NVIDIA DOCA Allreduce

Application Guide

MLNX-15-060535 _v2.2.0 | October 2023

Table of Contents

Chapter 1. Introdution

Allreduce is a collective operation which allows collecting data from different processing units to combine them into a global result by a chosen operator. In turn, the result is distributed back to all processing units.

Allreduce operates in stages. Firstly, each participant scatters its vector. Secondly, each participant gathers the vectors of the other participants. Lastly, each participant performs their chosen operation between all the gathered vectors. Using a sequence of different allreduce operations with different participants, very complex computations can be spread among many computation units.

Allreduce is widely used by parallel applications in high-performance computing (HPC) related to scientific simulations and data analysis, including machine learning calculation and the training phase of neural networks in deep learning.

Due to the massive growth of deep learning models and the complexity of scientific simulation tasks that utilize a network, effective implementation of allreduce is essential for minimizing communication time.

This document describes how to implement allreduce using the UCX communication framework, which leverages NVIDIA® BlueField® DPU by providing low-latency and high-bandwidth utilization of its network engine.

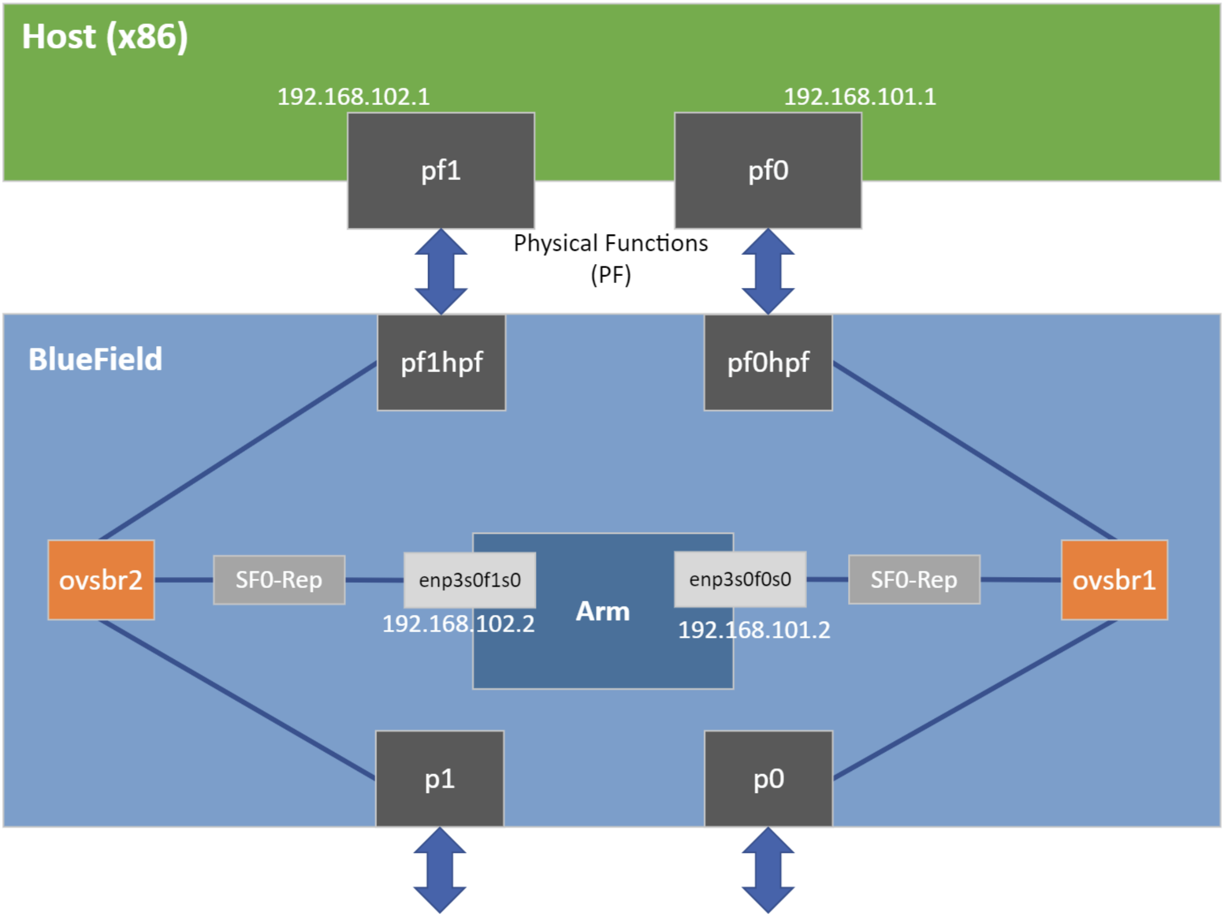
This document describes the following types of allreduce:

- Offloaded client – processes running on the host which only submit allreduce operation requests to a daemon running on the DPU. The daemon runs on the DPU and performs the allreduce algorithm on behalf of its on-host-clients (offloaded-client).
- Non-offloaded client – processes running on the host which execute the allreduce algorithm by themselves

Chapter 2. System Design

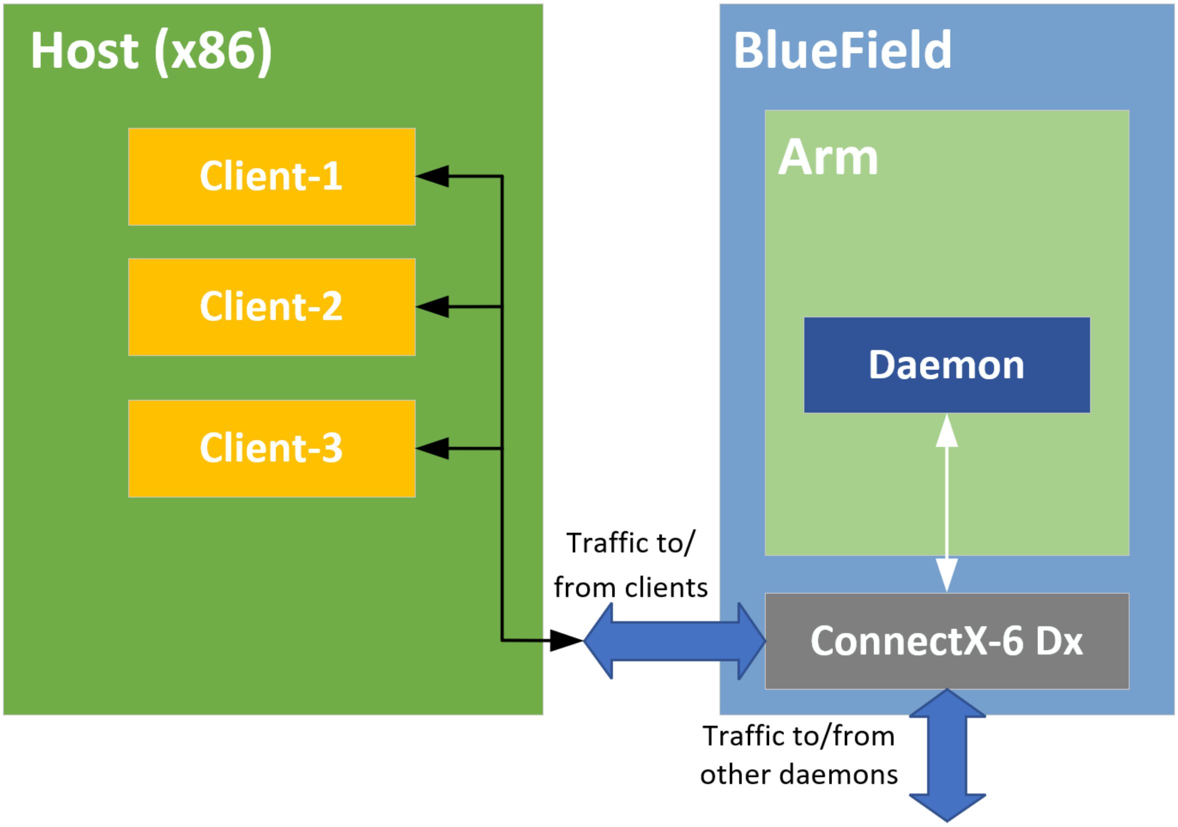
The application is designed to measure three metrics:

- Communication time taken by offloaded and non-offloaded allreduce operations
- Computation time taken by matrix multiplications which are done by clients until the allreduce operation is completed
- The overlap of the two previous metrics. The percentage of the total runtime during which both the allreduce and the matrix multiplications were done in parallel.

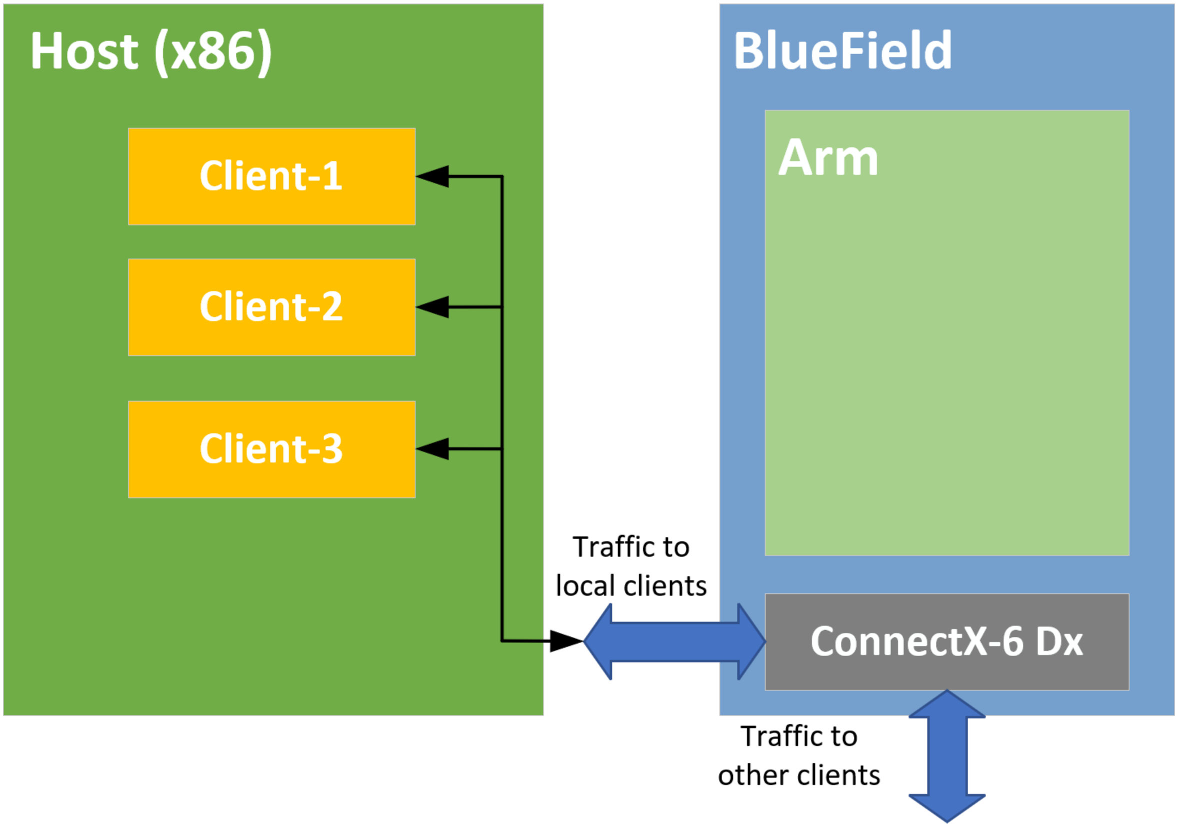


The allreduce implementation is divided into two different types of processes: clients and daemons. Clients are responsible for allocating vectors filled with data and initiating allreduce operations by sending a request with a vector to their daemon. Daemons are responsible for gathering vectors from all connected clients and daemons, applying a chosen operator on all received buffers, and then scattering the reduced result vector back to the clients.

- Offloaded mode



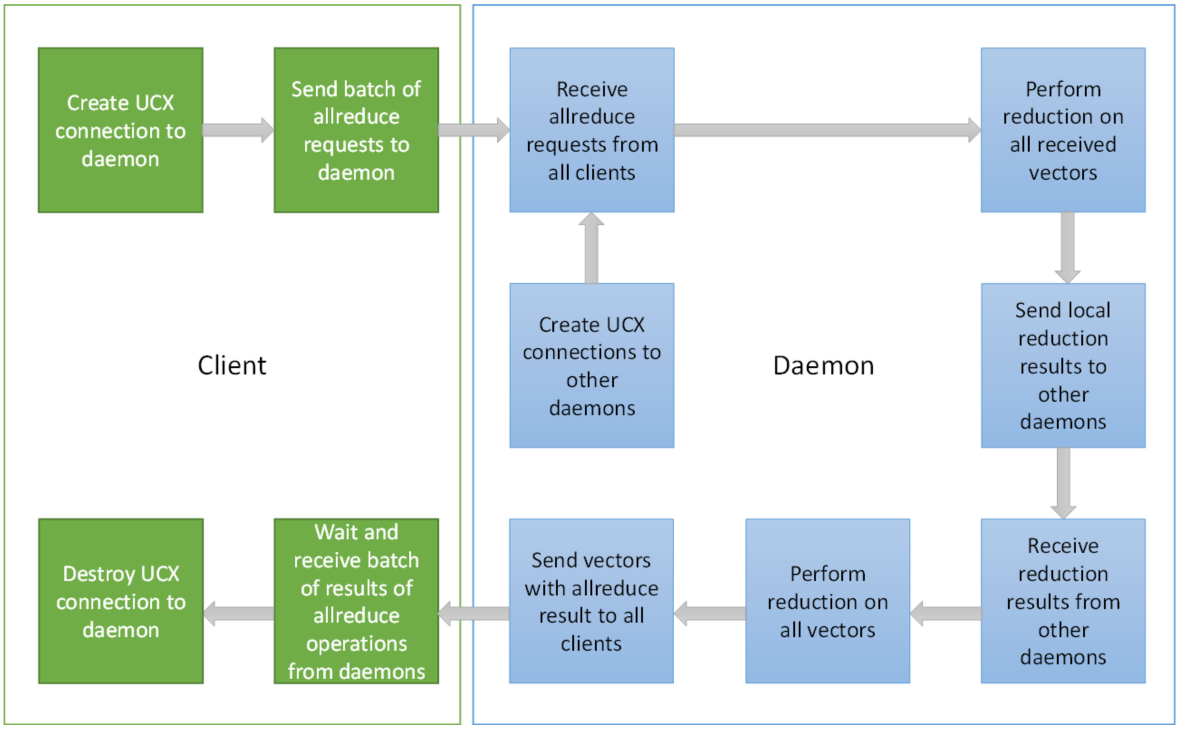
- Non-offloaded mode


Host (x86)
Client-1
Client-2
Client-3
BlueField
Arm
ConnectX-6 Dx
Traffic to local clients
Traffic to other clients

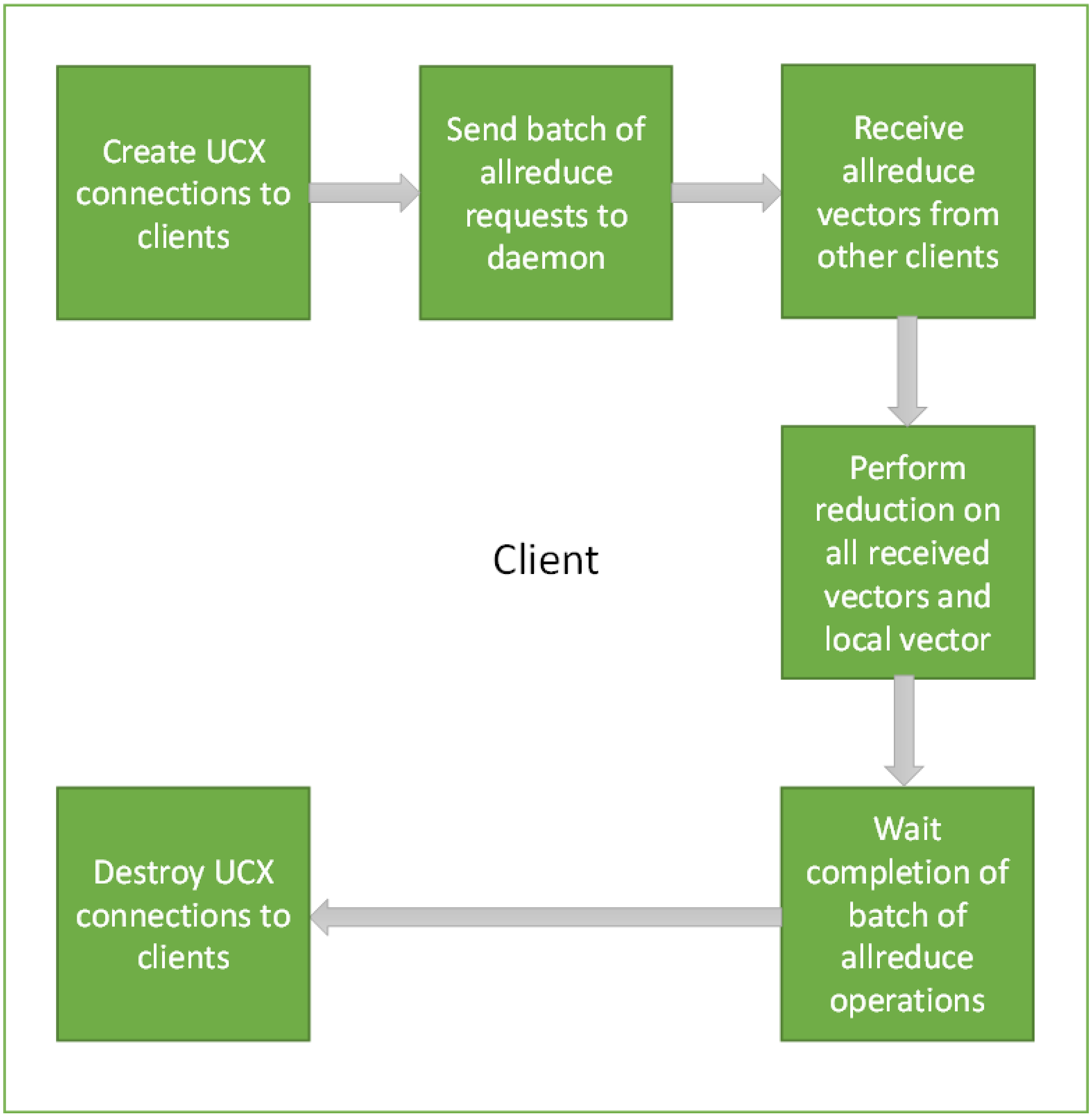
Chapter 3. Application Architecture

DOCA's allreduce implementation uses Unified Communication X (UCX) to support data exchange between endpoints. It utilizes UCX's sockaddr-based connection establishment and the UCX Active Messages (AM) API for communications.

- Offloaded mode


Create UCX connection to daemon
Send batch of allreduce requests to daemon
Receive allreduce requests from all clients
Perform reduction on all received vectors
Create UCX connections to other daemons
Client
Daemon
Send local reduction results to other daemons
Destroy UCX connection to daemon
Wait and receive batch of results of allreduce operations from daemons
Send vectors with allreduce result to all clients
Perform reduction on all vectors
Receive reduction results from other daemons

- Non-offloaded mode


Create UCX connections to clients
Send batch of allreduce requests to daemon
Receive allreduce vectors from other clients
Perform reduction on all received vectors and local vector
Wait completion of batch of allreduce operations
Destroy UCX connections to clients
Client

1. Connections between processes are established by UCX using IP addresses and ports of peers.
2. Allreduce vectors are sent from clients to daemons in offloaded mode, or from clients to clients in non-offloaded mode.
3. Reduce operations on vectors are done using received vectors from other daemons in offloaded mode, or other clients in non-offloaded mode.
4. Vectors with allreduce results are received by clients from daemons in offloaded mode, or are already stored in clients after completing all exchanges in non-offloaded mode.
5. After completing all allreduce operations, connections between clients are destroyed.

Chapter 4. DOCA Libraries

This application leverages the UCX framework DOCA driver.

Chapter 5. Configuration Flow

1. Parse application argument.
    a). Initialize arg parser resources and register DOCA general parameters.
    ```
    doca_argp_init();
    ```
    b). Register UCX allreduce application parameters.
    ```
    register_allreduce_params();
    ```
    c). Parse all registered parameters.
    ```
    doca_argp_start();
    ```
2. UCX initialization.
    a). Initialize hash table of connections.
    ```
    allreduce_ucx_init();
    ```
    b). Create UCP context.
    ```
    ucp_init();
    ```
    c). Create UCP worker.
    ```
    ucp_worker_create();
    ```
    d). Set AM handler for receiving connection check packets.
    ```
    ucp_worker_set_am_recv_handler();
    ```
3. Initialization of the allreduce connectivity.
    ```
    communication_init();
    ```
    a). Initialize hash table of allreduce super requests.

    b). Set "receive callback" for handshake messages.

    c). If daemon or non-offloaded client:
    - i. Set AM handler for receiving allreduce requests from clients.
      ```
      allreduce_ucx_am_set_recv_handler();
      ```
    - ii. Initialize UCX listening function. This creates a UCP listener.
      ```
      allreduce_ucx_listen();
      ```

    d). Initialize all connections.
    ```
    connections_init();
    ```
    - i. Go over all destination addresses and connect to each peer.
    - ii. Repeat until a successful send occurs (to check connectivity).
      ```
      ucp_am_send_nbx();
      allreduce_ucx_request_wait();
      ```
    - iii. Insert the connection to the hash table of connections.
      ```
      allreduce_outgoing_handshake();
      ```

   e). Scatter handshake message to peers/daemon to make sure they all have the same `-s`, `-i`, `-b`, and `-d` flags.

4. Daemon: Start UCX progress.

   ```
   daemon_run();
   ```

   a). Set AM handler to receive allreduce requests from clients.

   ```
   allreduce_ucx_am_set_recv_handler();
   ```

   b). Perform UCP worker progress.

   ```
   while (running)
       allreduce_ucx_progress();
   ```

   c). Callbacks are invoked by incoming/outgoing messages by calling `allreduce_ucx_progress`.

5. Client:

   ```
   client_run();
   ```

   a). Allocate buffers to store allreduce initial data and results.

   ```
   allreduce_vectors_init();
   ```

   b). Set an AM handler for receiving allreduce results.

   ```
   allreduce_ucx_am_set_recv_handler();
   ```

   c). Perform allreduce barrier. Check that all daemons and clients are active.

   ```
   allreduce_barrier();
   ```

      i. Submit a batch of allreduce operations with 0 byte.

      ii. Wait for completions.

   d). Reset metrics and vectors.

   ```
   allreduce_metrics_init();
   ```

      i. Submit some batches and calculate estimated network time.

      ii. Allocate matrices to multiply.

      iii. Estimate how many matrix multiplications could have been performed instead of networking (same time window).

      iv. Calculate the actual computation time of these matrix multiplications.

   e). Reset vectors.

   f). Submit a batch of allreduce operations to daemon/peer (depends on mode).

   g). Perform matrix multiplications during a time period which is approximately equal to doing a single batch of allreduce operations and calculate the actual time cost.

   h). Wait for the allreduce operation to complete and calculate time cost.

   i). Update metrics.

   ```
   Do num-batches (flag) times:
   e.    allreduce_vectors_reset();
   f.    allreduce_batch_submit();
   g.    cpu_exploit();
   h.    allreduce_batch_wait();
   i.    allreduce_metrics_calculate();
   ```

   j). Print summary of allreduce benchmarking.

6. Arg parser destroy.

   ```
   doca_argp_destroy();
   ```

7. Communication destroy.

a). Clean up connections.

```
allreduce_ucx_disconnect();
```

   i. Remove the connection from the hash table of the connections.

   ii. Close inner UCP endpoint.

   ```
   ucp_ep_close_nbx();
   ```

   iii. Wait for the completion of the UCP endpoint closure.

   iv. Destroy connection.

   v. Free connections array.

b). Destroy the hash table of the allreduce super requests.

8. Destroy UCX context.

   a). Destroy the hash table of the connections.

   ```
   g_hash_table_destroy();
   ```

   b). If the UCP listener was created, destroy it.

   ```
   ucp_listener_destroy();
   ```

   c). Destroy UCP worker.

   ```
   ucp_worker_destroy();
   ```

   d). Destroy UCP context.

   ```
   ucp_cleanup();
   ```

Chapter 6. Running the Application

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA applications.
   - NVIDIA DOCA Applications Overview for additional compilation instructions and development tips of DOCA applications.
2. The allreduce binary is located under `/opt/mellanox/doca/applications/allreduce/bin/doca_allreduce`. To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/
meson build
ninja -C build
```

3. To build only the allreduce application:

   a). Edit the following flags in `/opt/mellanox/doca/applications/meson_options.txt`:
   - Set `enable_all_applications` to `false`
   - Set `enable_allreduce` to `true`

   b). Run the commands in step 2.

> Note: `doca_allreduce` is created under `./build/allreduce/src/`.

Application usage:

```
Usage: doca_allreduce [DOCA Flags] [Program Flags]

DOCA Flags:
  -h, --help                        Print a help synopsis
  -v, --version                     Print program version information
  -l, --log-level                   Set the log level for the program
 <CRITICAL=20, ERROR=30, WARNING=40, INFO=50, DEBUG=60>

Program Flags:
  -r, --role                        Run DOCA UCX allreduce process as: "client"
 or "daemon"
  -m, --mode <allreduce_mode>       Set allreduce mode: "offloaded", "non-
offloaded" (valid for client only)
  -p, --port <port>                 Set default destination port of daemons/
clients, used for IPs without a port (see '-a' flag)
  -t, --listen-port <listen_port>   Set listening port of daemon or client
  -c, --num-clients <num_clients>   Set the number of clients which participate
 in allreduce operations (valid for daemon only)
  -s, --size <size>                 Set size of vector to do allreduce for
  -d, --datatype <datatype>         Set datatype
 ("byte", "int", "float", "double") of vector elements to do allreduce for
  -o, --operation <operation>       Set operation ("sum", "prod") to do between
 allreduce vectors
  -b, --batch-size <batch_size>     Set the number of allreduce operations
 submitted simultaneously (used for handshakes by daemons)
  -i, --num-batches <num_batches>   Set the number of batches of allreduce
 operations (used for handshakes by daemons)
  -a, --address <ip_address>        Set comma-separated list of destination IPv4/
IPv6 addresses and ports optionally (<ip_addr>:[<port>]) of daemons or clients
```

4. Running the application on BlueField:

   - All daemons should be deployed before clients. Only after connecting to their peers are they able to handle clients.
   - Pre-run setup:

     UCX probes the system for any available net/IB devices and, by default, tries to create a multi-device connection. This means that if some network devices are available but provide an unreachable path from the daemon to the peer/client, UCX may still use that path. A common case is that a daemon tries to connect to a different BlueField using `tmfifo_net0` which is connected to the host only. To fix this issue, follow these steps:

     a). Use the UCX env variable `UCX_NET_DEVICES` to set usable devices. For example:

     ```
     export UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0
     /opt/mellanox/doca/applications/allreduce/bin/doca_allreduce -r daemon -t
      34001 -c 1 -s 100 -o sum -d float -b 16 -i 16
     ```

     Or:

     ```
     env UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0 /opt/mellanox/doca/applications/
     allreduce/bin/doca_allreduce -r daemon -t 34001 -c 1 -s 100 -o sum -d
      float -b 16 -i 16
     ```

     b). Get the mlx device name and port of a SF to limit the UCX network interfaces and allow IB. For example:

     ```
     dpu> show_gids
     DEV  PORT INDEX GID      IPv4     VER DEV
     --- ---- ----- ---      ------------        --- ---
     mlx5_2 1 0     fe80:0000:0000:0000:0052:72ff:fe63:1651       v2
      enp3s0f0s0
     mlx5_3 1 0     fe80:0000:0000:0000:0032:6bff:fe13:f13a       v2
      enp3s0f1s0

     dpu> UCX_NET_DEVICES=enp3s0f0s0,enp3s0f1s0,mlx5_2:1,mlx5_3:1 /opt/
     mellanox/doca/applications/allreduce/bin/doca_allreduce -r daemon -t 34001
      -c 1 -s 100 -o sum -d float -b 16 -i 16
     ```

   - CLI example for running a client:

     ```
     /opt/mellanox/doca/applications/allreduce/bin/doca_allreduce -r client -m
      offloaded -t 34001 -a 10.21.211.3:35001 -s 65535 -i 16 -b 128 -o sum -d float
     ```

     Note: The flags `-s`, `-i`, `-b`, and `-d` must be the same for all clients and daemons participating in the allreduce operation.

- CLI example for running a daemon:

  ```
  /opt/mellanox/doca/applications/allreduce/bin/doca_allreduce -r daemon -t
   34001 -c 2 -a 10.21.211.3:35001,10.21.211.4:36001 -s 65535 -o sum -d float -
  i 16 -b 128
  ```

  > Note: The flag `-a` is necessary for communicating with other daemons. In case of an offloaded client, the address must be that of the daemon which performs the allreduce operations for them. In case of a daemon or non-offloaded clients, the flag could be a single or multiple addresses of other daemons/non-offloaded clients which exchange their local allreduce results.

  > Note: The flag `-c` must be specified for daemon processes only. It indicates how many clients submit their allreduce operations to the daemon.

  > Note: The daemon listens to incoming connection requests on all available IPs, but the actual communication after the initial "UCX handshake" does not necessarily use the same device used for the connection establishment.

5. Running the application on the host, CLI example:

   ```
   /opt/mellanox/doca/applications/allreduce/bin/doca_allreduce -r client -m non-
   offloaded -t 34001 -a 10.21.211.3:35001,10.21.211.4:36001 -s 65535 -i 16 -b 128 -
   o sum -d float
   /opt/mellanox/doca/applications/allreduce/bin/doca_allreduce -r client -m
    offloaded -p 34001 -a 192.168.100.2 -s 65535 -i 16 -b 128 -o sum -d float
   ```

   > Note: Refer to section "Running DOCA Application on Host" in NVIDIA DOCA Virtual Functions User Guide.

Chapter 7. Arg Parser DOCA Flags

Refer to NVIDIA DOCA Arg Parser Programming Guide for more information.

Flag Type	Short Flag	Long Flag/JSON Key	Description
General flags	`l`	`log-level`	Set the log level for the application: ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60
	`v`	`version`	Print program version information
	`h`	`help`	Print a help synopsis
Program flags	`r`	`role`	Run DOCA UCX allreduce process as either `client` or `daemon`
	`m`	`mode`	Set allreduce mode. Available types options: ▶ `offloaded` ▶ `non-offloaded` (valid for client only)
	`p`	`port`	Set default destination port of daemons/clients. Used for IPs without a port (see `-a` flag).
	`c`	`num-clients`	Set the number of clients which

Flag Type	Short Flag	Long Flag/JSON Key	Description
			participate in allreduce operations Note: Valid for daemon only.
	s	size	Set size of vector to perform allreduce for
	d	datatype	Set datatype of vector elements to do allreduce for ▶ byte ▶ int ▶ float ▶ double
	o	operation	Set operation to perform between allreduce vectors
	b	batch-size	Set the number of allreduce operations submitted simultaneously. Used for handshakes by daemons.
	i	num-batches	Set the number of batches of allreduce operations. Used for handshakes by daemons.
	t	listen-port	Set listening port of daemon or client
	a	address	Set comma-separated list of destination IPv4/IPv6 address and ports optionally of daemons or clients. Format: <ip_addr>:[<port>].

Chapter 8. Running Application on NVIDIA Converged Accelerator

This section details the steps necessary to run DOCA Allreduce on NVIDIA converged accelerator.

Allreduce running on the converged accelerator has the same logic as described in previous sections except for the reducing of vectors. The reduce of incoming vectors is performed on the GPU side in batches that include the vectors from all peers or all clients. When the GPUDirect module is active, incoming vectors and outgoing vectors are received/sent directly to/from the GPU.

To make use of the GPU's capabilities, make sure to perform the following:

1. Refer to the NVIDIA DOCA Installation Guide for Linux for instructions on installing NVIDIA driver for CUDA and a CUDA-repo on your setup.
2. Create the sub-functions and configure the OVS according to Scalable Function Setup Guide.

8.1. Compiling and Running Application on Converged Accelerator

Since there is no pre-compiled allreduce application binary provided that uses GPU support, you must compile it and run it. All the sources needed for building, compiling, and running the application with GPU support are found under `/opt/mellanox/doca/applications/allreduce/src`.

To build and run the application:

1. Setup CUDA paths:

   ```
   export CPATH=/usr/local/cuda/targets/sbsa-linux/include:$CPATH
   export LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64:
   $LD_LIBRARY_PATH
   export PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:$PATH
   ```

2. Reinstall UCX with CUDA support. Follow the UCX installation procedure with an additional flag, `--with-cuda=/usr/local/cuda/`, passed to `configure-release`:

   ```
   dpu# ./contrib/configure-release --with-cuda=/usr/local/cuda/
   ```

3. To build the application with GPU support:

   a). Set the `enable_gpu_support` flag to `true` in `/opt/mellanox/doca/applications/meson_options.txt`.

   b). Compile the application sources. Run:

   ```
   cd /opt/mellanox/doca/applications/
   meson build
   ninja -C build
   ```

   `doca_allreduce_gpu` is created under `./build/allreduce/src/` alongside the regular `doca_allreduce` binary that is compiled without the GPU support.

4. To run the application with GPU support, follow the steps in Running the Application.

Chapter 9. References

- `/opt/mellanox/doca/applications/allreduce/src/`
- `/opt/mellanox/doca/applications/allreduce/bin/allreduce_client_params.json`
- `/opt/mellanox/doca/applications/allreduce/bin/allreduce_daemon_params.json`

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com