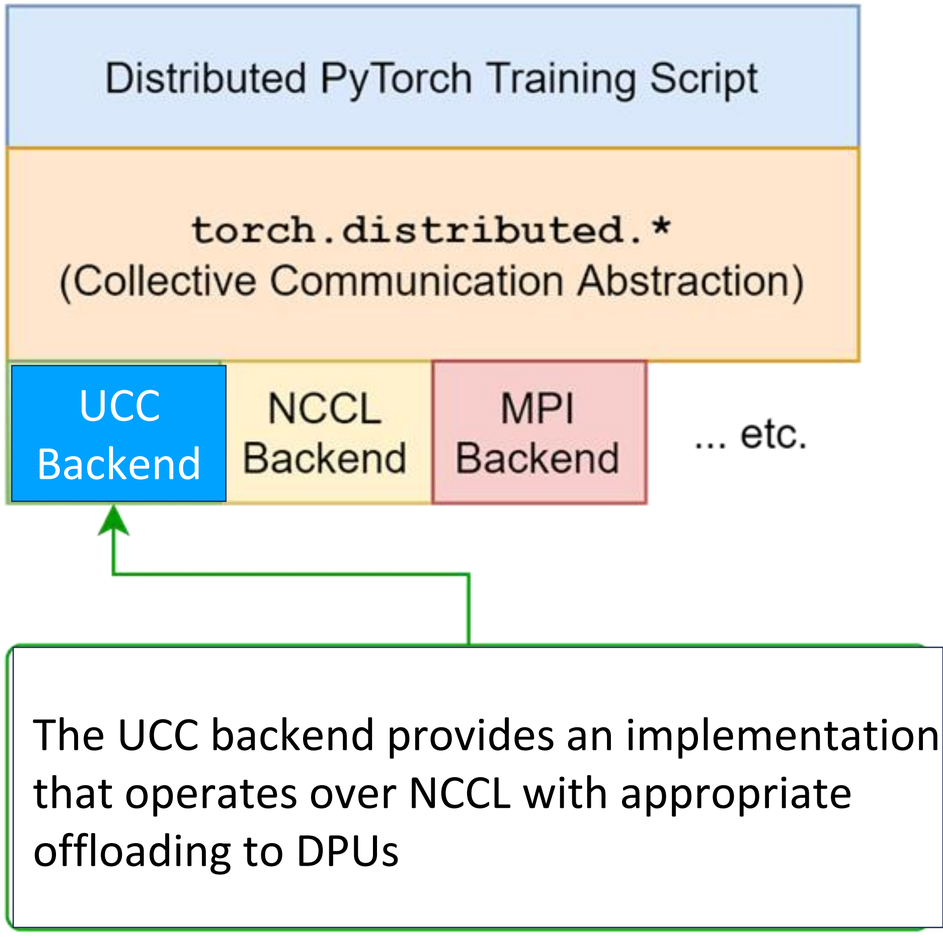
PyTorch and TensorFLow

- Current Pytorch solutions are based on NCCL and MPI
- UCC supports pytorch/TF
- Allows for the potential to offload collectives to DPU
- Improving collectives extremely useful for:
  - Scalable Model parallelism
  - Data parallelism, In-layer, pipeline parallelism
  - Eliminate noise introduced by GPUs executing collectives
- UCC can be used to have a unified API to measure performance of collectives on multiple devices



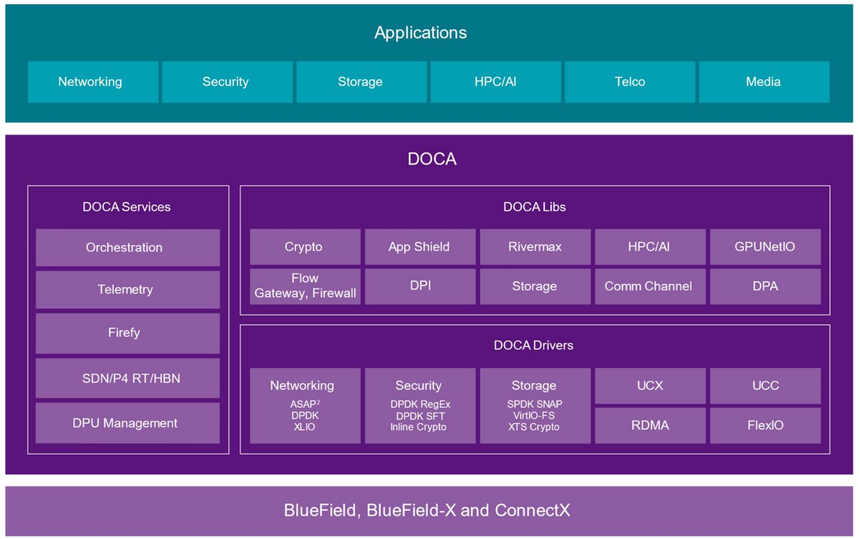
Explicit parallelism: communication backend for torch distributed collectives

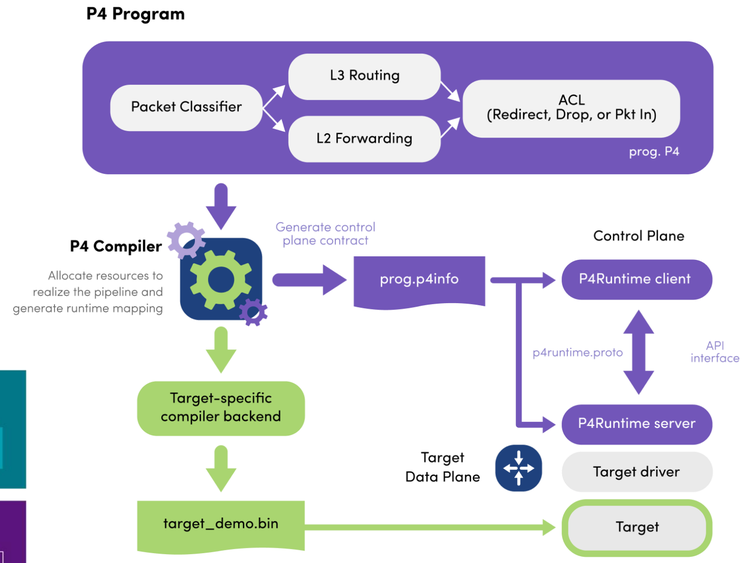




Other Frameworks for DPUs

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





Learn more about P4 and DOCA at:

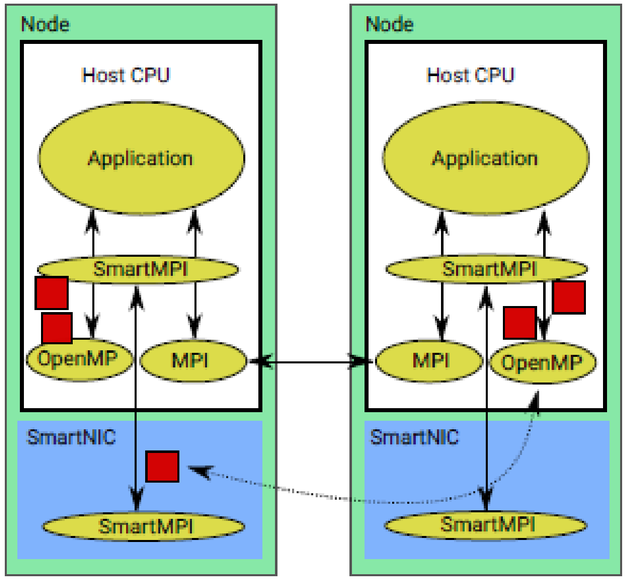
P4 - https://p4.org

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

Future directions in programming models

Future Programming Models

- Kokkos, OpenMP Remote Offload
- PGAS
- Task-based runtimes
- Python and data analytics
- AI Frameworks



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/*

Future Directions for DPU Research

- Application use-cases
- Task-based runtime
- Monitoring and performance tools
- Acceleration storage
- Edge computing
- AI and Machine learning evaluations
- Secured data in-motion and at rest

Transforming applications with MPI and SHMEM

Leveraging SmartNICs

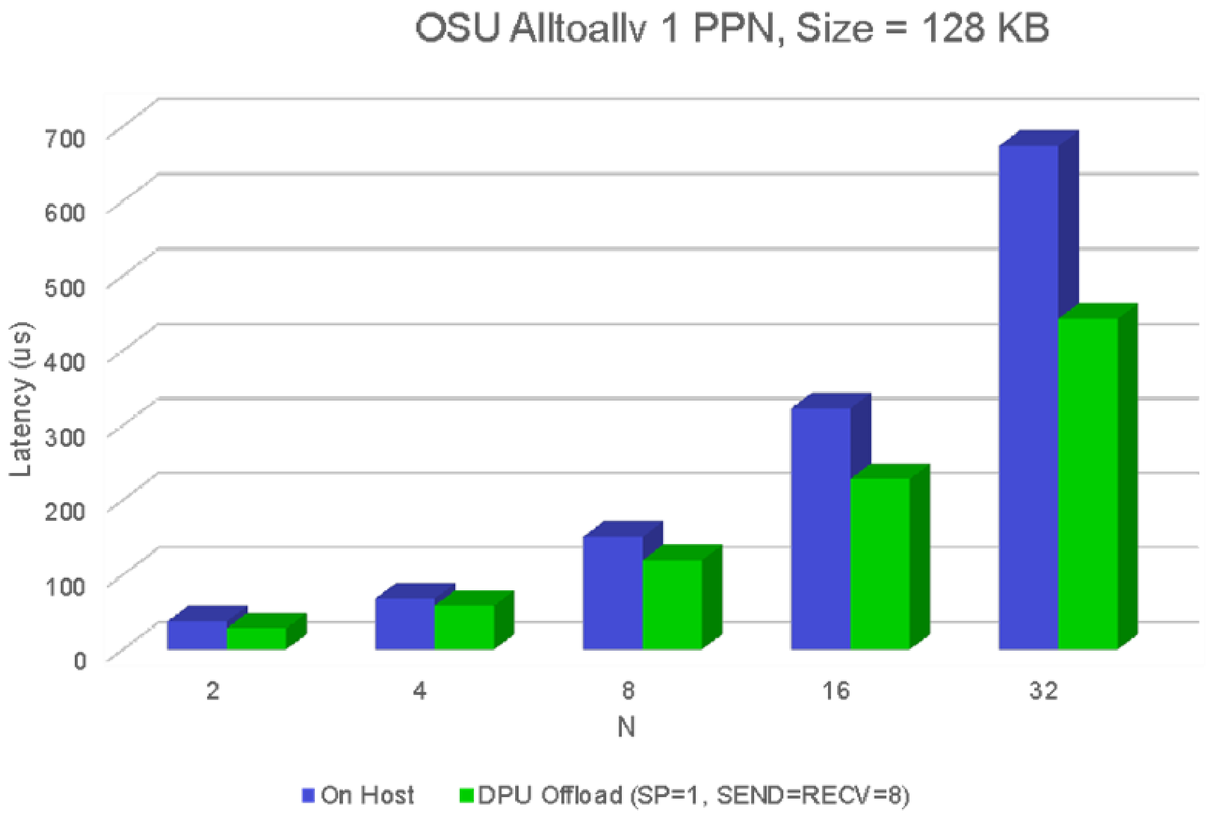
- Run a compiled application against a communication library
  - No application changes
  - Communication library treats the SmartNIC as it would any other communication medium
- Modify the application use communication routines that leverage the SmartNIC's capabilities
  - Example: Change the application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

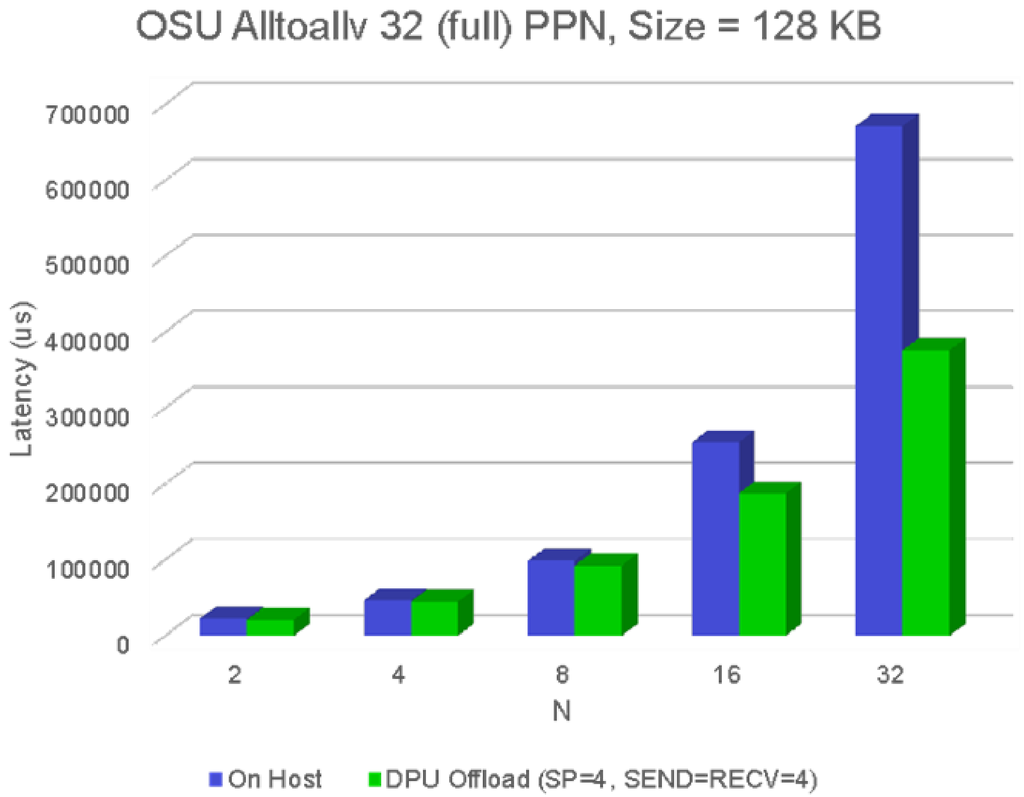
High-level Programming Models

Open MPI DPU support

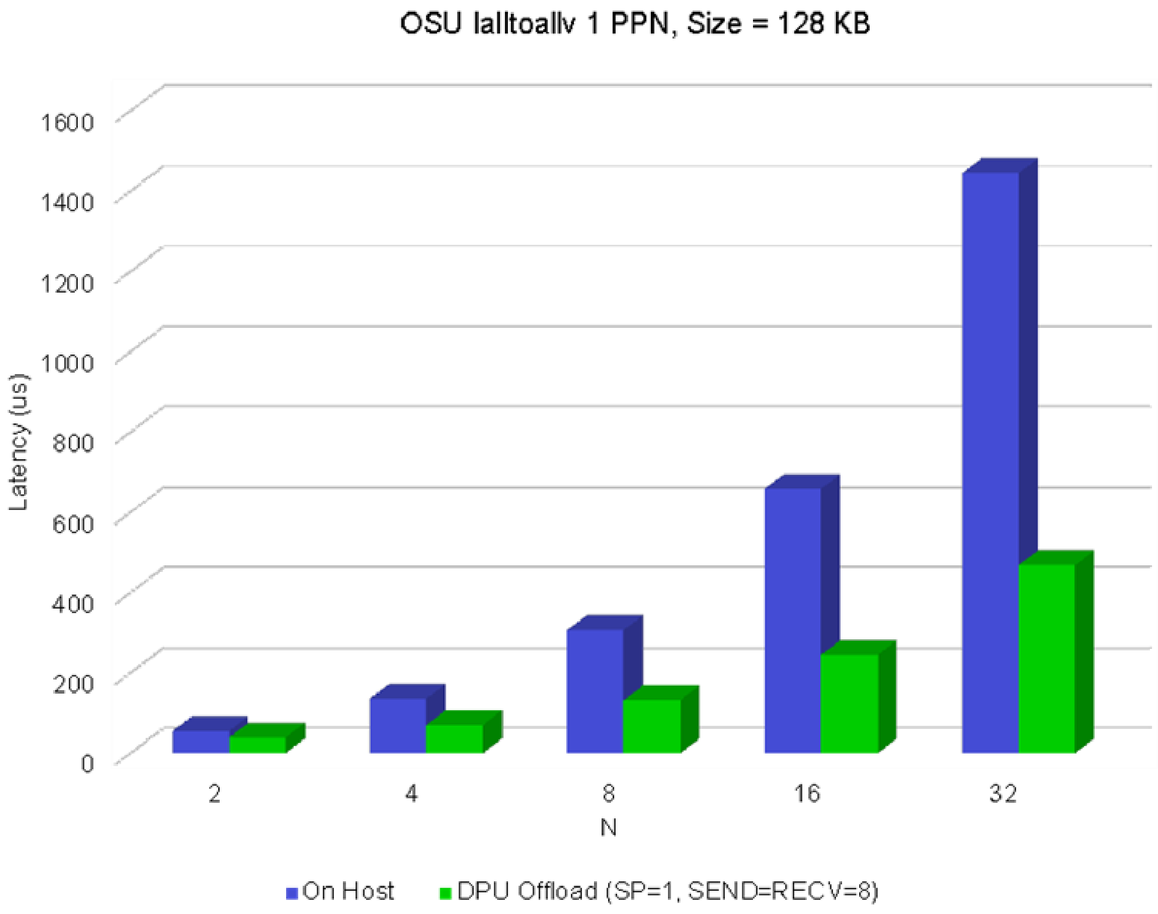
- Collectives supported: Alltoall, Alltoallv, Allgather, Allgatherv
- Enabled via mpirun (see hands on)
  - To select appropriate algorithm that will run on the DPUs
  - Works with Open MPI versions that supports Using Unified Communication Collectives (UCC)
- No application changes are needed

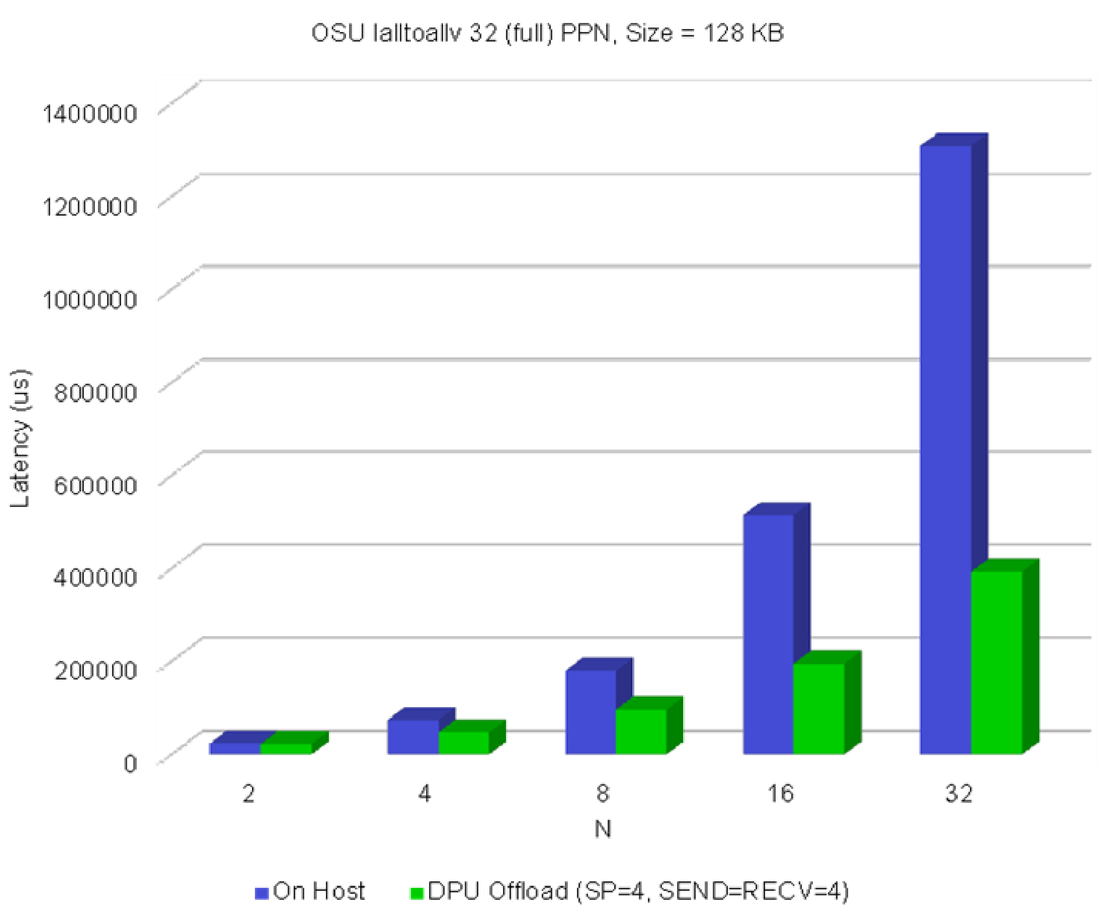
Alltoallv Latency



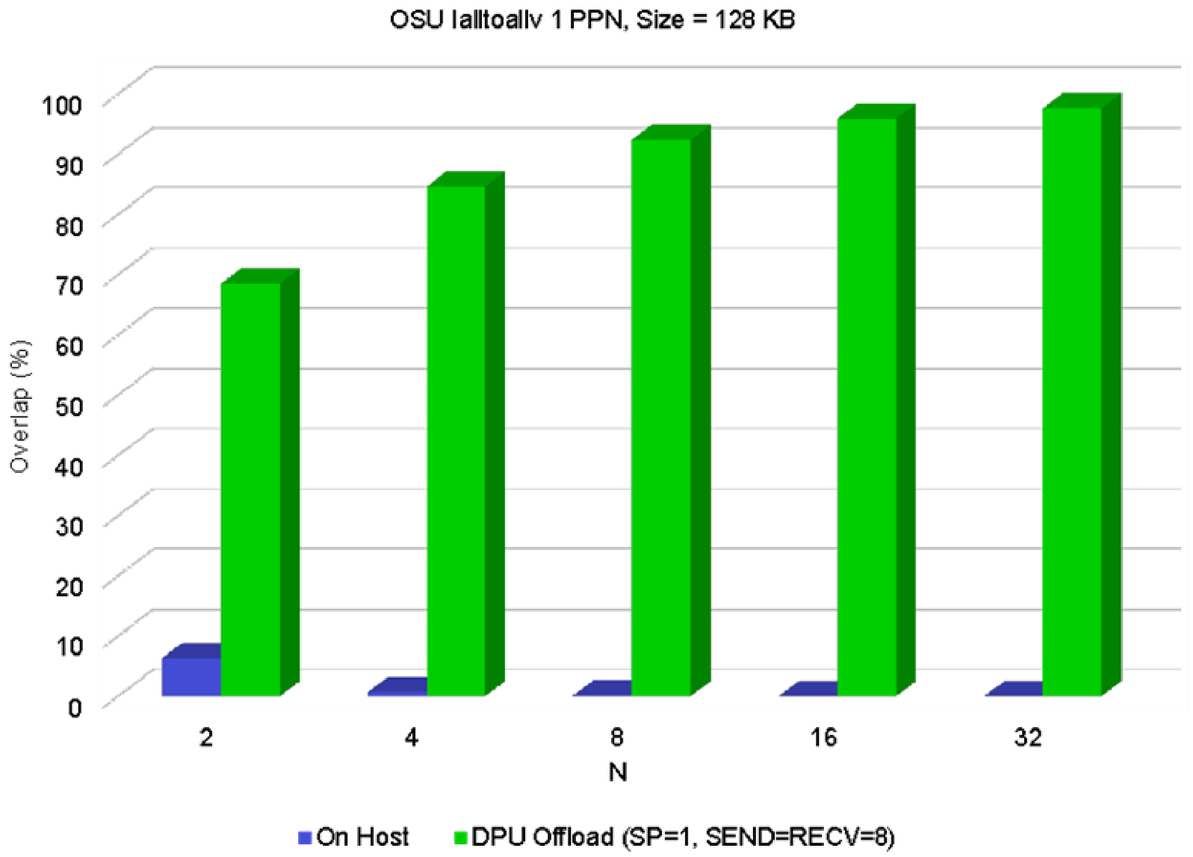


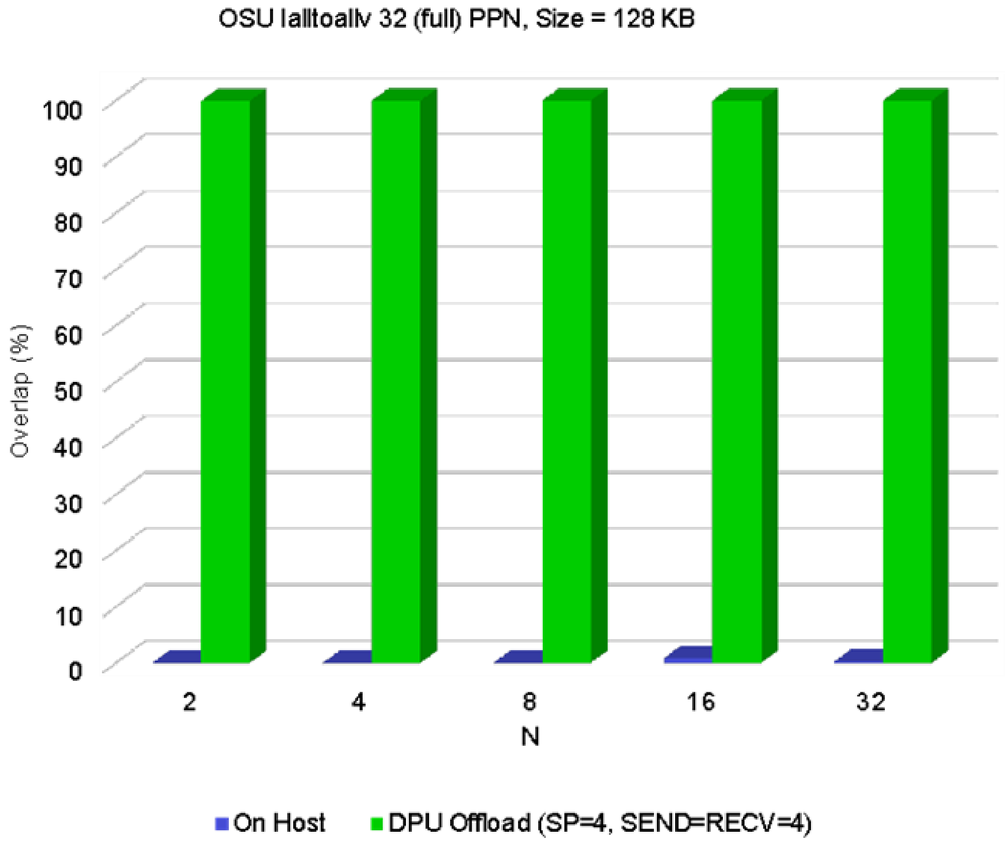
iAlltoallv Latency





iAlltoallv Computation/Communication Overlap



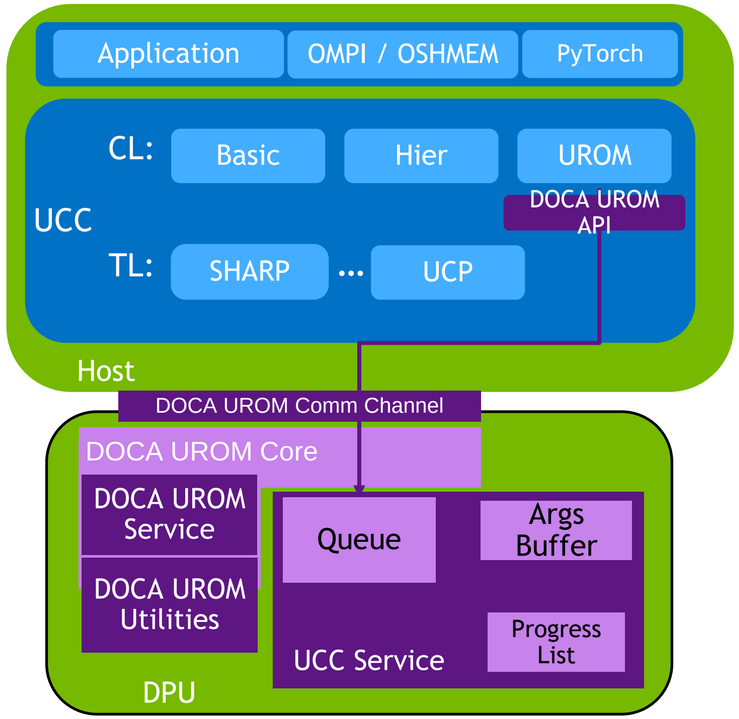


DPU UCC Offload with one-sided collectives

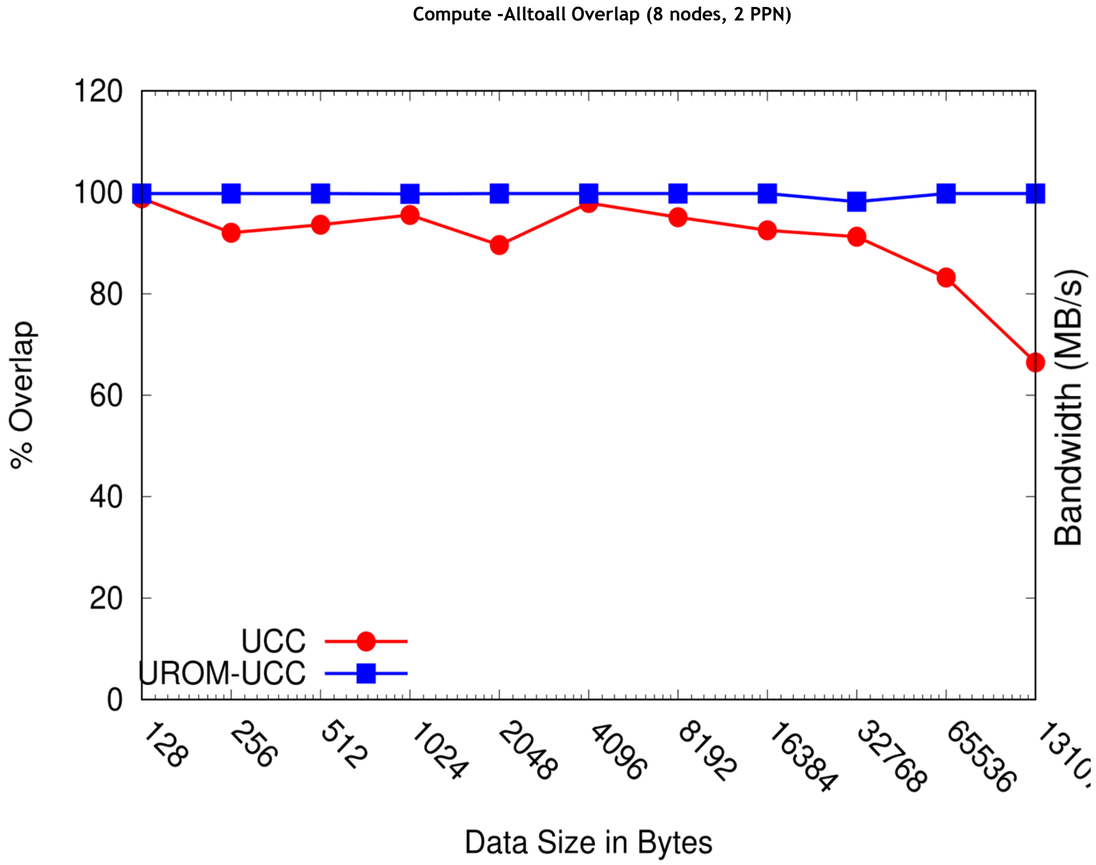
- Complete Offload of UCC library to DPU
  - Offloaded Collective Progress
    - Capable of 100% overlap
  - Capable of smart collective progress to minimize congestion
    - Leverage network telemetry and make informed decisions
- Enabled by UCC UROM CL
  - Execution of unmodified applications (e.g.,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

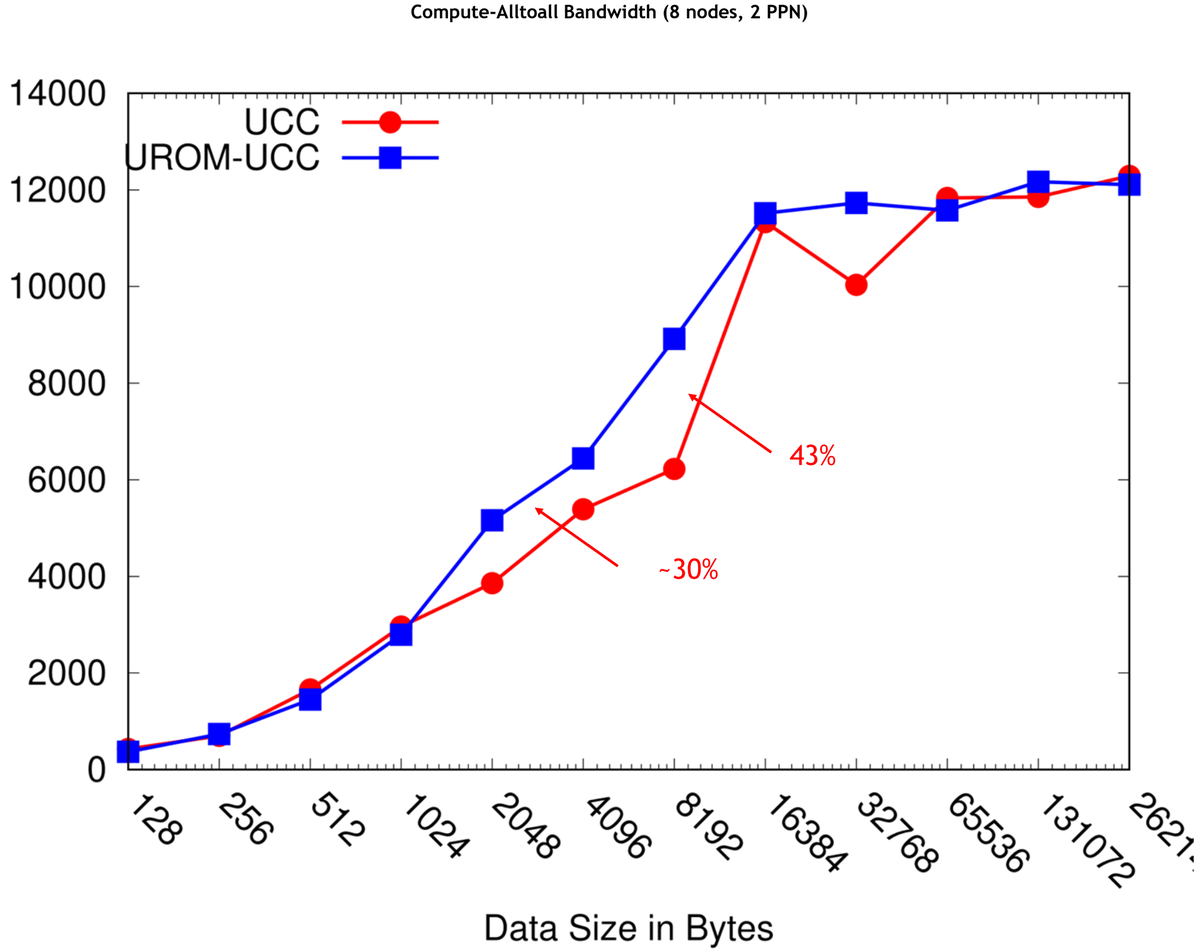
```
oshrun -np 4 -mca scoll_ucc_enable 1 --mca scoll_ucc_priority 100 --mca scoll_ucc_cls urom,basic ./my_ucc_app
```

- 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



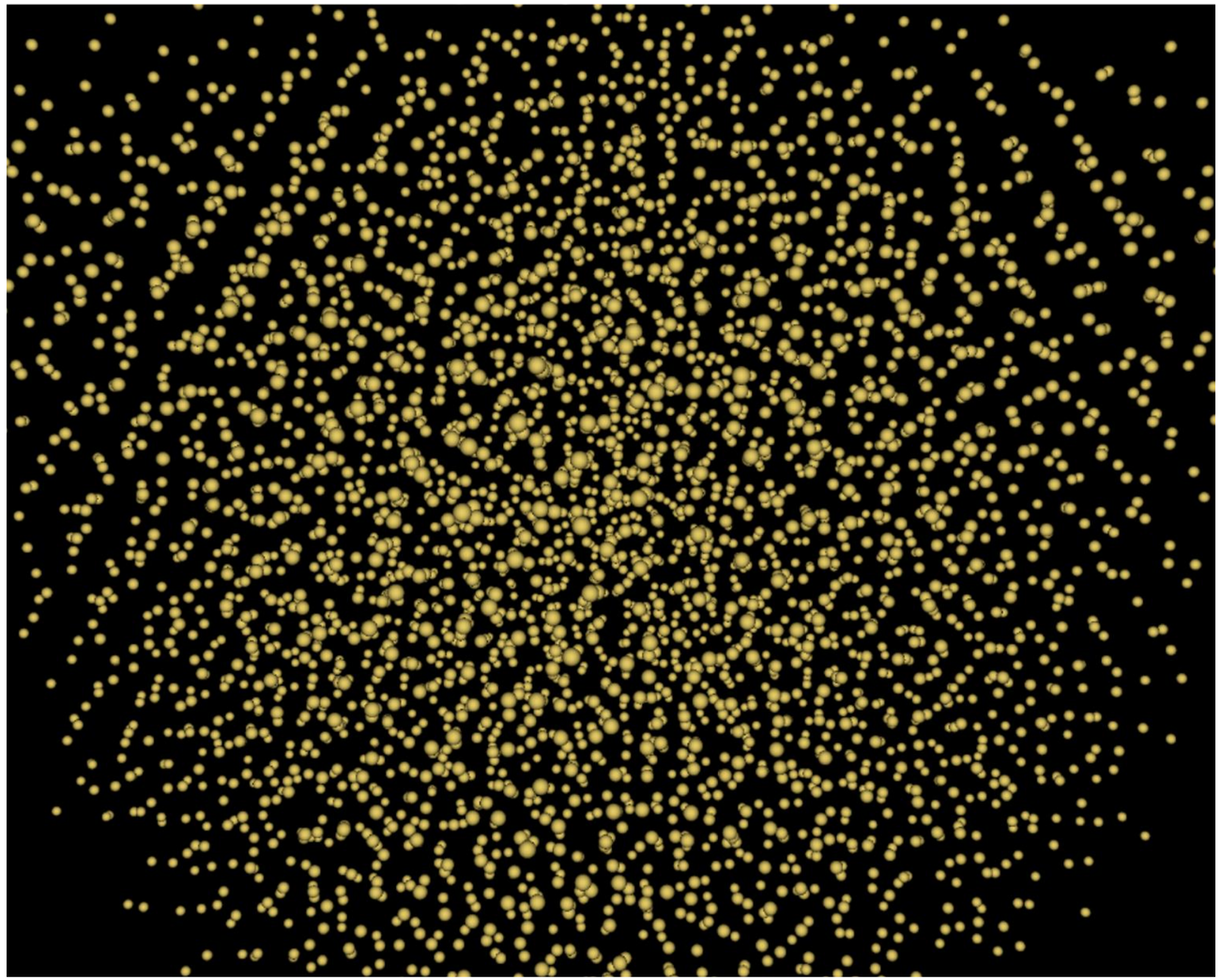
PERFORMANCE OF COMPUTE-ALLTOALL BENCHMARK





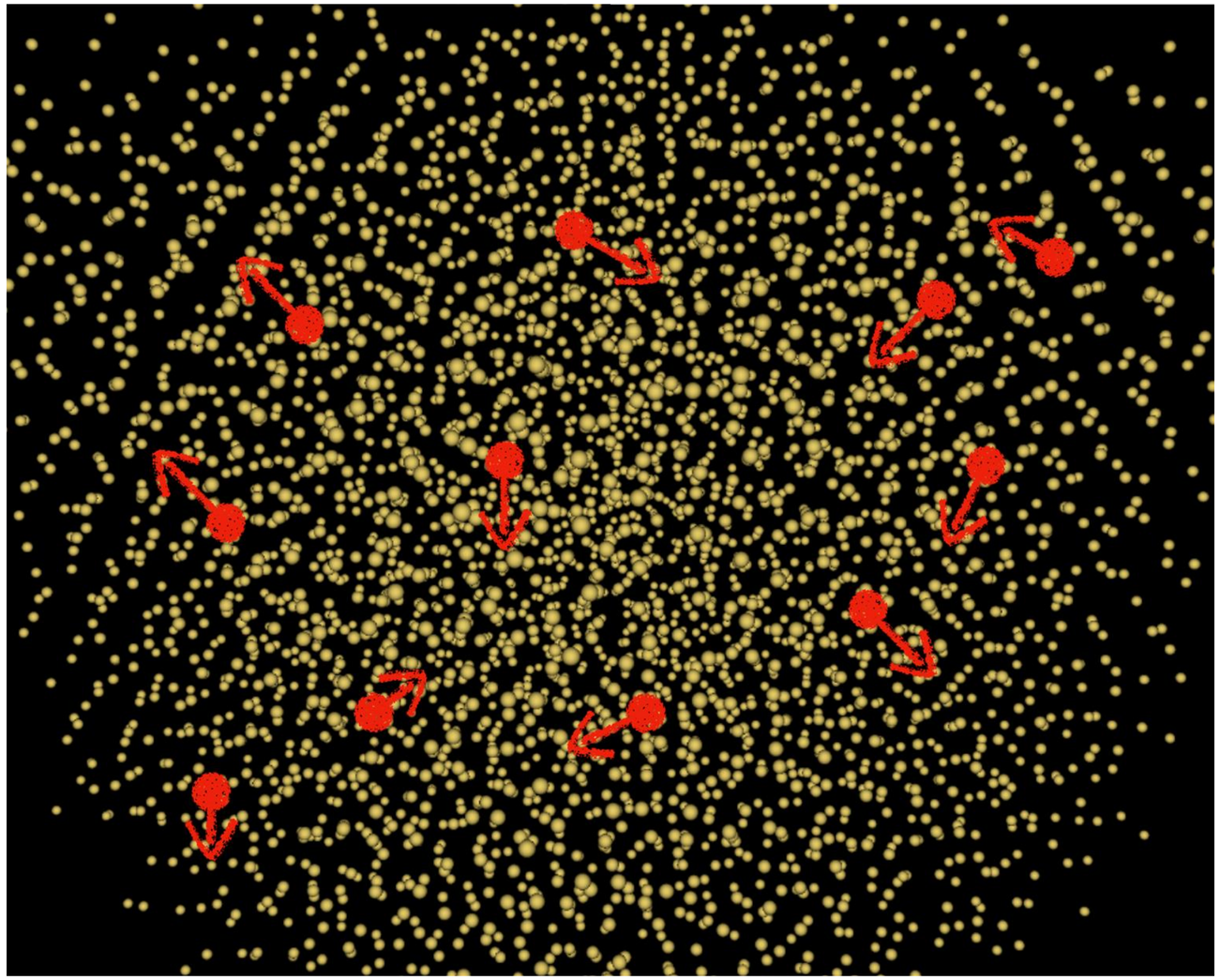
Heterogeneous Transformation with MPI: MiniMD case study

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



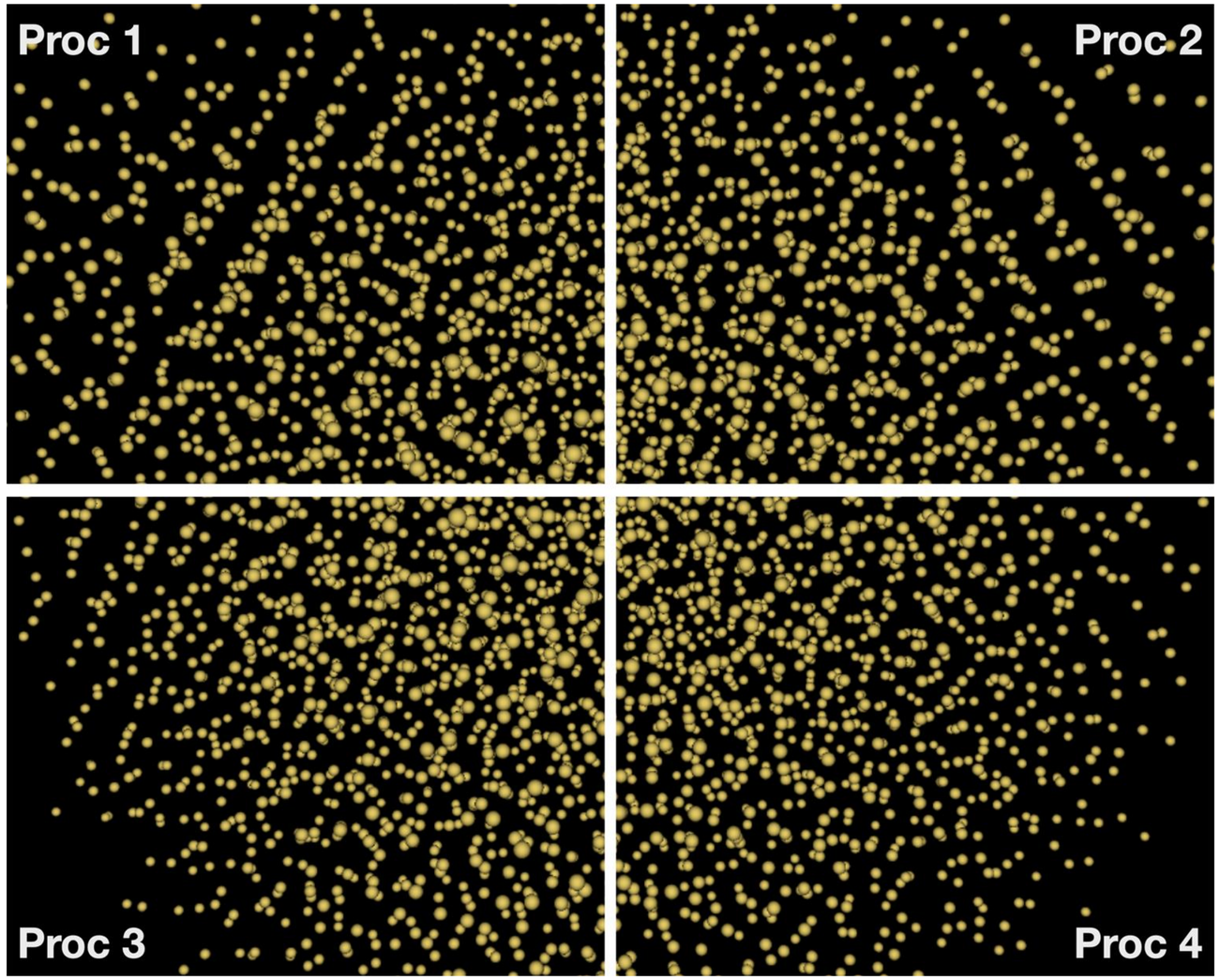
Baseline MiniMD

MiniMD is a molecular dynamics proxy app.



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).

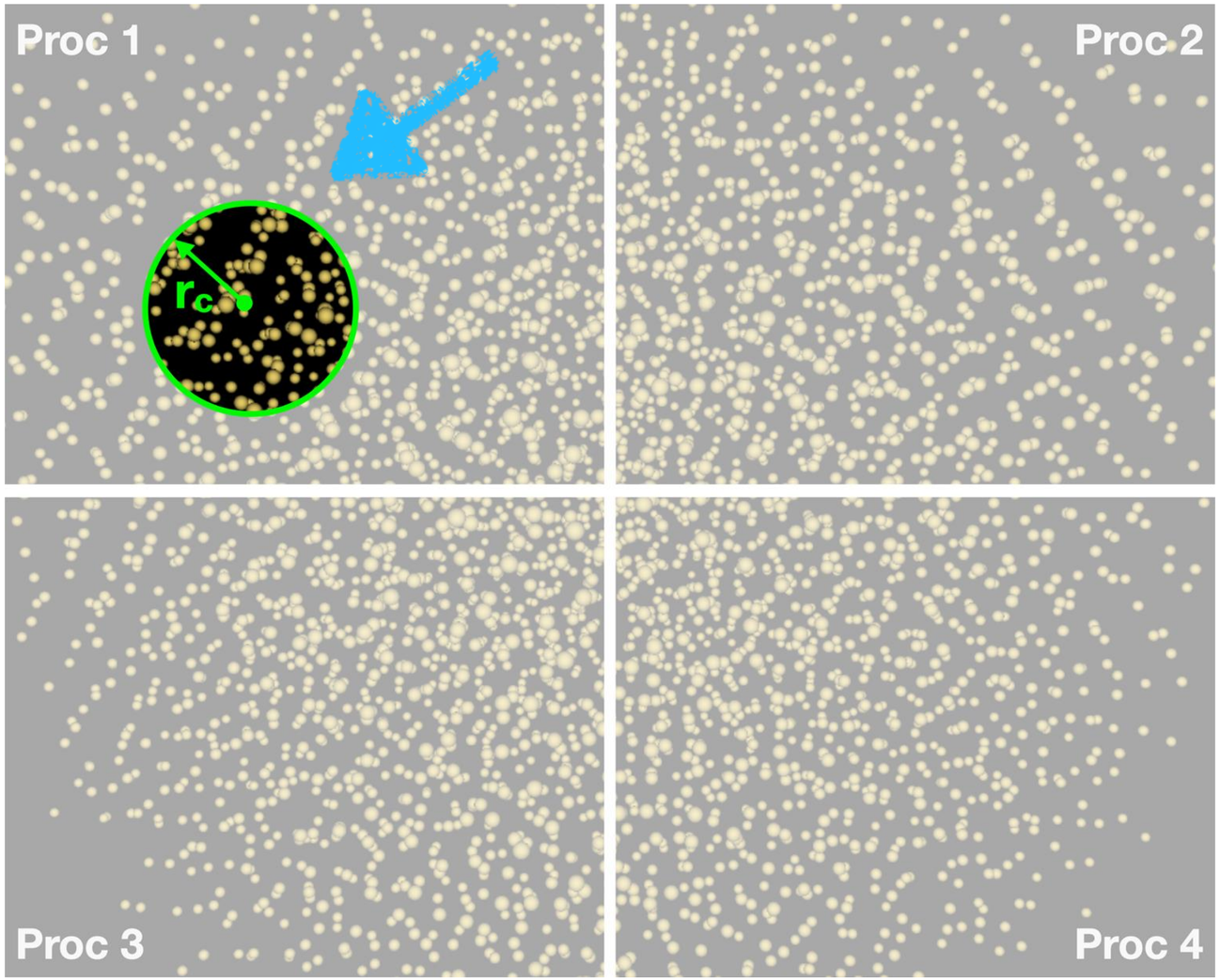


Baseline MiniMD

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

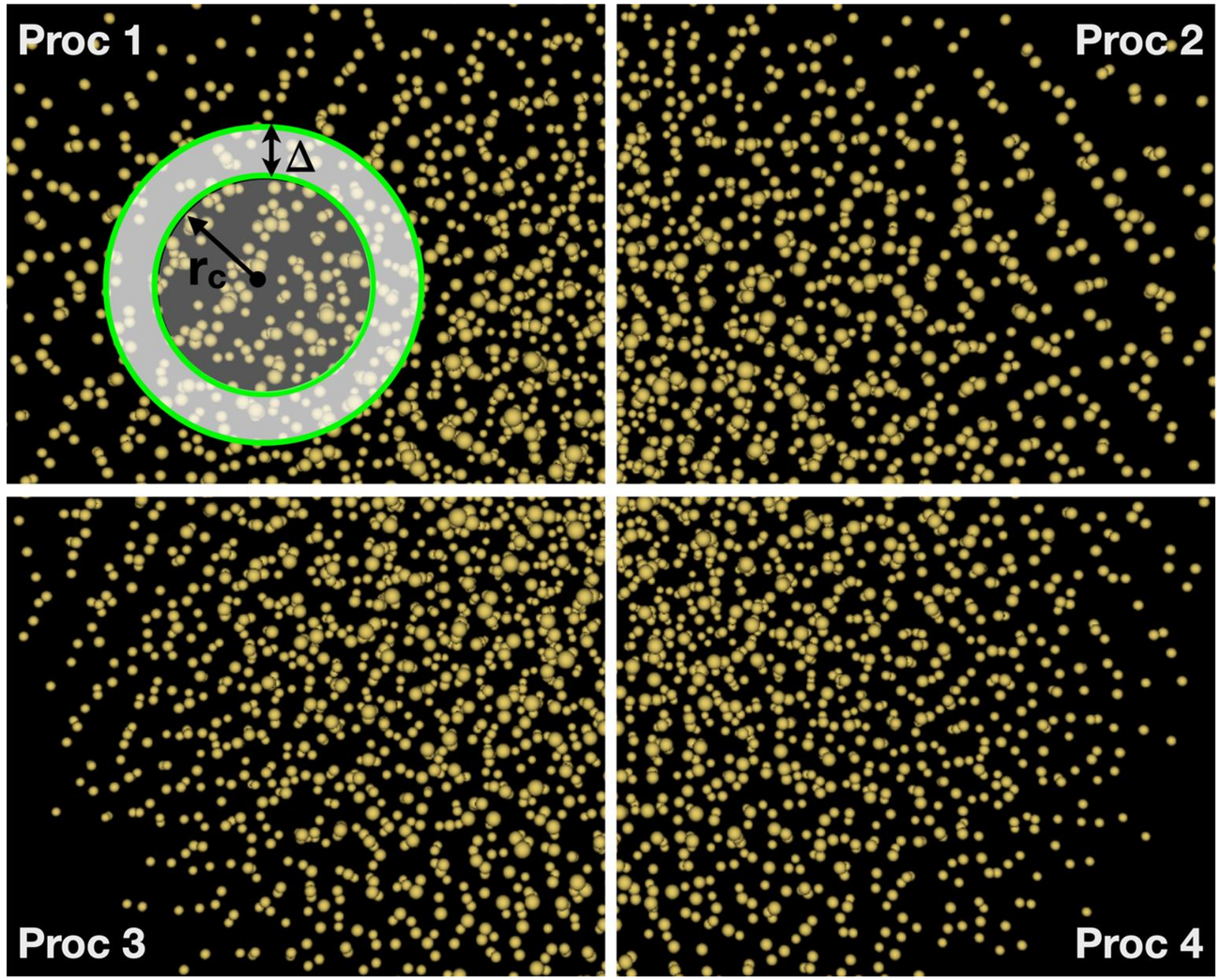
In the distributed-memory setting, the simulation domain is divided spatially among MPI processes.

Every process owns its particles, computes force on these particles and then updates the position and velocity of these particles.



Baseline MiniMD

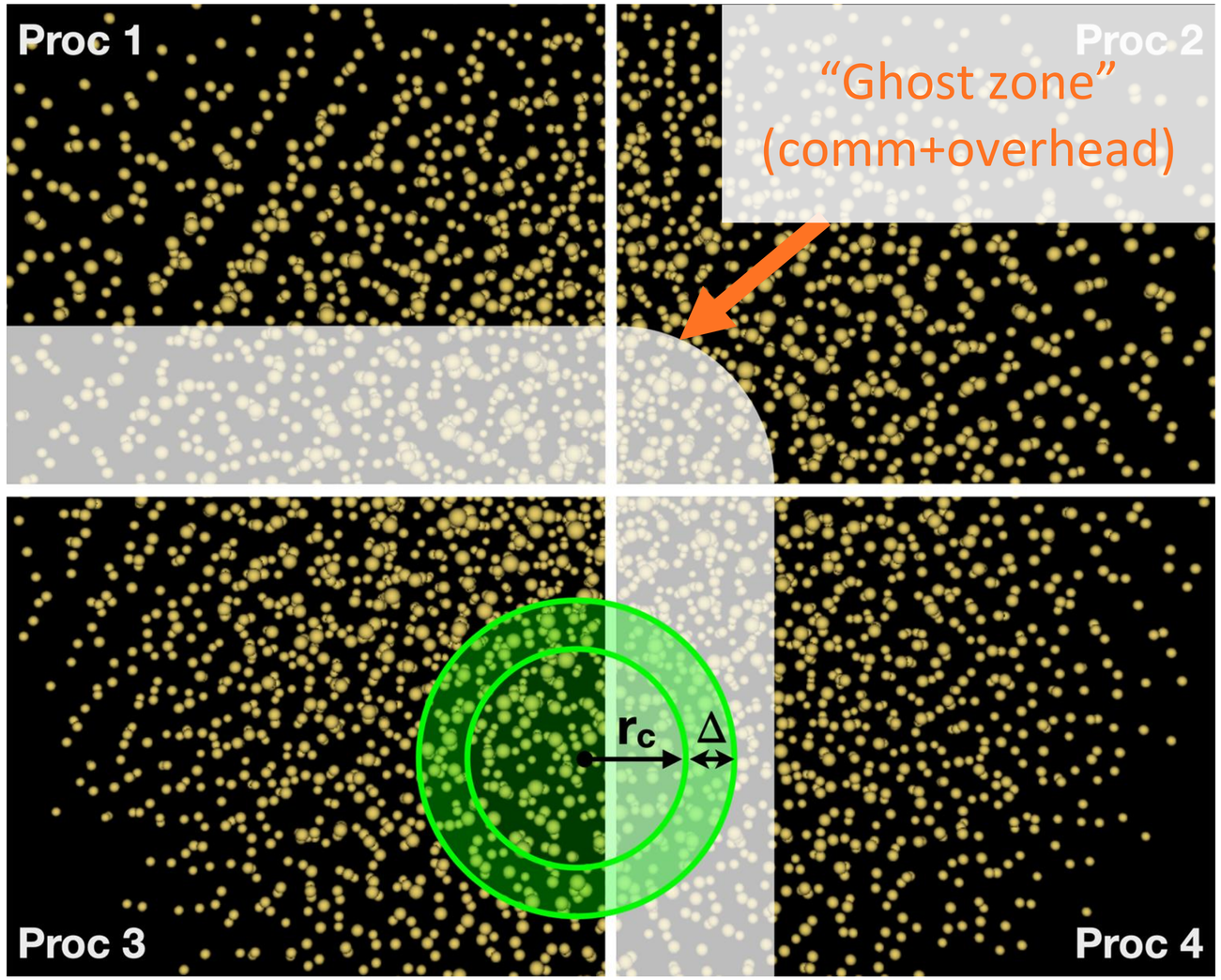
In each iteration, every particle interacts with others that lie within a some cutoff distance (r_c). A particle's neighbor list stores references to them.



Baseline MiniMD

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.

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).

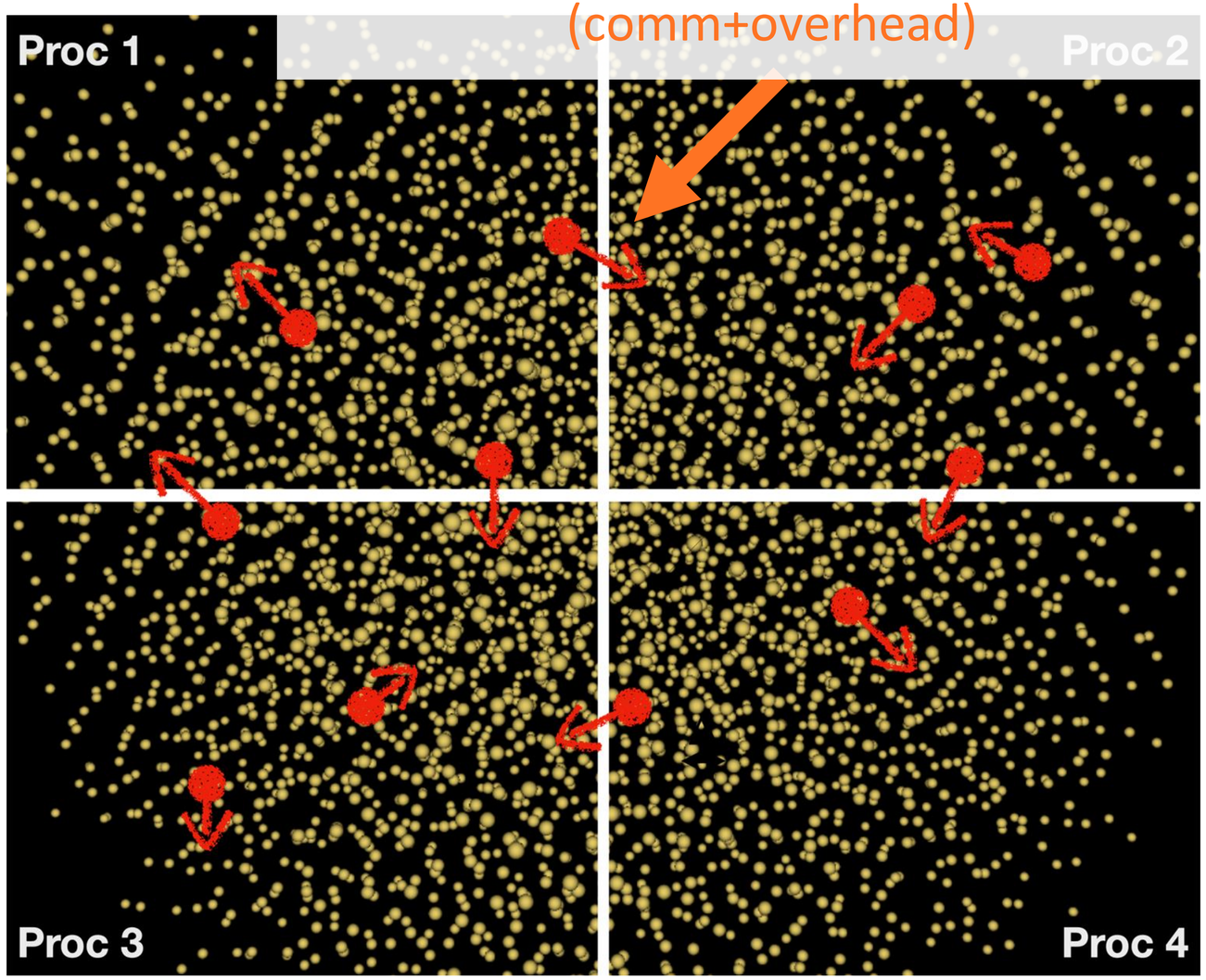


Baseline MiniMD

The cutoff distance (r_c) and skin thickness (Δ) imply the size of the interaction region just outside the boundaries of each process.

Each process keeps a copy of particles in that region.

Particles move through subdomains (comm+overhead)

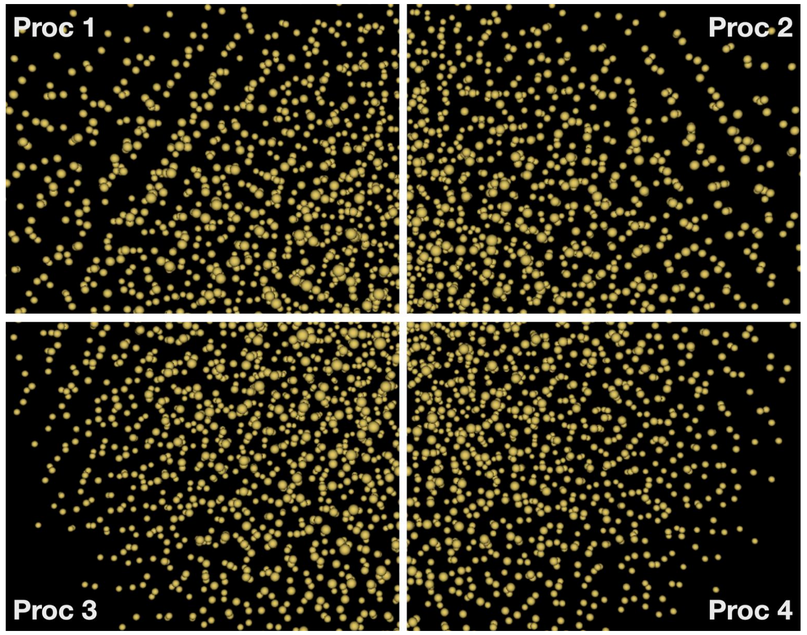


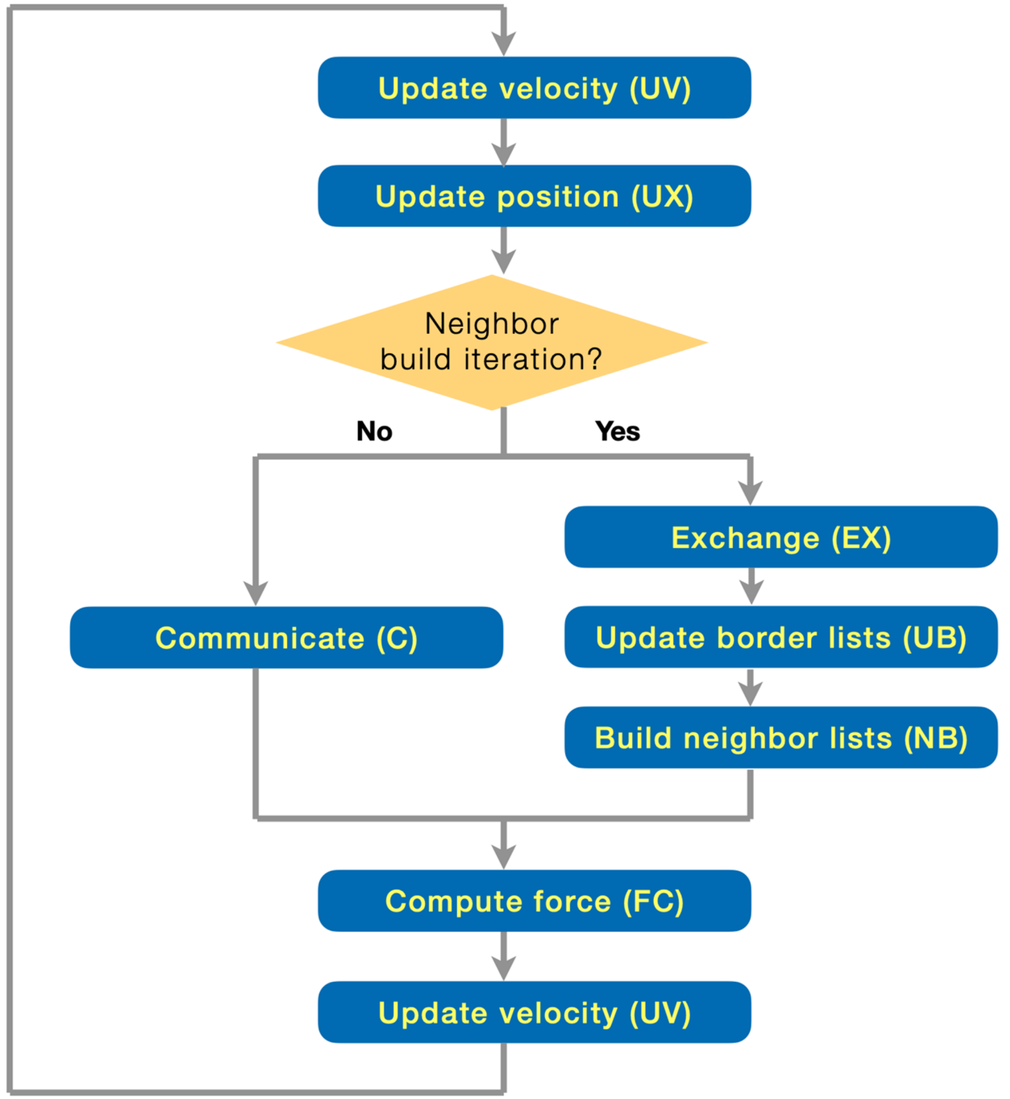
Baseline MiniMD

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).

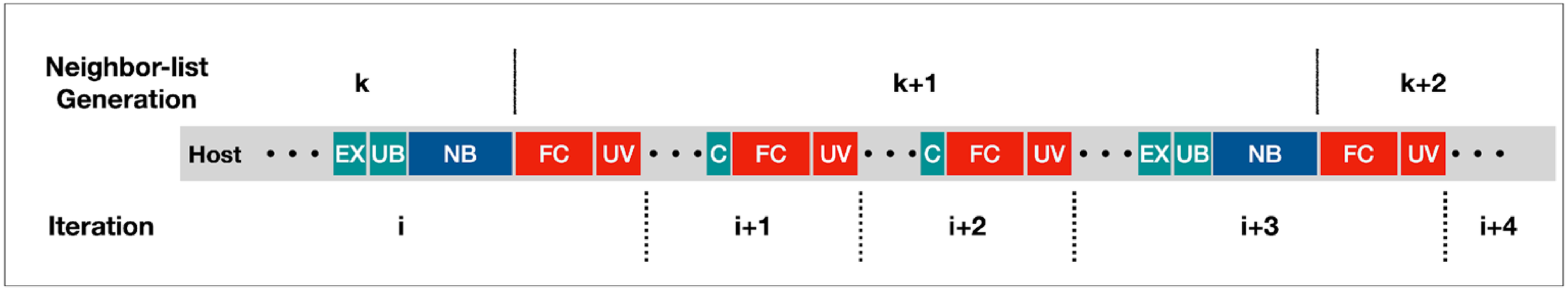
Baseline MiniMD



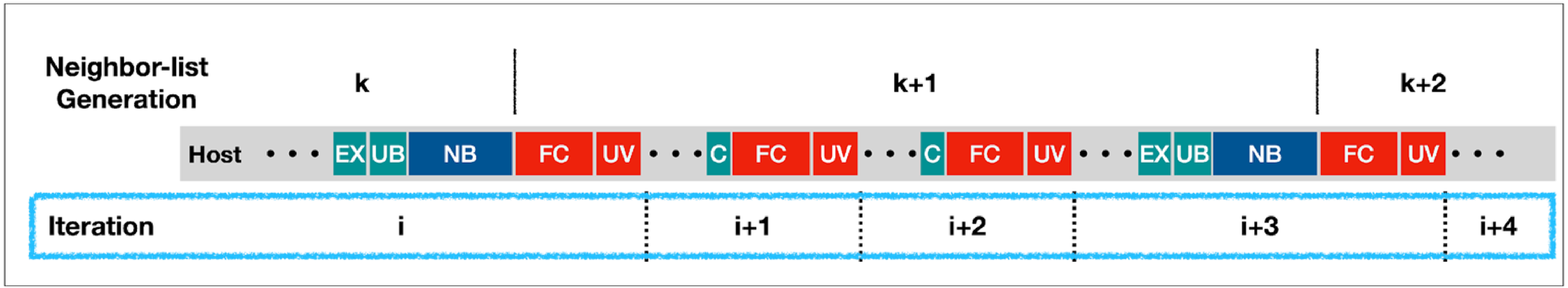


Each task is parallelizable but the sequence is sequential as shown by edges

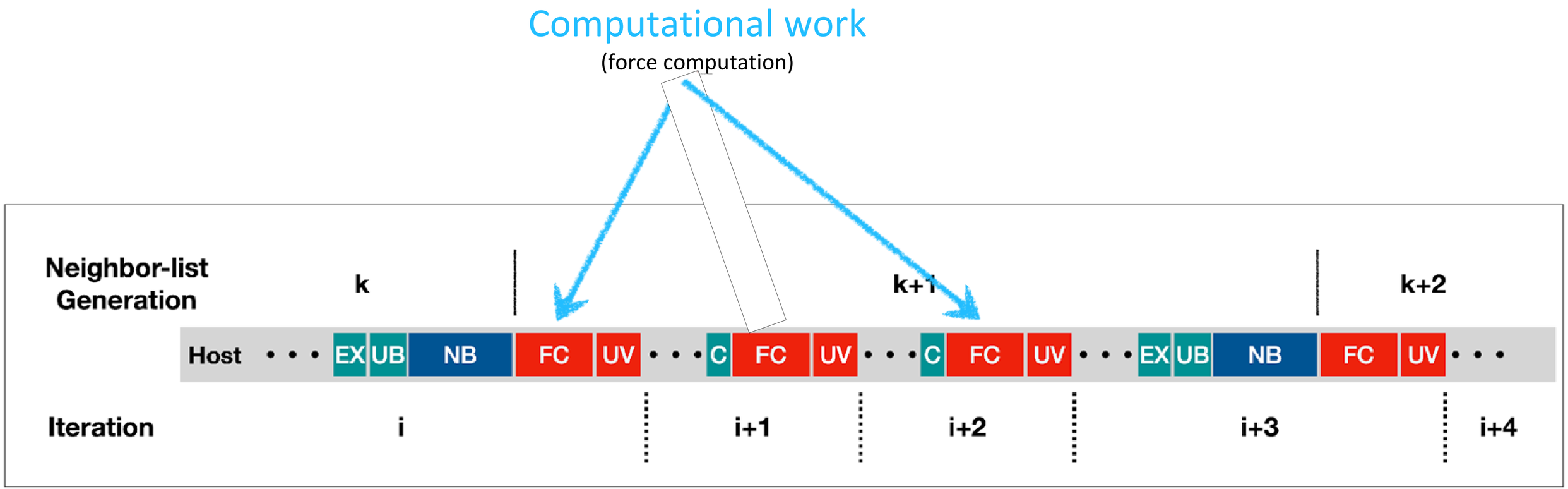
Serial dependencies



Serial dependencies

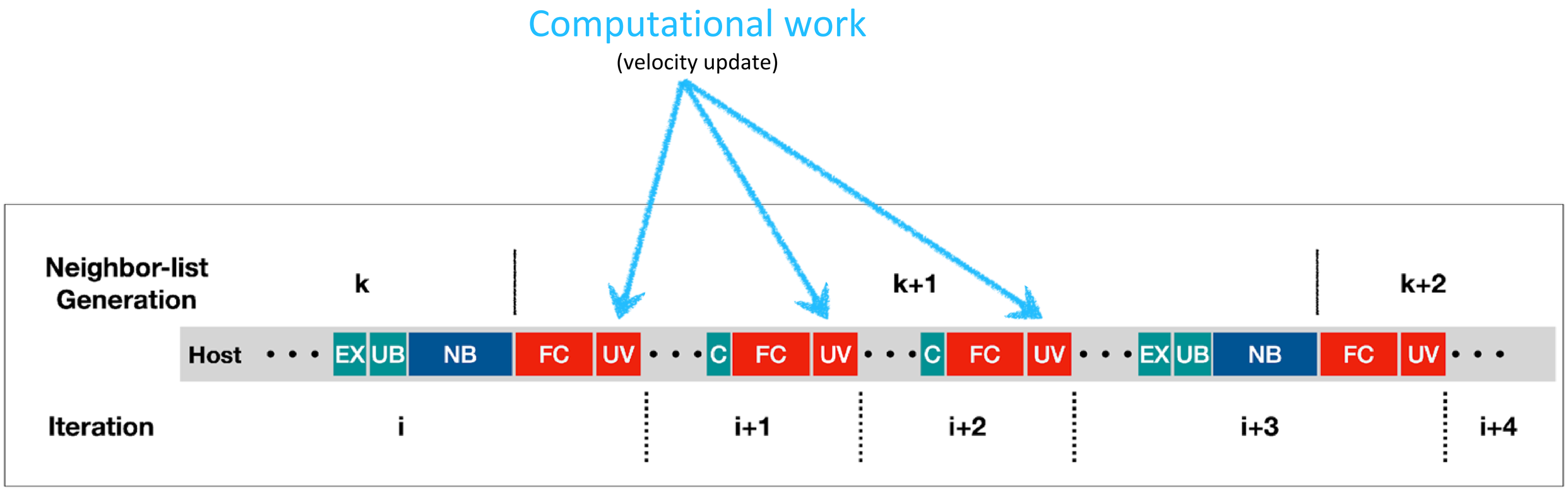


Serial dependencies



FC work: force computation

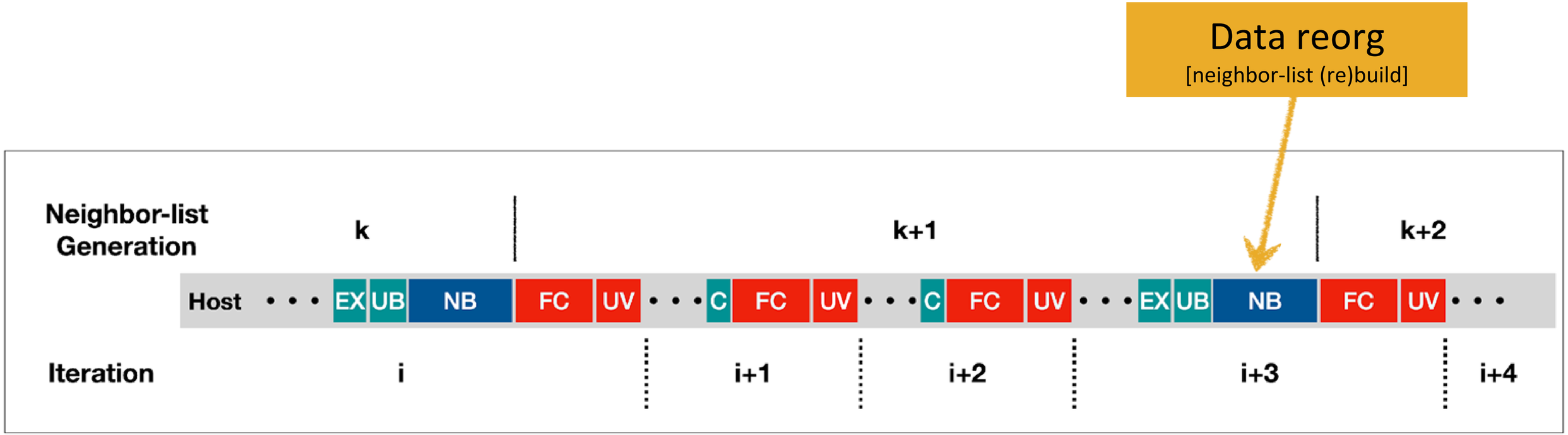
Serial dependencies



FC work: force computation

UV work: update velocity

Serial dependencies

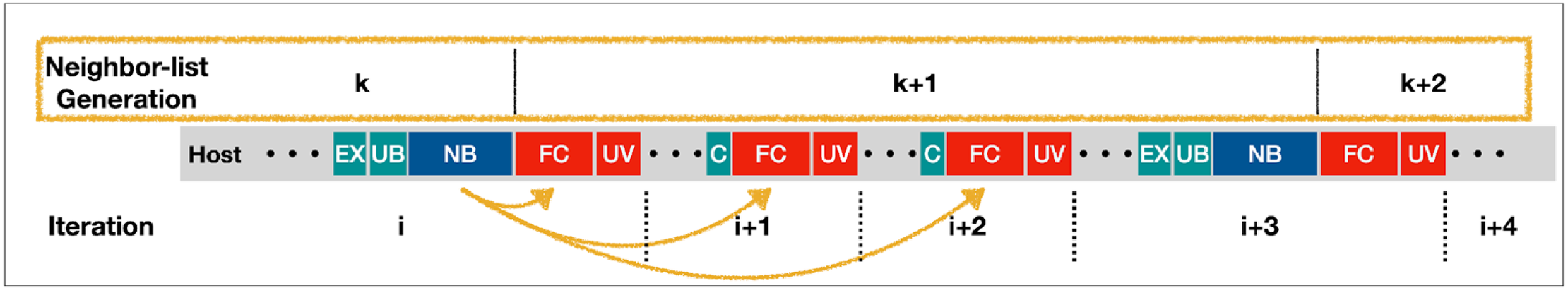


Serial dependencies

FC work: force computation

UV work: update velocity

NB neighbor-list rebuild

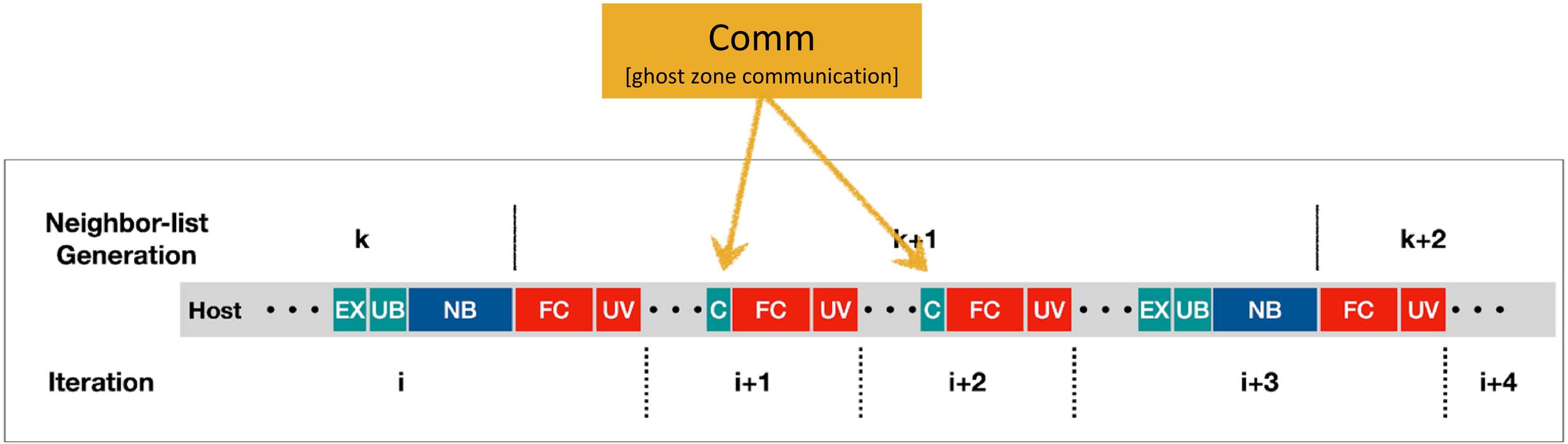


Serial dependencies

FC work: force computation

UV work: update velocity

NB neighbor-list rebuild



Serial dependencies

FC work: force computation

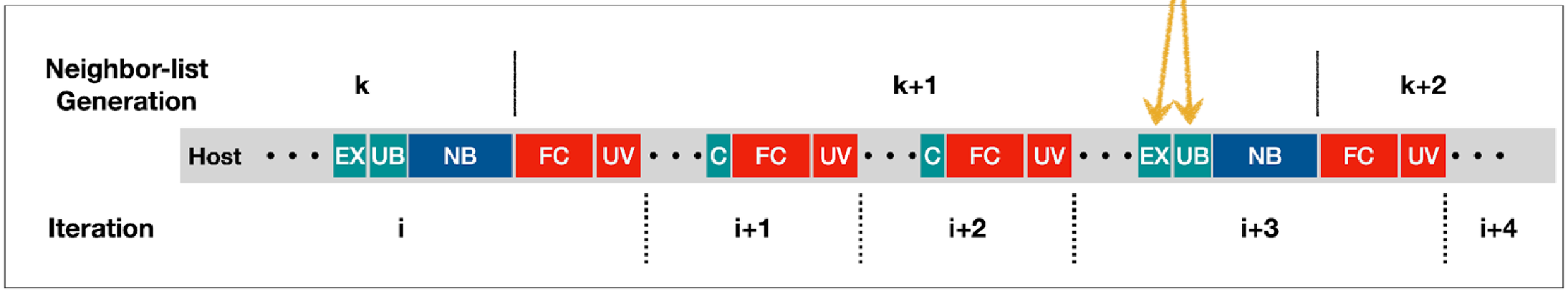
UV work: update velocity

NB neighbor-list rebuild

C communication

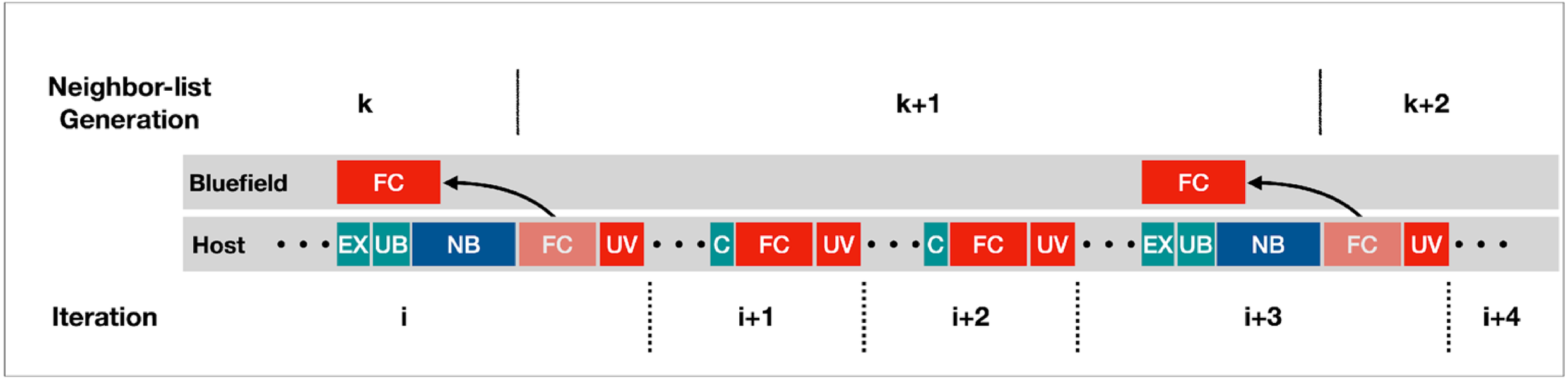
Data reorg + comm

[particles reallocation and border lists update]



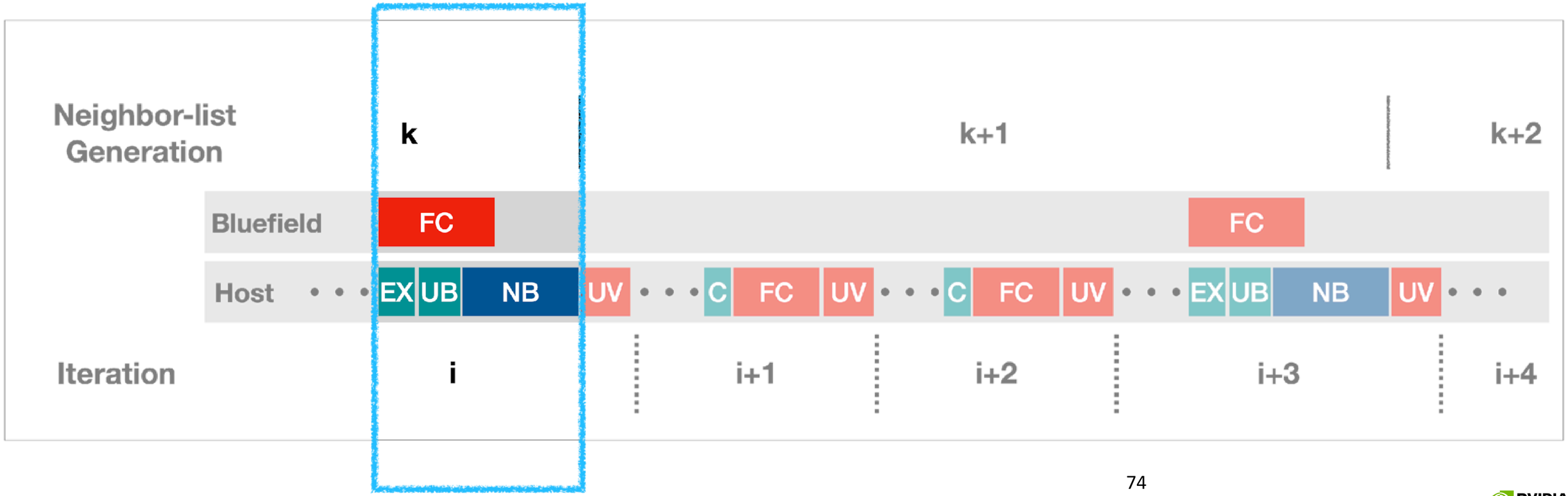
Breaking the dependencies
(“Off-path” algorithm)

FC	work: force computation	C	communication
UV	work: update velocity	UB	comm: update border lists
NB	neighbor-list rebuild	EX	comm: particles reallocation



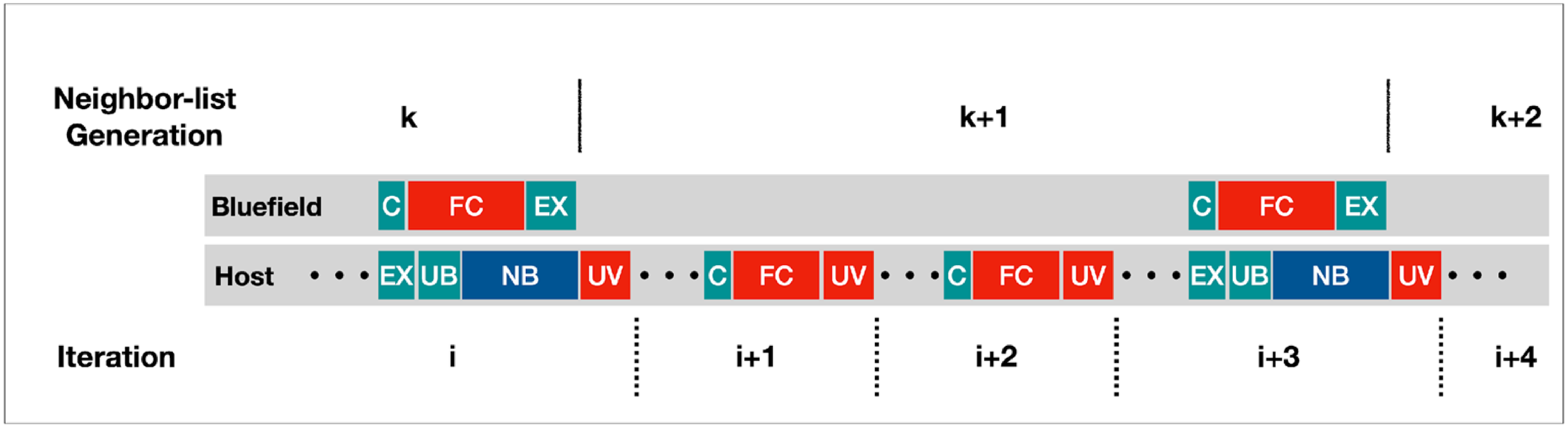
Breaking the dependencies
(“Off-path” algorithm)

FC work: force computation
UV work: update velocity
NB neighbor-list rebuild
C communication
UB comm: update border lists
EX comm: particles reallocation



Breaking the dependencies
(“Off-path” algorithm)

FC work: force computation
UV work: update velocity
NB neighbor-list rebuild
C communication
UB comm: update border lists
EX comm: particles reallocation



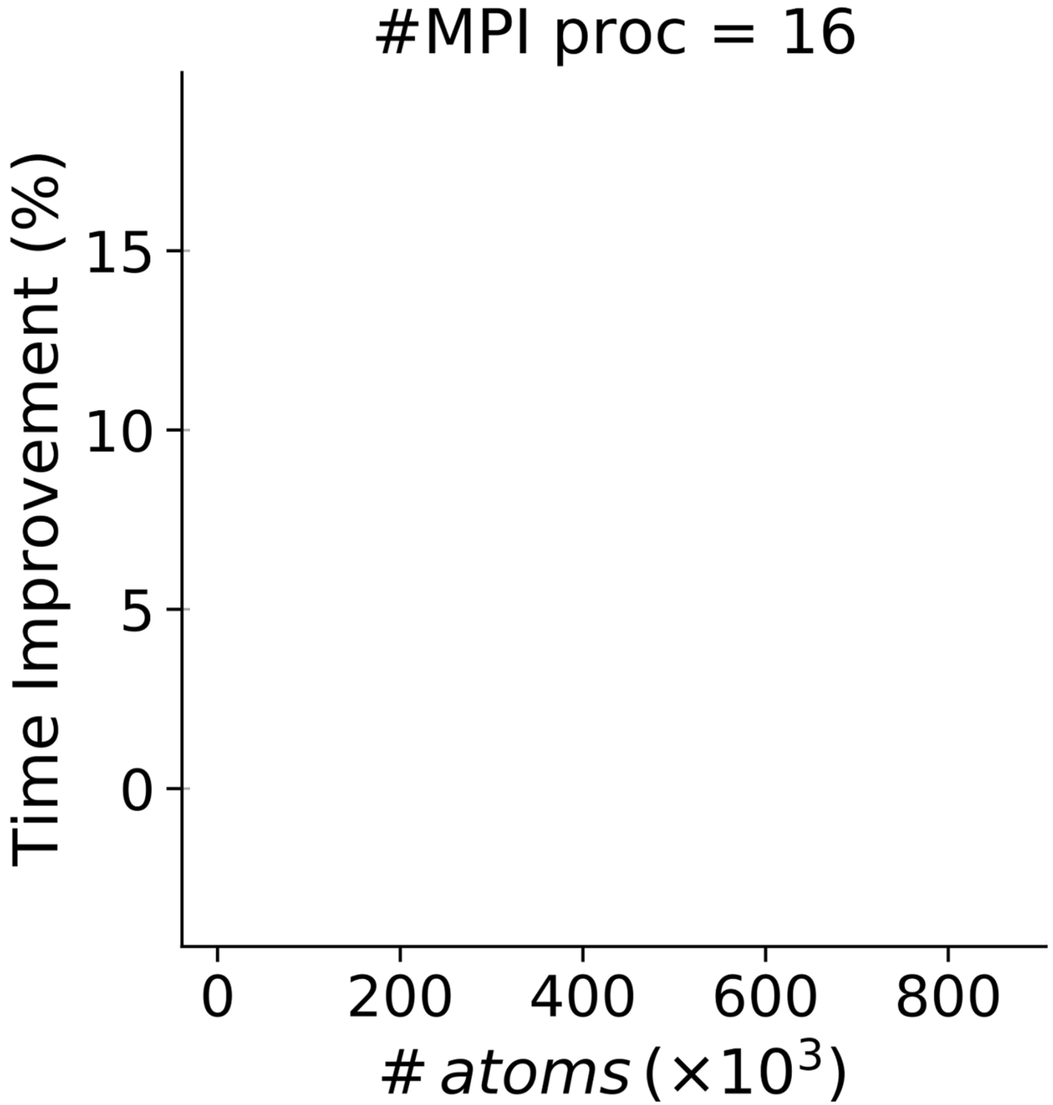
Baseline experiments

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

- 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)
- 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

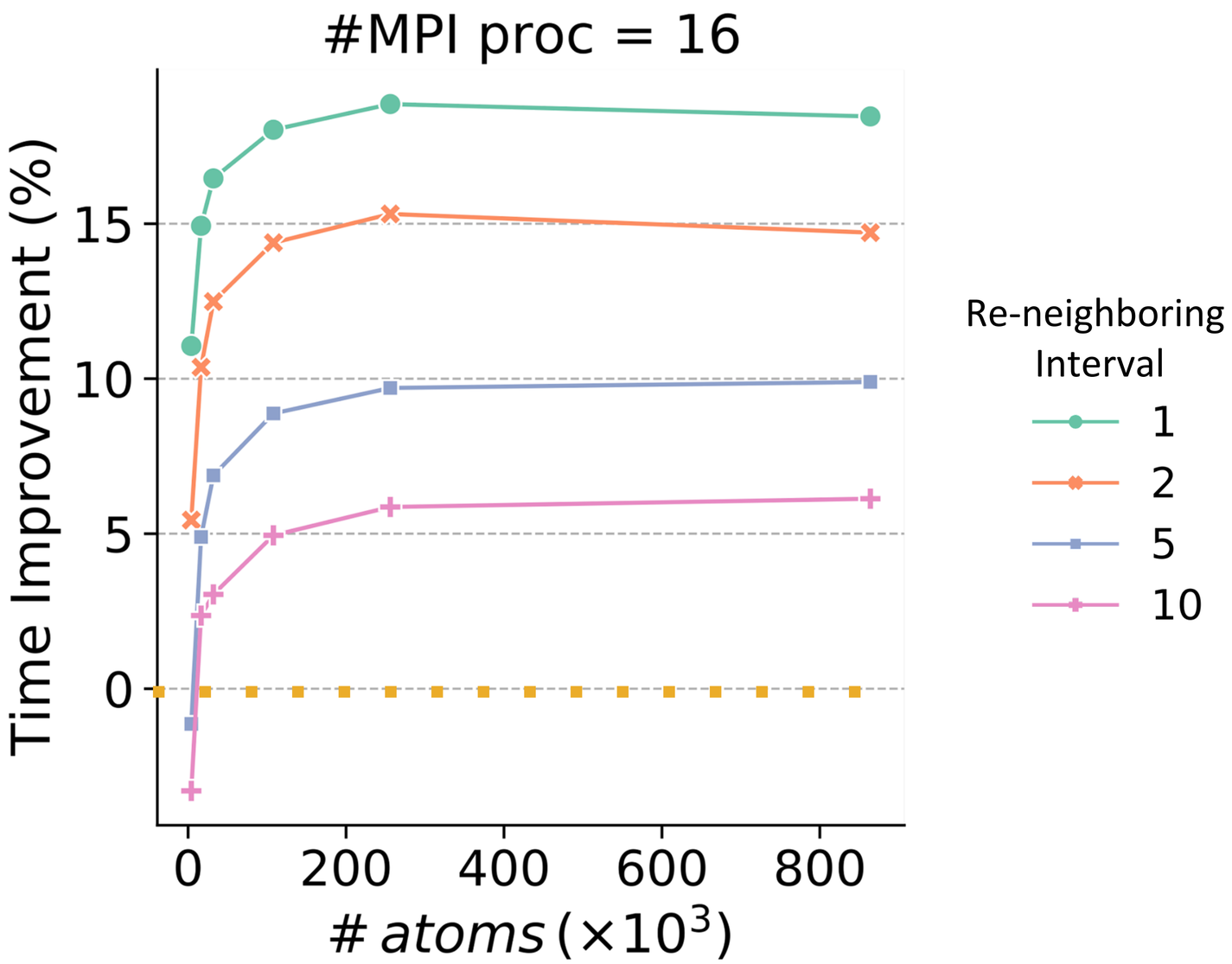
…



Restructured method is faster

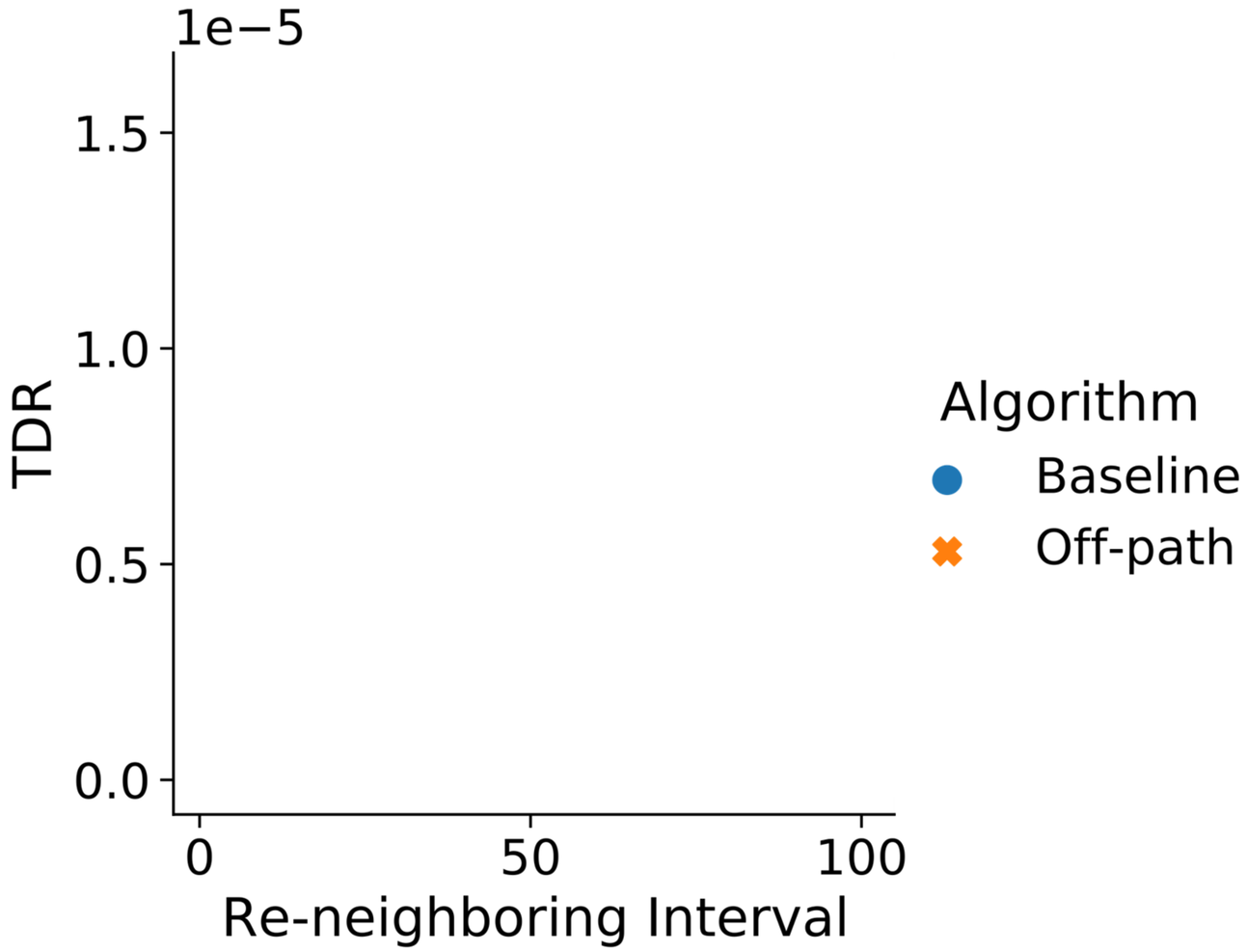
We observe small, but largely uniform, speedups of up to 20% compared to host-only execution with conventional NICs.

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%.



Restructured method is a viable simulation heuristic

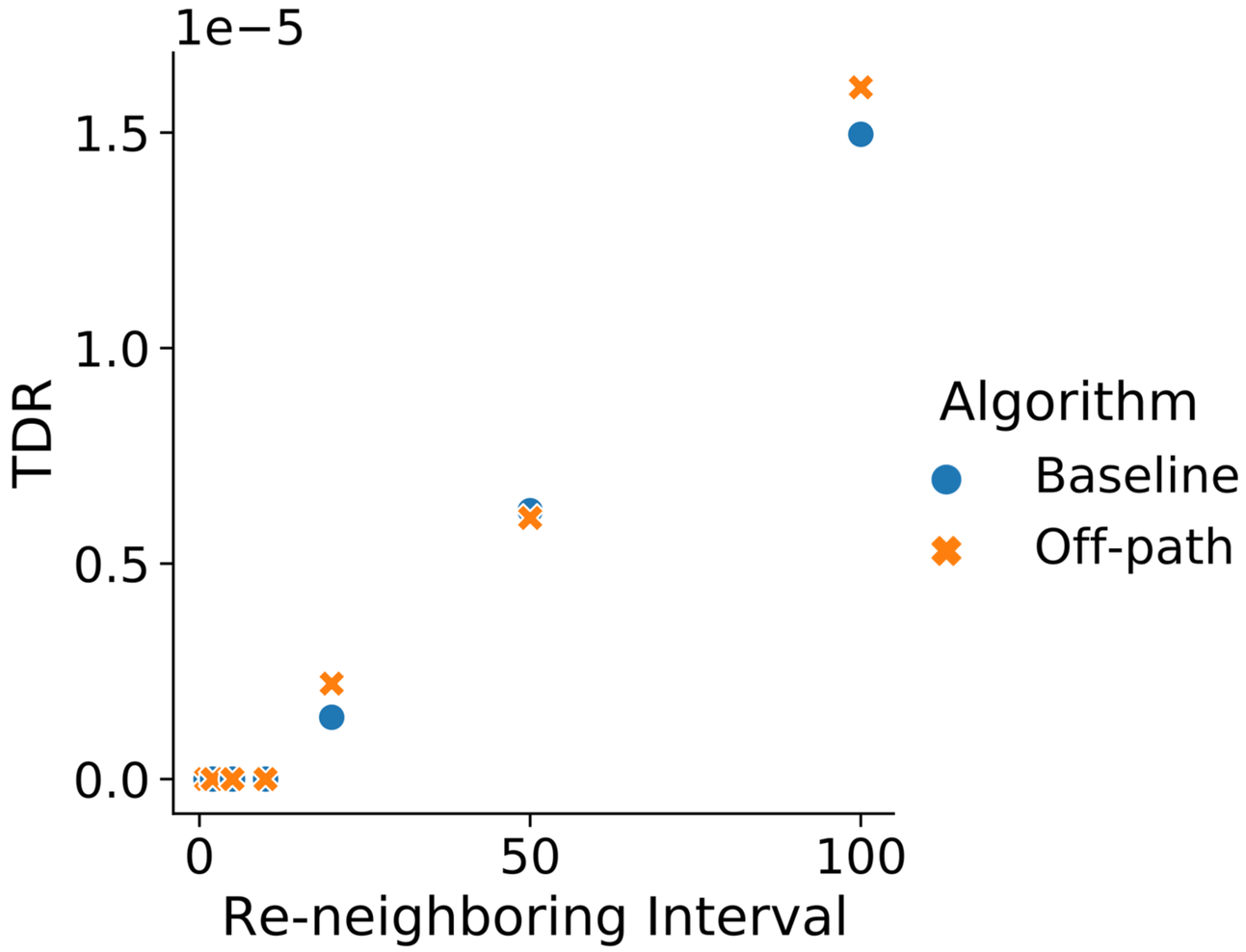
Temperature divergence rate (TDR): a proxy metric to assess the accuracy of our algorithm



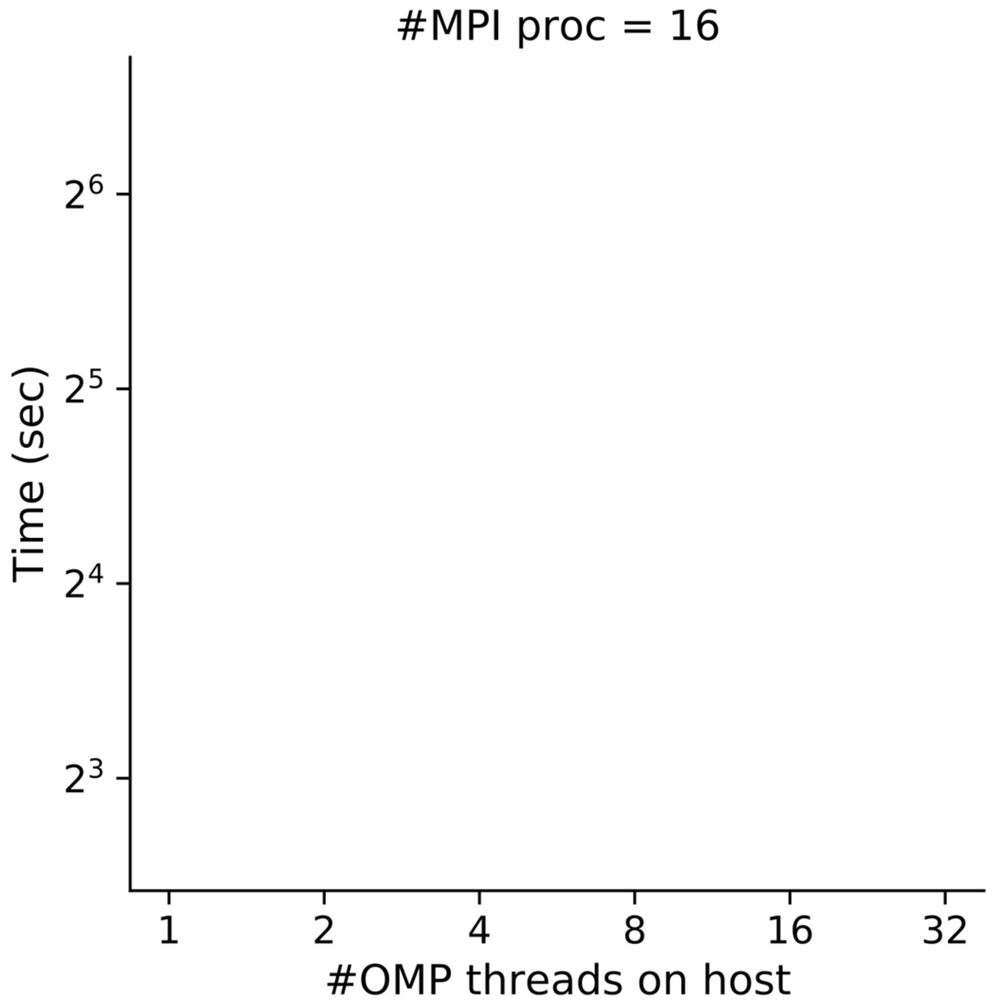
Restructured method is a viable simulation heuristic

Temperature divergence rate (TDR): a proxy metric to assess the accuracy of our algorithm

We also verified that the computed results of the restructured method are still within an acceptable level of accuracy.



Hybrid MPI/OpenMP performance results



Lower is better

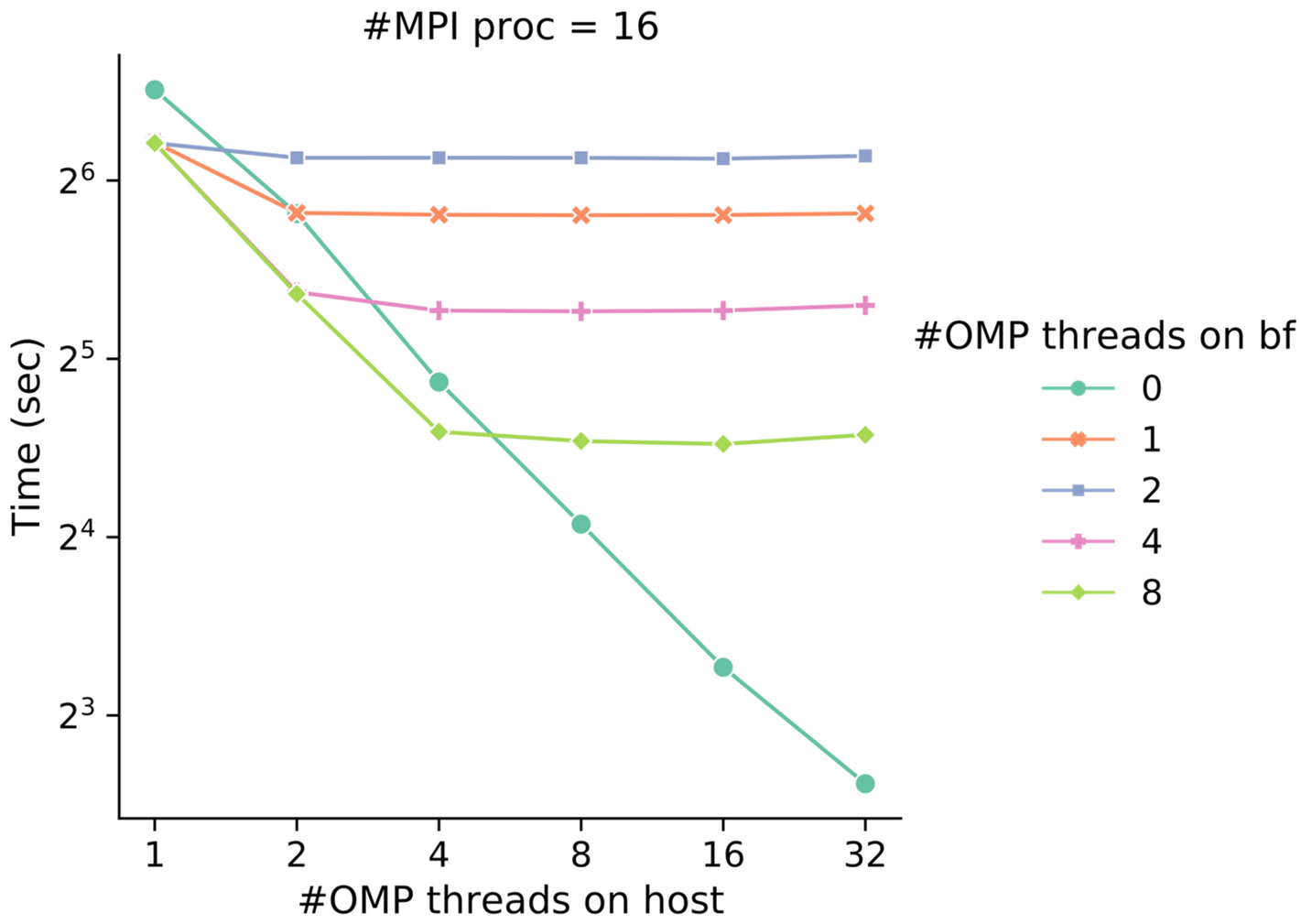
Hybrid MPI/OpenMP performance results

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.

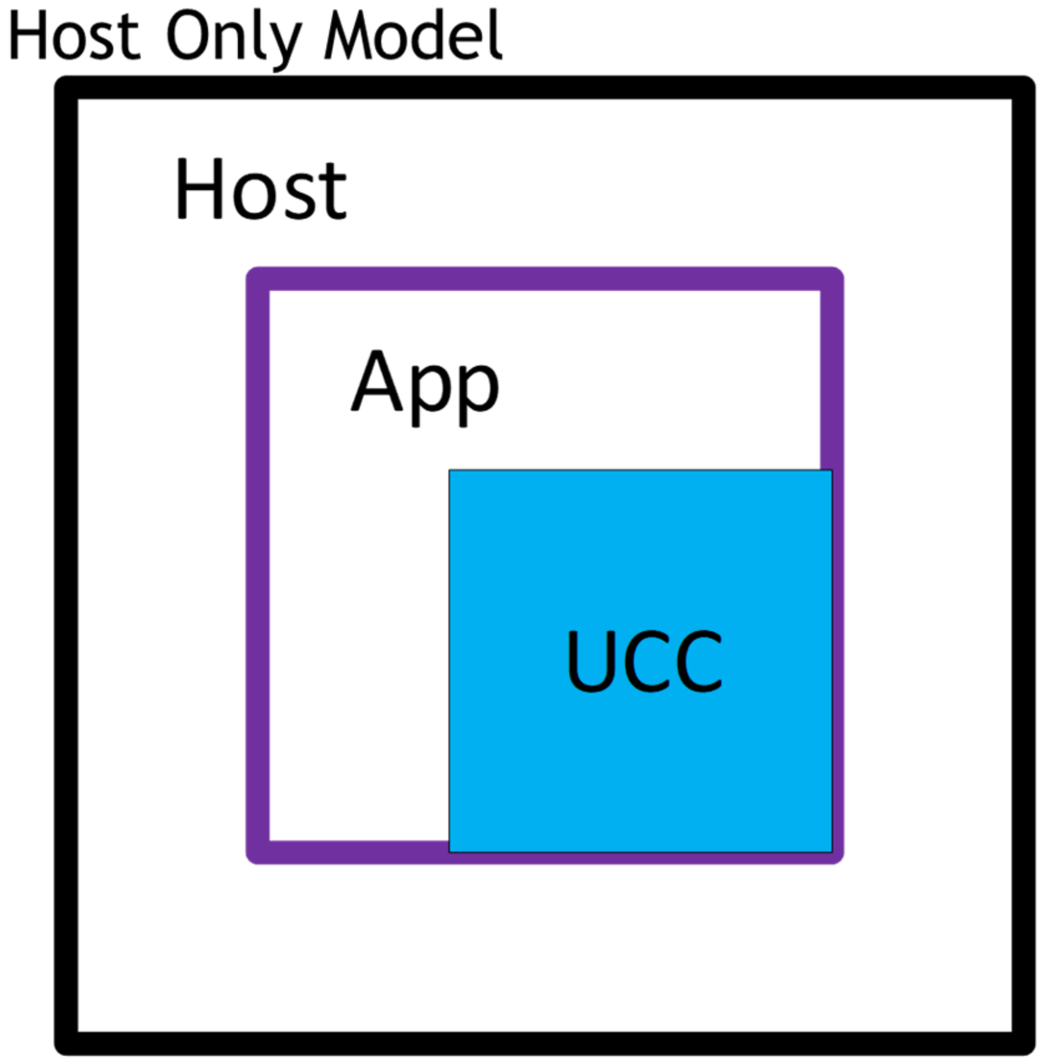
The knee of each curve indicates where the running times of neighbor-build on the host and force-compute on the BlueField are closest.

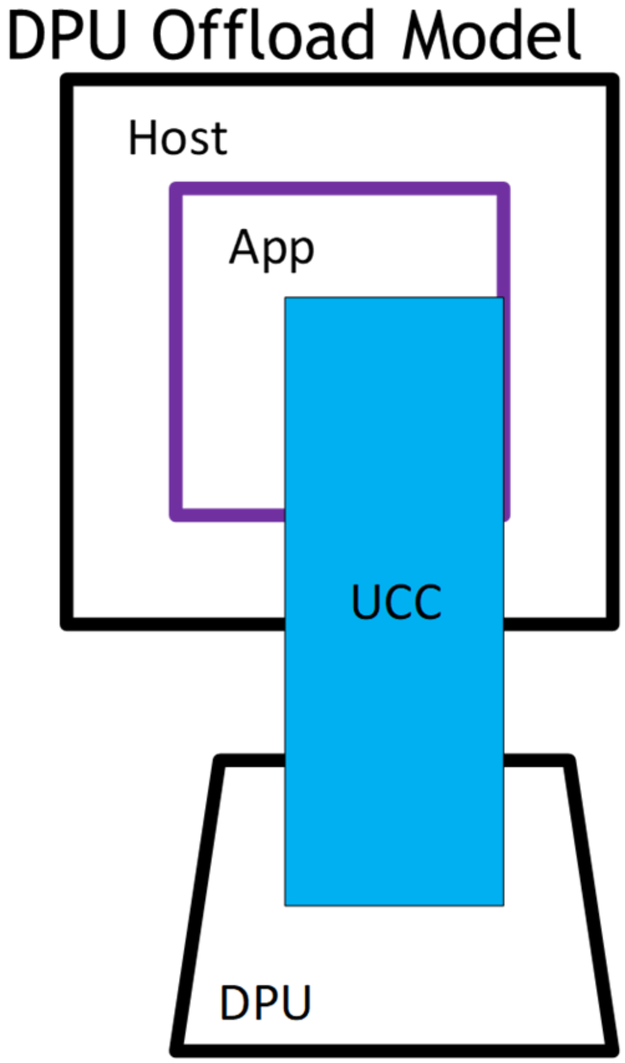
Thread synchronization overhead in the force computation routine causes the performance not to scale proportionally to the number of threads.



Using the SmartNIC as a Communication Accelerator

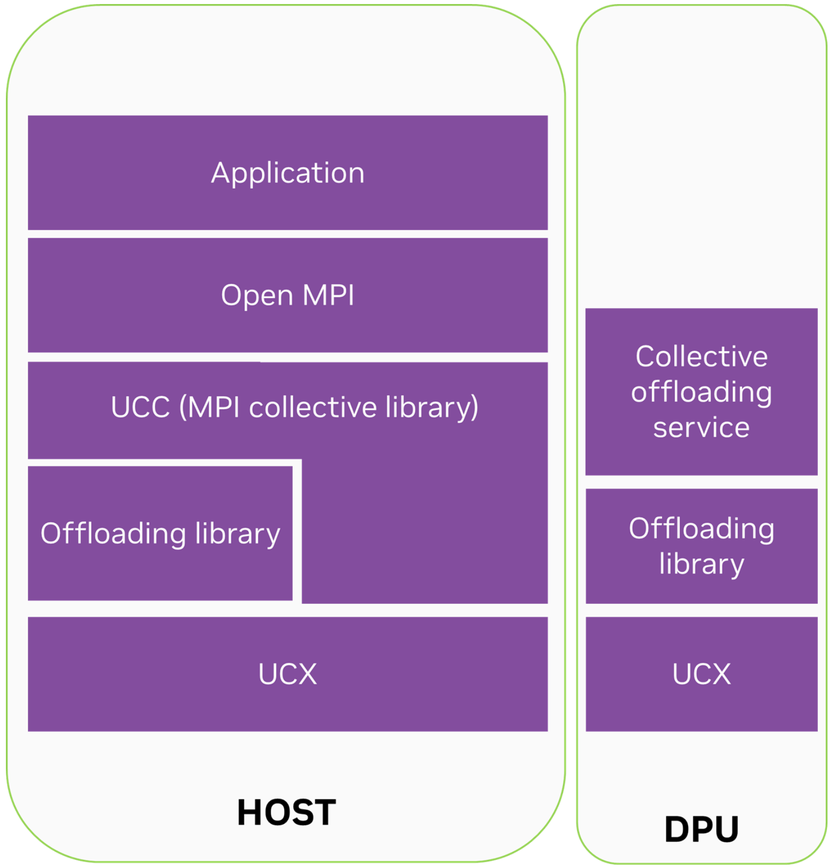
UCC Offload Model





Overview of the Software Stack

- UCX – low level comm library with access to hardware features (XGVMI)
- MIMOSA – asynchronous agent to progress the offloaded collectives
- UCC – collective library with selective algorithms
- Open MPI (vanilla)
- User applications (vanilla)



UCC Offload Software Stack

Host

- MLNX_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 (MIMOSA)
  - https://github.com/gvallee/dpu_offload_service
- UCC (example)
  - https://github.com/yqin/ucc/tree/topic/rdma_workshop
- Open MPI
  - https://github.com/open-mpi/ompi

DPU

MLNX_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 (MIMOSA)

https://github.com/gvallee/dpu_offload_service

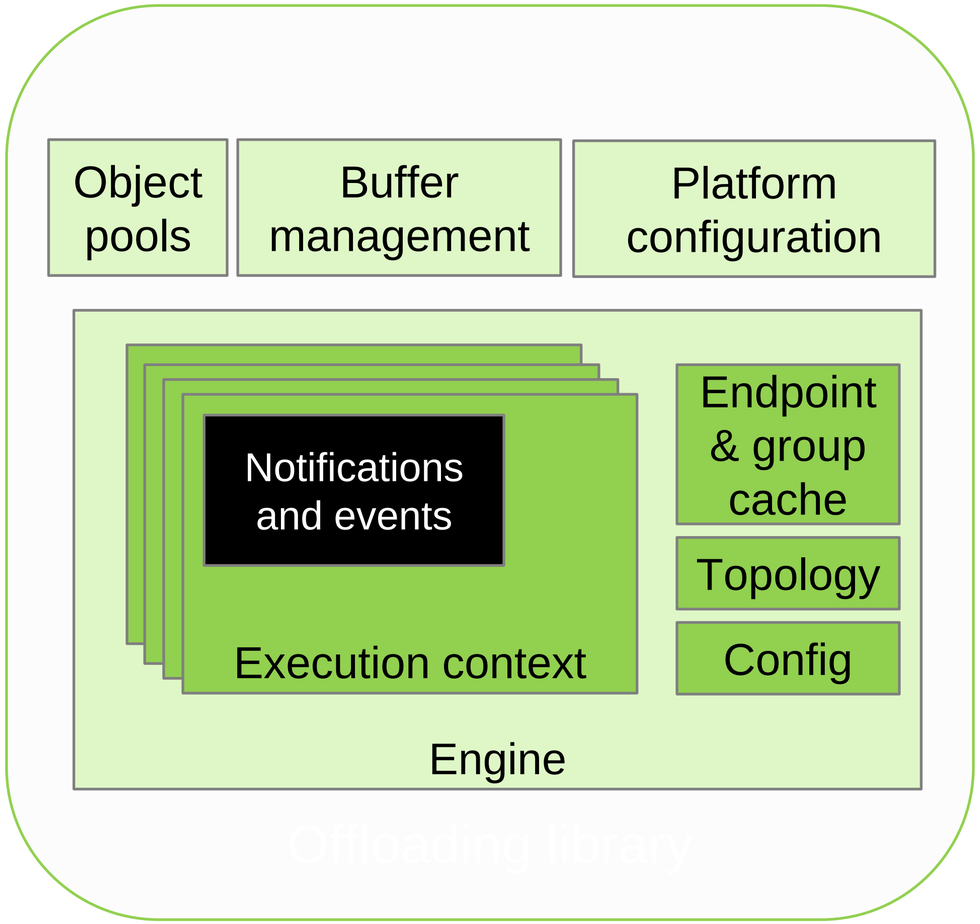
UCC* (example)

https://github.com/yqin/ucc/tree/topic/rdma_workshop

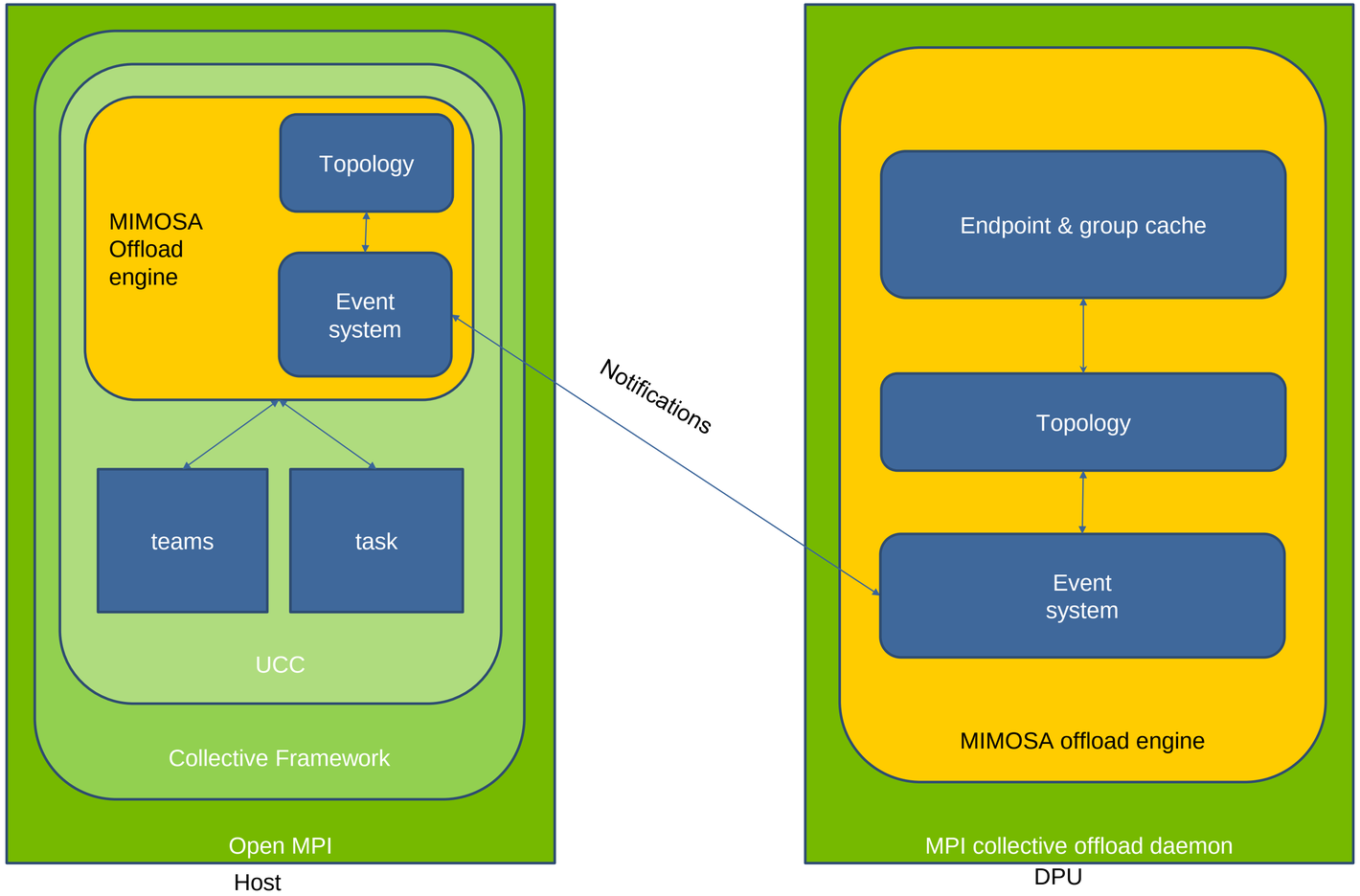
* UCC is not strictly required on the DPU, but the offload daemon and associated build script are located in this repository

Offloading infrastructure: MIMOSA

- **Multi-tenant Intelligent Modular Offloading Service Architecture**
- **Documentation**
  - https://github.com/gvallee/dpu_offload_service/tree/main/doc
  - https://gvallee.github.io/dpu_offload_service/
- **Daemon & Service Process**
  - An executable binary, based on building blocks, that implements a service running on a DPU
  - Support communications and notification between service processes
  - Support connection from any number of processes from the host
- **Service configuration**
  - Support configuration files specific to the platform that can be shared between users
- **Engine: umbrella object representing a specific service**
- **Execution context**
  - Client or server
  - Used to handle connectivity between processes

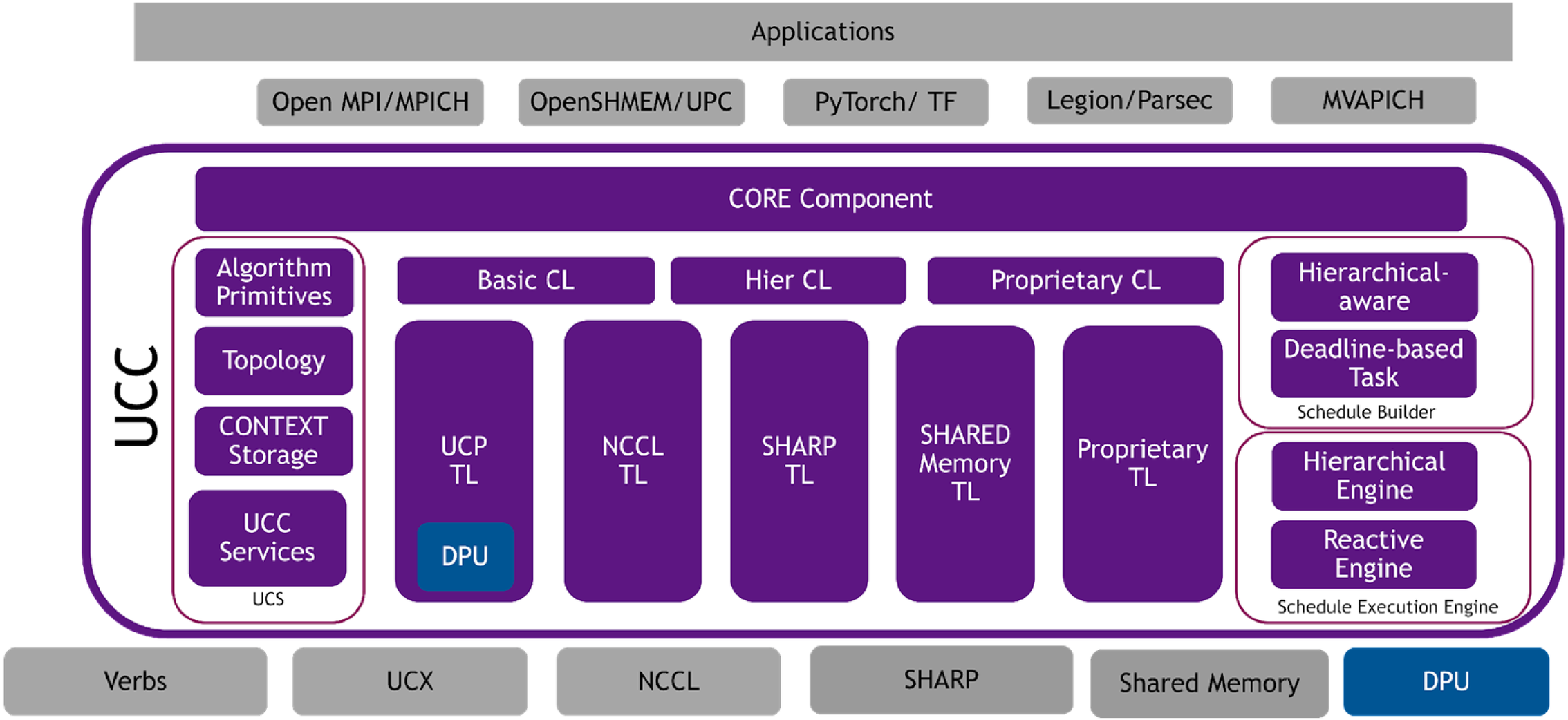


Architecture for the offload of MPI collective



- MIMOSA engine integrated into UCC
- MPI communicator/UCC teams are made *offload-aware*
- For the implementation of offloaded algorithms
  - MIMOSA's event system is used to implement control path
  - MIMOSA's endpoint & group cache is used to issue XGVMI operation to implement the data path
- MIMOSA's topology service
  - MPI communicator/UCC team aware
  - Provide mapping service process, hosts and ranks involved in the offloaded operations

Unified Collective Communication (UCC) Architecture



UCC Key Concepts

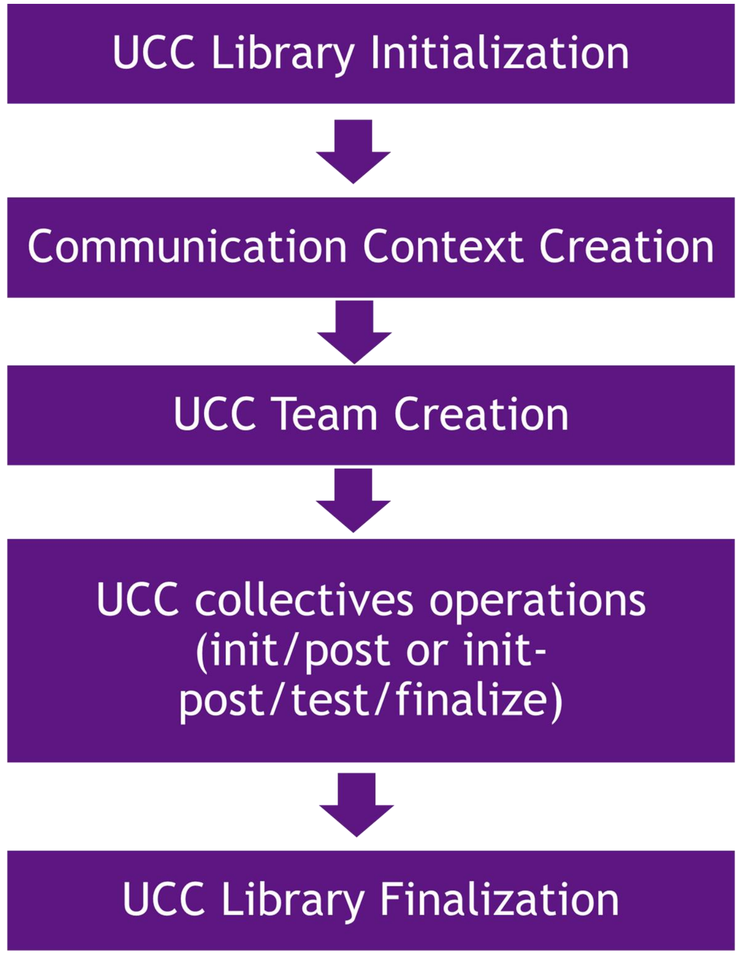
- Abstractions
  - Abstracts the resources required for collective operations
  - Local: Library, Context, Endpoints, Execution Engine
  - Global: Teams

- Operations
  - Defines how to interact with the abstractions
  - Create/modify/destroy the resources
  - Build, launch, and finalize collectives

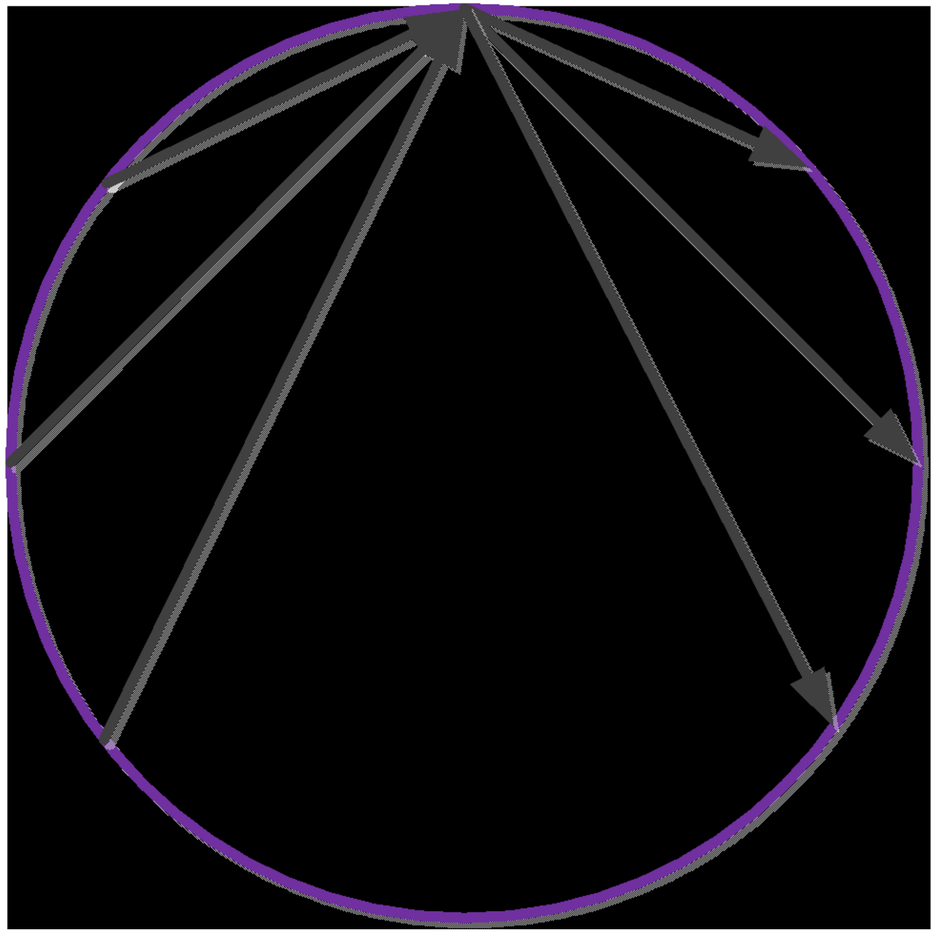
- Properties
  - Defines how to customize abstractions and operations
  - Explicit way to request optional features, semantics, and optimizations (opt-in or opt-out model)
  - Provides an ability to express and request many cross-cutting features
  - Properties are preferences expressed by the user of the library and what the library provides is queried

UCC Code Flow

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



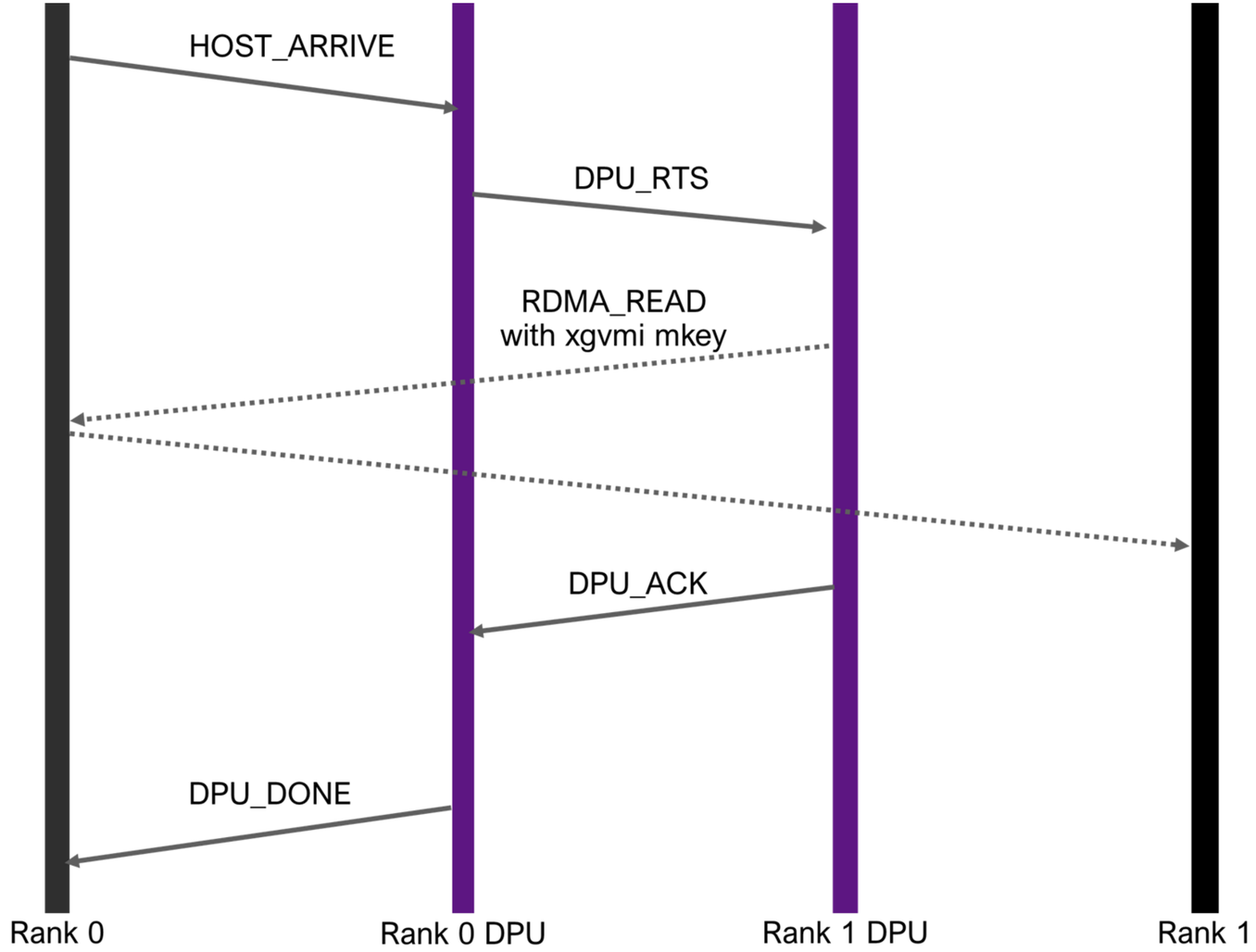
Host-Based Algorithm (Ring)



DPU Offload Algorithm

Sender: Rank 0

Receiver: Rank 1



Pros and Cons of This 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)

Break

See the updated agenda at https://github.com/gt-crnch-rg/smartnic-tutorial-sc23

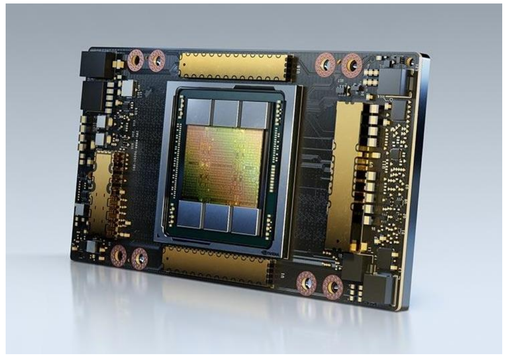
OpenMP offload to DPUs

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

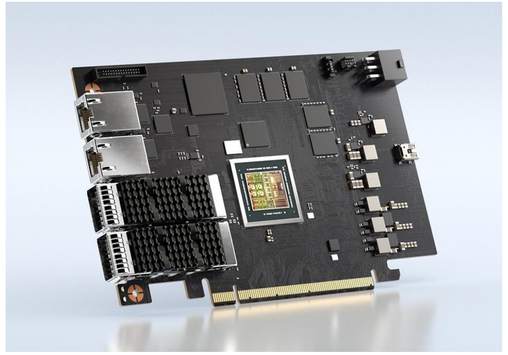
DPU Offloading Programming with standard OpenMP

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

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

GPU



DPU



OpenMP Target Offloading

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Basic Syntax

```
// exec in host


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

How to Choose Device?

```
// exec in host
#define __GPUID__ 4

#pragma omp target device(__GPUID__)
{
    // 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
```

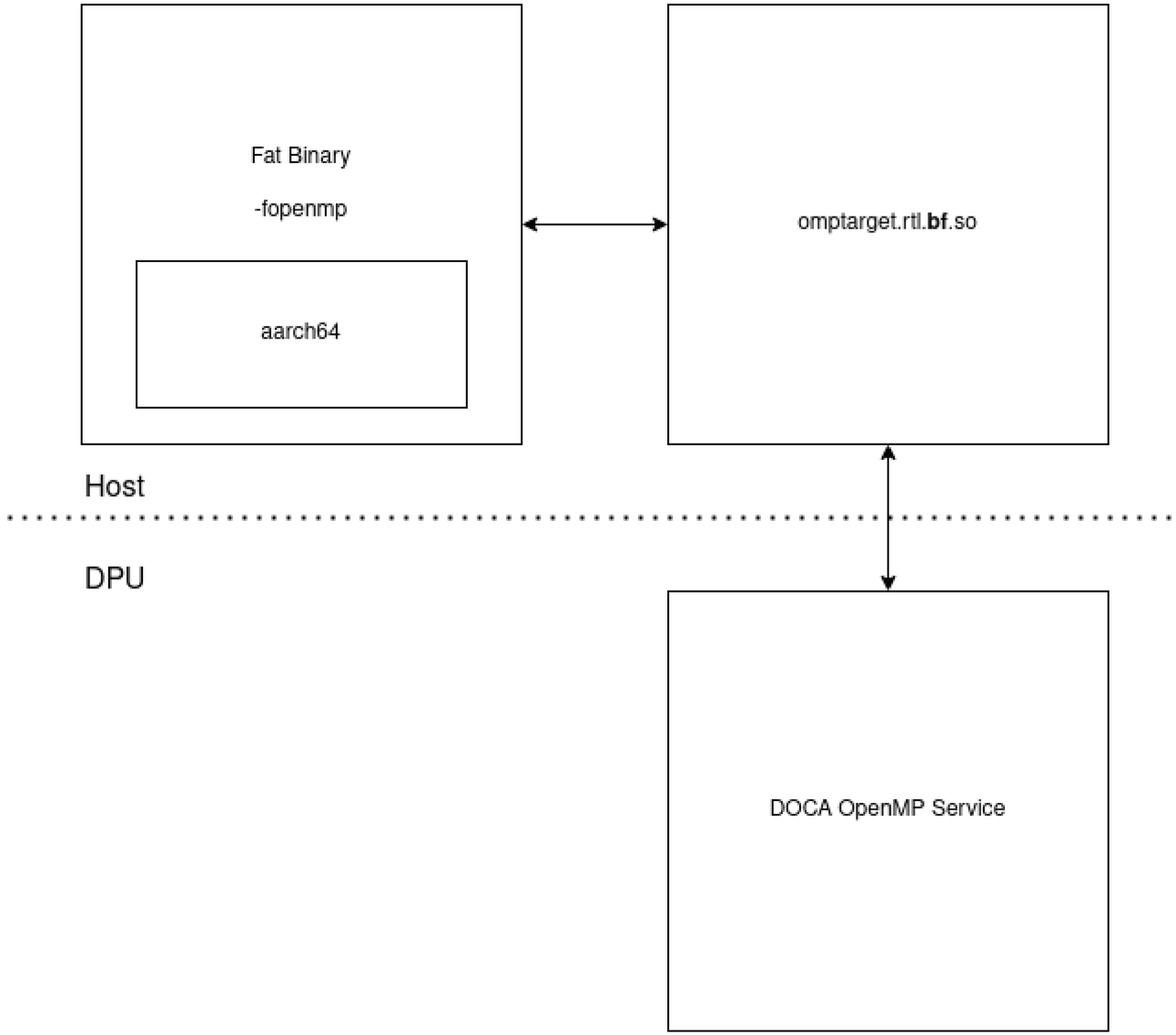
DPU Offloading with OpenMP Programming Model

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

DPU as an Accelerator

- Communication using DOCA to exchange data and binary

- LLVM Integration as a
  - Target Plugin
  - Cross-Compilation



Download ODOS (OpenMP DOCA Offloading Support) from the AccelCom@BSC software repository: https://www.bsc.es/discover-bsc/organisation/scientific-structure/accelerators-and-communications-hpc/team-software

How to Choose DPU?

```
// exec in host
#define __DPUID__ 7

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

How do I Know the Device Number?

```
$ llvm-omp-device-info

...
Device (4):
    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
...
```

Example Applications

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Test App for BlueField Target in OpenMP

```
// buf[0:12] = ‘A’

#pragma omp target
    puts(“\n[app] printf test”);           // printing in the DPU

#pragma omp target map(tofrom:buf[0:13])  // offloading ++ to the DPU
{
    for (i = 0; i < 13; ++i)
        ++buf[i];
}

// buf[0:12] is now ‘B’ in the host, after having being increased in the DPU
```

```
uthmanhere@jupiter030:~/test_openmp/build$ ./basic
[app] increment test: A
 A  A  A  A  A  A  A  A  A  A  A  A  A
[app] increment test: B
 B  B  B  B  B  B  B  B  B  B  B  B  B
uthmanhere@jupiter030:~/test_openmp/build$
```

```
uthmanhere@jupiterbf030:~$
[app] printf test
[app] increment test: A
[app] increment test: B
```

Asynchronous I/O in BlueField Target using OpenMP

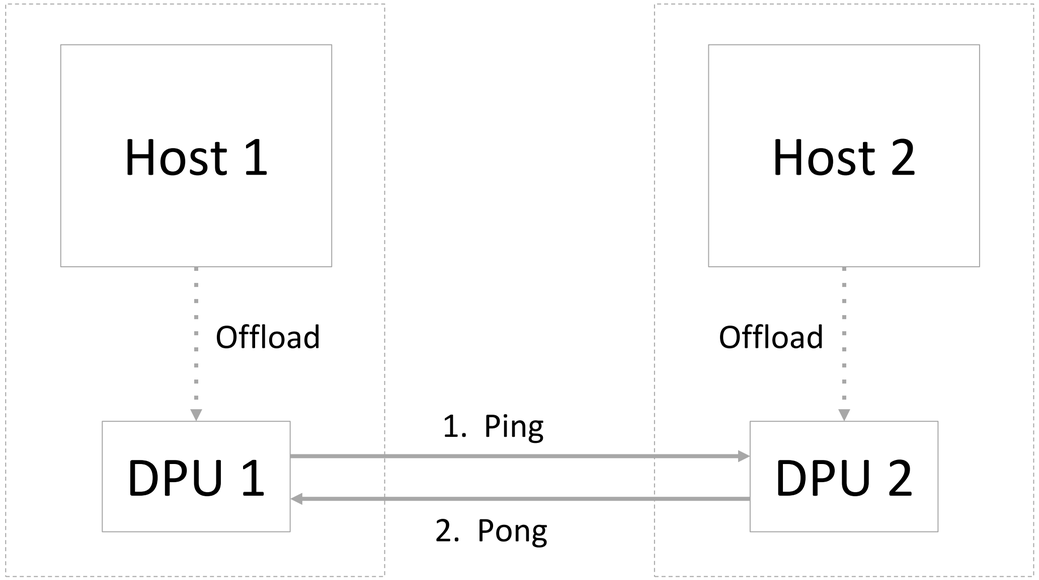
```
#pragma omp parallel
#pragma omp single
{
#pragma omp task
#pragma omp target nowait
{
    for (i = 0; i < 5; ++i)
        printf(“hola - %05d\n”, i);
} // end omp target
#pragma omp task
{
    for (j = 0; j < 5; ++j)
        printf(“adios - %06d\n”, j);
} // end omp task
} // end omp single
```

```
uthmanhere@jupiter030:~/test_openmp/build$ ./async
adio - 000000
adio - 000001
adio - 000002
adio - 000003
adio - 000004
uthmanhere@jupiter030:~/test_openmp/build$
```

```
uthmanhere@jupiterbf030:~$ hola - 00000
hola - 00001
hola - 00002
hola - 00003
hola - 00004
```

Asynchronous TCP I/O in BlueField Target using OpenMP

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

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

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



Asynchronous TCP I/O in BlueField Target using OpenMP

```
void ping_pong(int fd, char mode)
{
        int i, m;
        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$ ./net_accel
uthmanhere@thor013:~/omp_exp/labs_sc23/task_d/build$
```


```
uthmanhere@thorbf3a013:~$ ./doca_openmp_service
connection established.
to   client > 01
to   client > 03
to   client > 05
to   client > 07
to   client > 09
to   client > 11
to   client > 13
to   client > 15
to   client > 17
to   client > 19
```


```
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$ ./net_accel c
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$
```


```
uthmanhere@thorbf3a014:~$ ./doca_openmp_service
connection established.
to   server > 00
to   server > 02
to   server > 04
to   server > 06
to   server > 08
to   server > 10
to   server > 12
to   server > 14
to   server > 16
to   server > 18
```

MPI Use within BlueField Target in OpenMP

```
// host compute...

#pragma omp target nowait
{
    // presend compute...
    MPI_Send(...);
}

// host compute...

#pragma omp taskwait
```

COMING SOON...

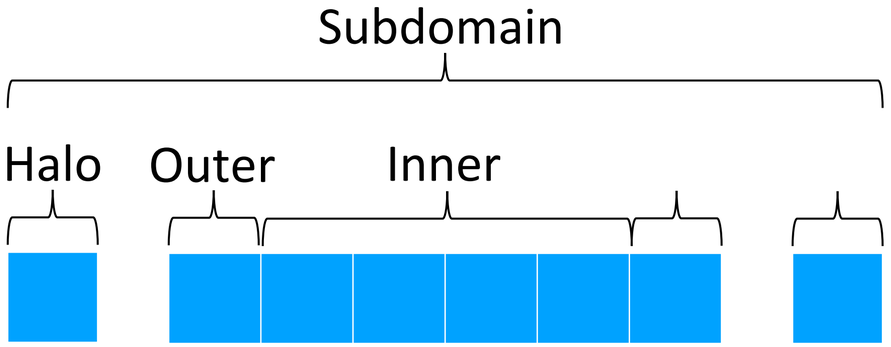
```
// host compute...

#pragma omp target nowait
{
    MPI_Recv(...);
    // postrecv copmute...
}

// host compute...

#pragma omp taskwait
```

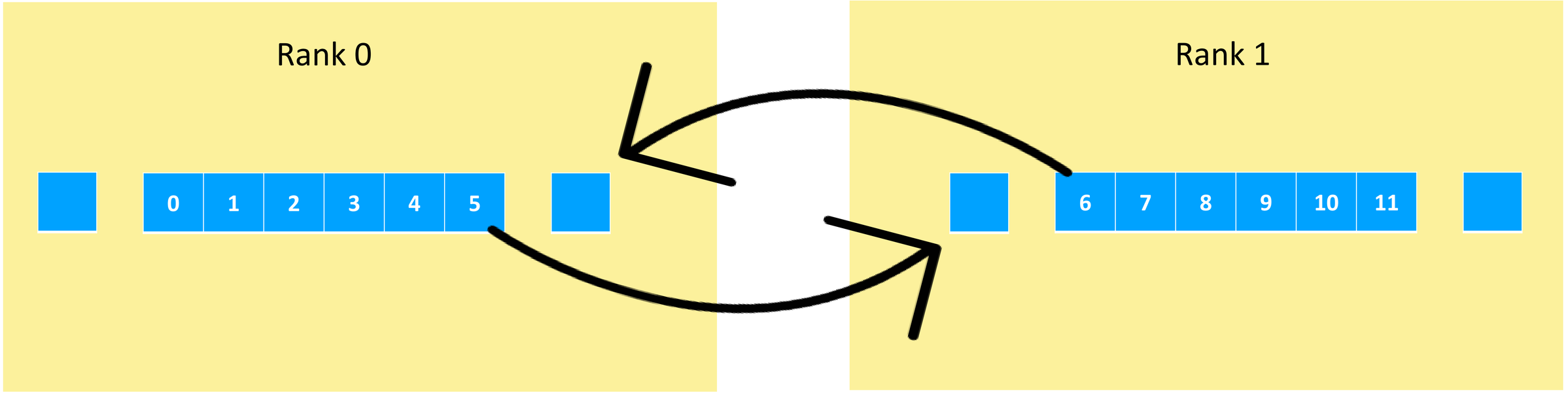
Example: Halo Exchange

Halo Exchange Offloading

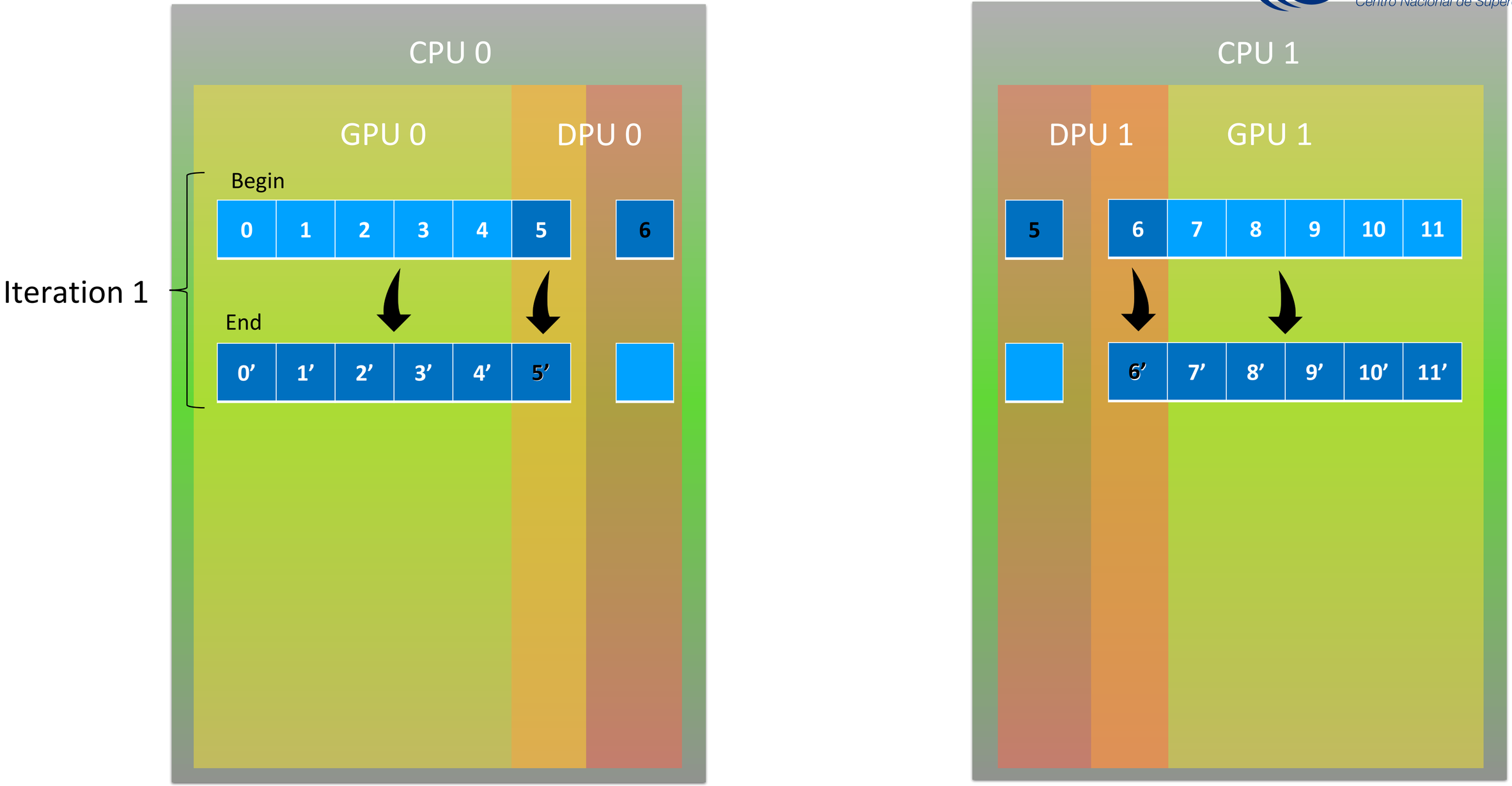
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

SIZE = 8
THICKNESS = 1



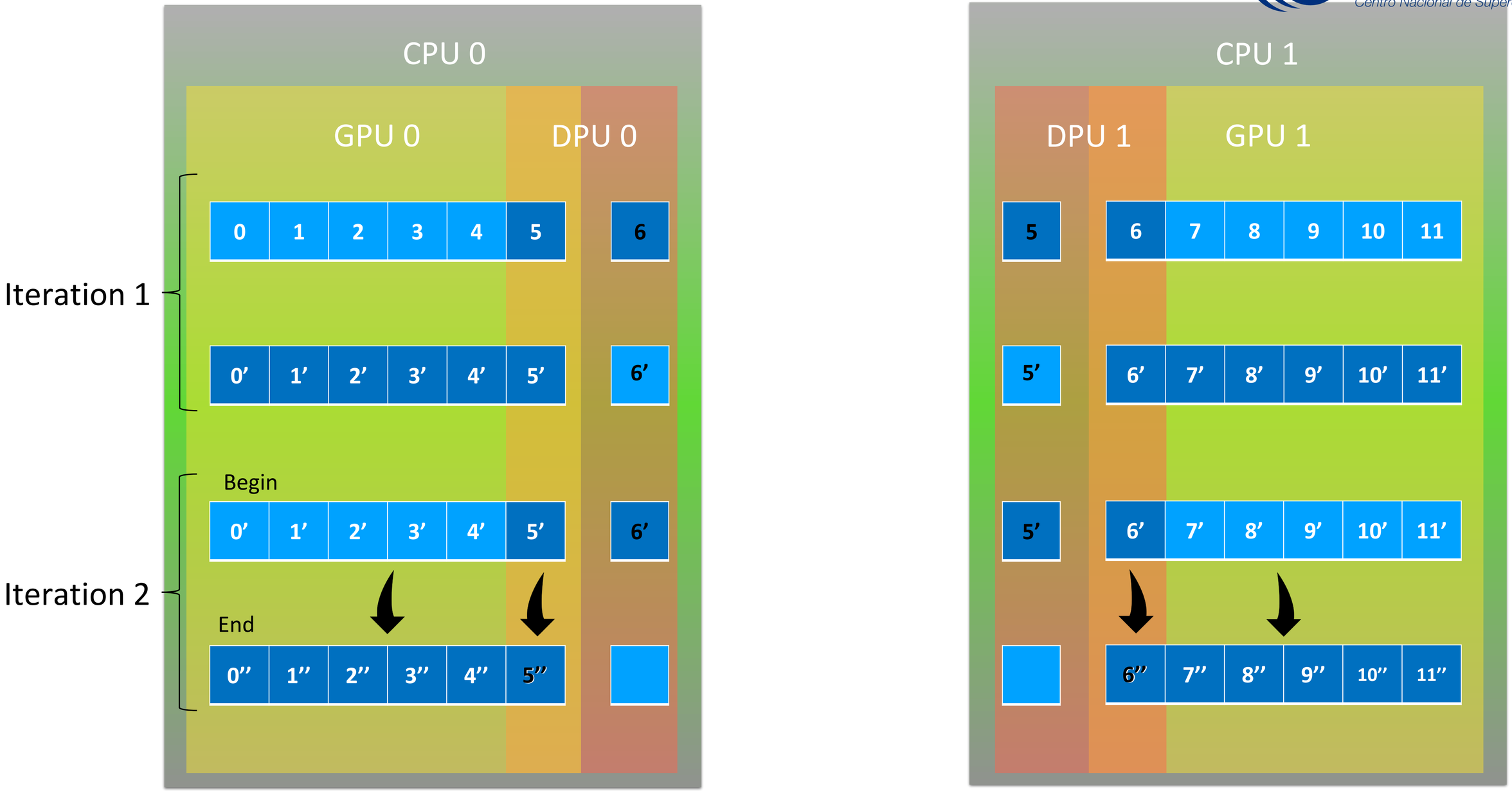


Execution step by step



Execution step by step

Barcelona Supercomputing Center
Centro Nacional de Supercomputación



```
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(0) 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(1) 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

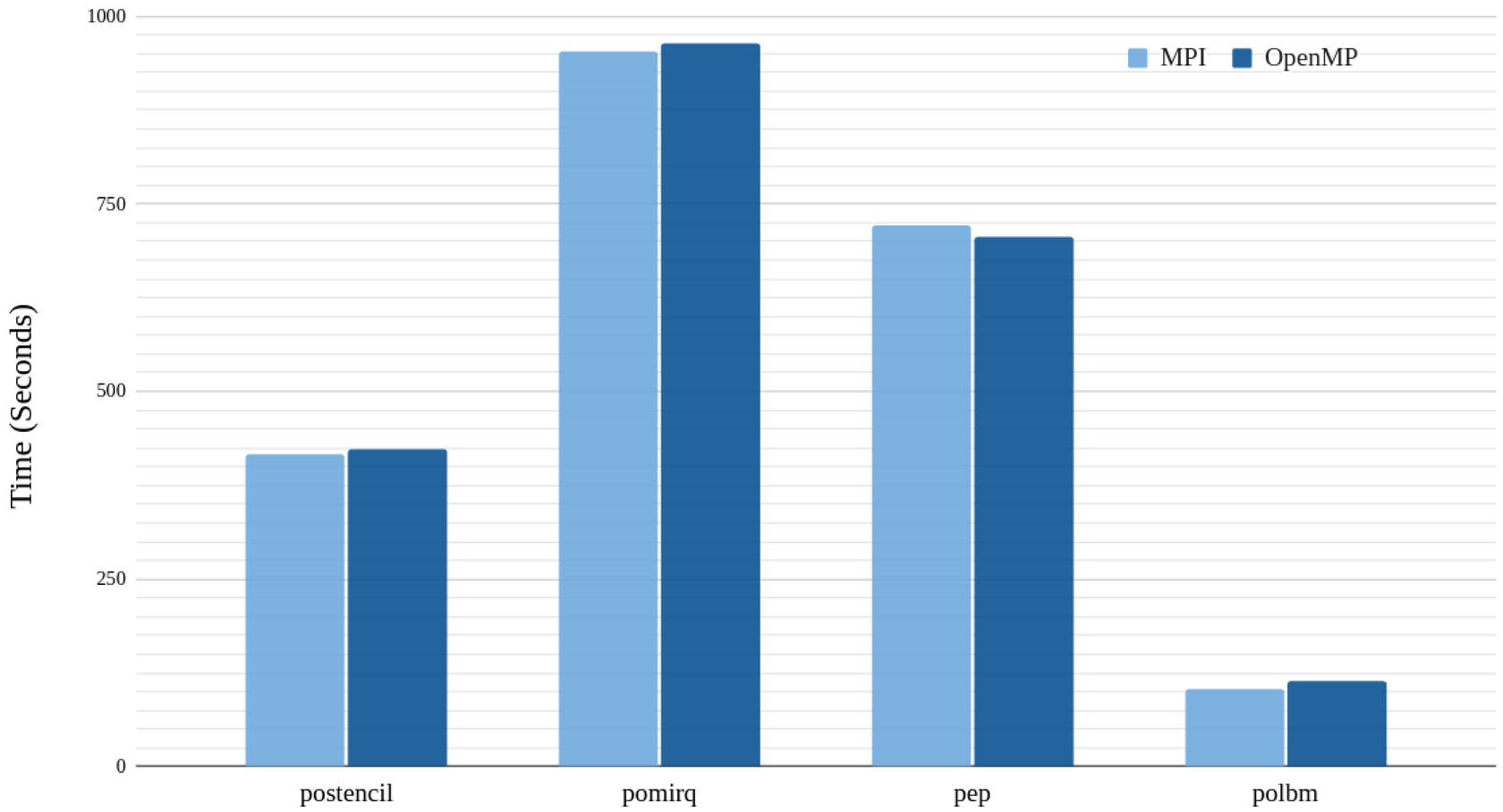
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

DPU Offloaded Function

```
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;
}
```

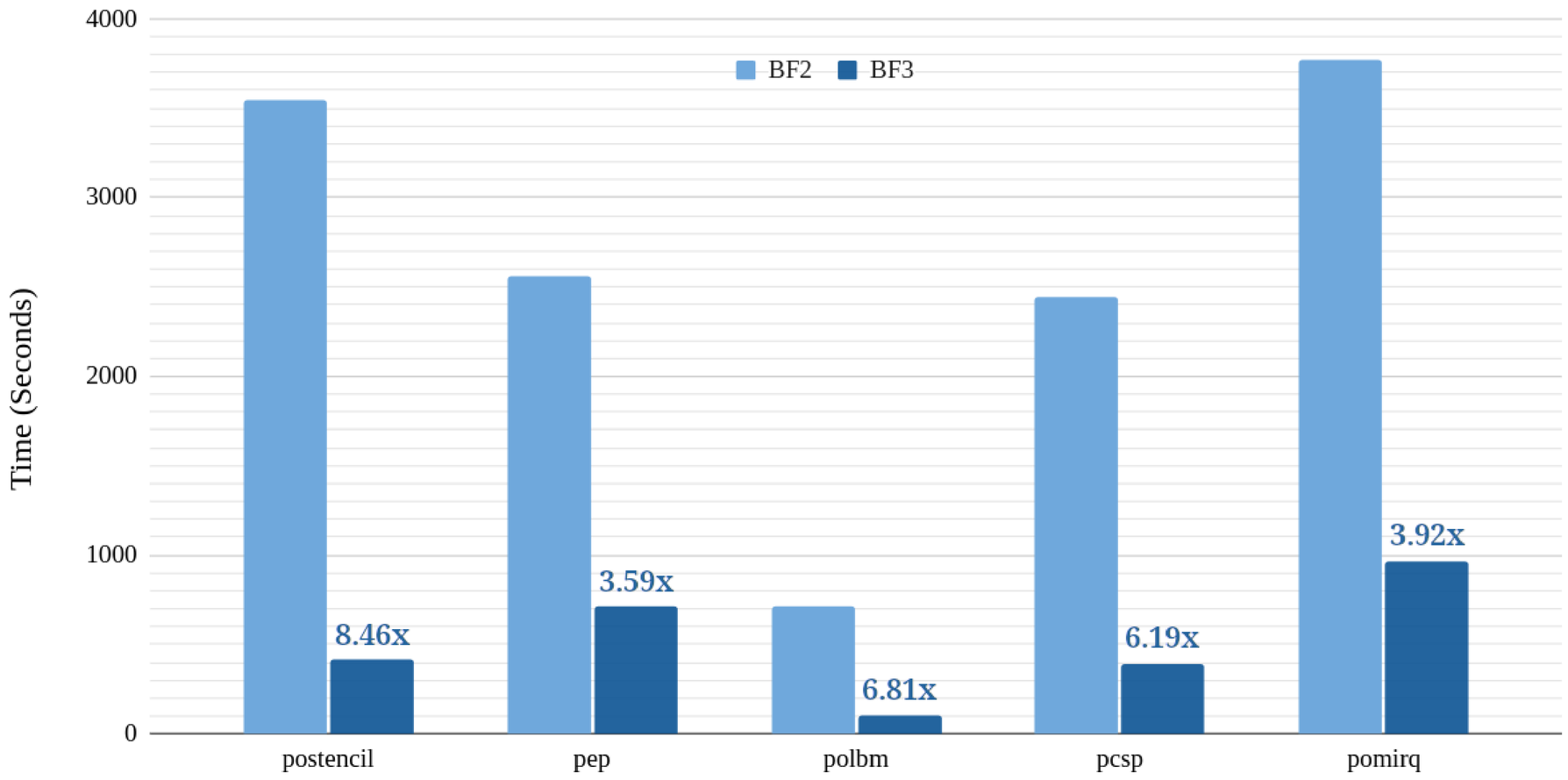
Performance

OpenMP and MPI Performance Comparison



Usman, Iserte, Ferrer, J. Peña, “DPU offloading programming with the OpenMP API”, in LLVM-HPC SC23 Workshop

BlueField 2 and BlueField 3 Performance Comparison



Usman, Iserte, Ferrer, J. Peña, “DPU offloading programming with the OpenMP API”, in LLVM-HPC SC23 Workshop

Hands On Section

Hands On Instructions

- Note that hands on instructions are on the Github
  - https://github.com/gt-crnch-rg/smartnic-tutorial-sc23
- The following slides are just examples of each hands on example and its expected output
  - Go ahead and start running jobs; you may not get to all the examples in the time allotted otherwise

Hands On Codes with BF3

- This section focuses on the following optimization strategies tested with BlueField DPUs
  - Unified Collective Collection (UCC) Offload
    - *Demo: UCC Collective offload example*
  - Blocking to Non-Blocking Transformations with UCC offload
    - *Demo: P3DFFT++*
  - Algorithmic Restructuring
    - *Demo: MiniMD*
  - Offload with OpenMP to the DPU
    - *Demo: OpenMP examples*

Hands On – MPI Collectives with UCC

Instructions

- https://github.com/gt-crnch-rg/smartnic-tutorial-sc23/blob/main/code/01-mpi-collectives/README.md
- Offload job launcher anatomy:
  - /global/home/groups/rdmaworkshop/offload_stack/run.sh bf3_ib ucc sep23 osu allgatherv 4 32 1 4 6 8
  - "/global/home/groups/rdmaworkshop/offload_stack/run.sh" – job launcher
  - "bf3_ib" – BF3 to use, the other ones are "bf2_ib" or "bf2_roce"
  - "ucc" – use offloaded UCC library, can also choose from "ompi" (use Open MPI tuned library, host only), or "hcoll" (use Nvidia HPC-X HCOLL library, host only), or "both" (use both UCC+HCOLL when applicable)
  - "sep23" – use "sep23" tech preview software stack
  - "osu" – benchmark application
  - "allgatherv" – benchmark test, can also choose from "alltoall" or "alltoallv", as well as all non-blocking ("i") versions
  - "4 32 1" – 4 nodes, 32 processes per node, 1 thread per process
  - "4 6 8" – 4 DPU offload service processes, 6 concurrent RDMA_READs and 8 concurrent RDMA_WRITEs

Offload Job Launcher Internal

```
if [[ ${BENCHMARK} == "osu" ]]; then
    start_offload_daemons "${OFFLOADLIST}" "${SPN}" "${DAEMONEXE}" "${CONFIG}" "${OFFLOADLIBS}" "${OFFLOAD_NET_DEVICES}" "${OFFLOAD_TLS}"
    EXEC="$(which ${BENCHMARK}_${TEST}) -f -m 256:1048576"
    build_mpirun
    $MPIRUN_CMD
else
    run_${BENCHMARK} ${TEST}
fi
```

P3DFFT+

Introducing P3DFFT++

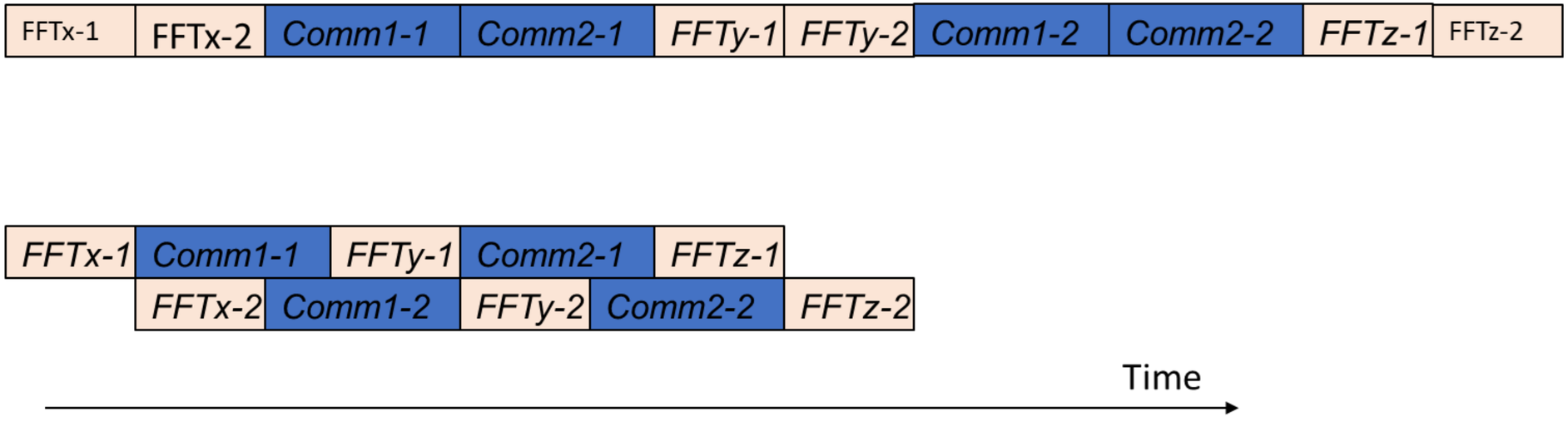
- Open source package implementing Fast Fourier Transforms in 3D based on 2D domain decomposition.
- Used in many areas of computational science, for purposes such as spectral analysis, solving non-linear differential equations with high precision, convolution and more
- Information about P3DFFT++ can 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 can also be run as timing experiments.
  - This demo will 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.
- This demo will 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).

Nonblocking implementation with multiple variables

- Overlap independent variables, in different phases of the algorithm (computation/communication)
- Larger buffer/message size, more efficient network bandwidth utilization



P3DFFT+ Demo

- https://github.com/gt-crnch-rg/smartnic-tutorial-sc23/tree/main/code/02-code-transformations-p3dfft/README.md
- run_p3dfft()

```
# P3DFFT++
function run_p3dfft {
   module load gcc/11 fftw/3.3.10-hpcx-2.15
   echo "1024 1024 1024 2 10 1 4" > stdin
   echo "32 10" > dims
   if [[ ${SPN} -ne 0 ]]; then
      HCOLL_TUNE="-x HCOLL_ML_DISABLE_ALLTOALLV=1 -x HCOLL_ML_DISABLE_IALLTOALLV=1"
      UCC_TUNE="-x UCC_TL_UCP_TUNE=allgatherv:0#alltoall:0#alltoallv:0-inf:@dpu_onesided"
   fi
   start_offload_daemons "${OFFLOADLIST}" "${SPN}" "${DAEMONEXE}" "${CONFIG}" "${OFFLOADLIBS}" "${OFFLOAD_NET_DEVICES}" "${OFFLOAD_TLS}"
   EXEC="/global/home/groups/rdmaworkshop/p3dfft++/sample/C++/test3D_c2c_cpp"
   build_mpirun
   echo "MPIRUN:             "$(echo ${MPIRUN_CMD} | tr -s ' ') | tee -a ${LOGFILE}
   /bin/time $MPIRUN_CMD 2>&1 | tee -a ${LOGFILE}
   rm -f stdin dims
}
```

MiniMD Demo

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-sc23/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

OpenMP Offload Hands-on

Setup

1. Log into the HPCAI machines (2 separate windows)

1. On 1 window, ssh to the BlueField and execute the DOCA OpenMP service

1. LLVM with ODOS is compiled for you, no need to recompile
   - Compiling instructions are in the handout for your reference only

OpenMP Offload (ODOS) Labs

- General instructions:
  a. Inspect and understand the code
  b. Compile with the provided script and execute
  c. Inspect and understand the shown results

1. Hello World
   - Prints “Hi Folks!” in the device
2. Target Parallel
   - Prints offloaded message in the device once per existing thread/core
3. Shared libraries
   - Demonstrates how to use shared libraries in the (ARM) DPU device (cross-compiling in the host)
4. Network accelerator
   - OpenMP-Offloaded ping-pong in the DPU (DPU-to-DPU) using TCP

Wrap-Up

How to continue working with SmartNICs and DPUs

You can access SmartNICs via a variety of general access testbeds

- HPC-AI Advisory Council Cluster
  - BlueField 2-3
  - High-speed NVIDIA switches
- CRNCH Rogues Gallery testbed
  - BlueField 1-3, Alveo FPGAs, NapaTech SmartNICs
- ORNL Wombat
  - BlueField 2-3
- Los Alamos testbed
  - Request access from testbed PIs
- Xilinx HACC centers
  - AMD Alveos for OpenNIC work

Testbed Links

https://hpcadvisorycouncil.atlassian.net/wiki/spaces/HPCWORKS/pages/107839491/To+request+an+account+at+HPC-AI+Advisory+Council+cluster+center

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

Reference Material

Cross-GVMI Memory Key API

UCX API

- Register mkey on the host

```
ucp_mem_map_params_t mparams = {.field_mask = UCP_MEM_MAP_PARAM_FIELD_ADDRESS | UCP_MEM_MAP_PARAM_FIELD_LENGTH,
.address = address, .length = length};
rc = ucp_mem_map(ucp_context, &mparams, memh);
```

- Pack memh on the host

```
ucp_memh_pack_params_t memh_pack_params = {.field_mask = UCP_MEMH_PACK_PARAM_FIELD_FLAGS, .flags = UCP_MEMH_PACK_FLAG_EXPORT};
rc = ucp_memh_pack(memh, &memh_pack_params, memh_buf, buf_size);
```

- Import memh on DPU/Create mkey alias on DPU

```
ucp_mem_map_params_t mparams = {.field_mask = UCP_MEM_MAP_PARAM_FIELD_EXPORTED_MEMH_BUFFER, .exported_memh_buffer = memh_buf};
rc = ucp_mem_map(ucp_context, &mparams, memh);
```

- Pack mkey alias as rkey for RDMA

```
rc = ucp_rkey_pack(ucp_context, memh, &rkey_buf, &rkey_buf_len);
```

UCT/DevX API

- Register mkey on the host
  - https://github.com/yqin/ucx/blob/a2212e9ed5cb833f53c8fd8475fae79ca30462aa/src/uct/ib/mlx5/dv/ib_mlx5dv_md.c#L1268-L1385
- Import mkey on DPU
  - https://github.com/yqin/ucx/blob/a2212e9ed5cb833f53c8fd8475fae79ca30462aa/src/uct/ib/mlx5/dv/ib_mlx5dv_md.c#L1387-L1452