



# NVIDIA DOCA Allreduce

## Application Guide

# Table of Contents

Chapter 1. Introduction.....	1
Chapter 2. System Design.....	2
Chapter 3. Application Architecture.....	5
Chapter 4. DOCA Libraries.....	7
Chapter 5. Configuration Flow.....	8
Chapter 6. Running Application.....	11
Chapter 7. Arg Parser DOCA Flags.....	14
Chapter 8. References.....	16

---

# Chapter 1. Introduction

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.

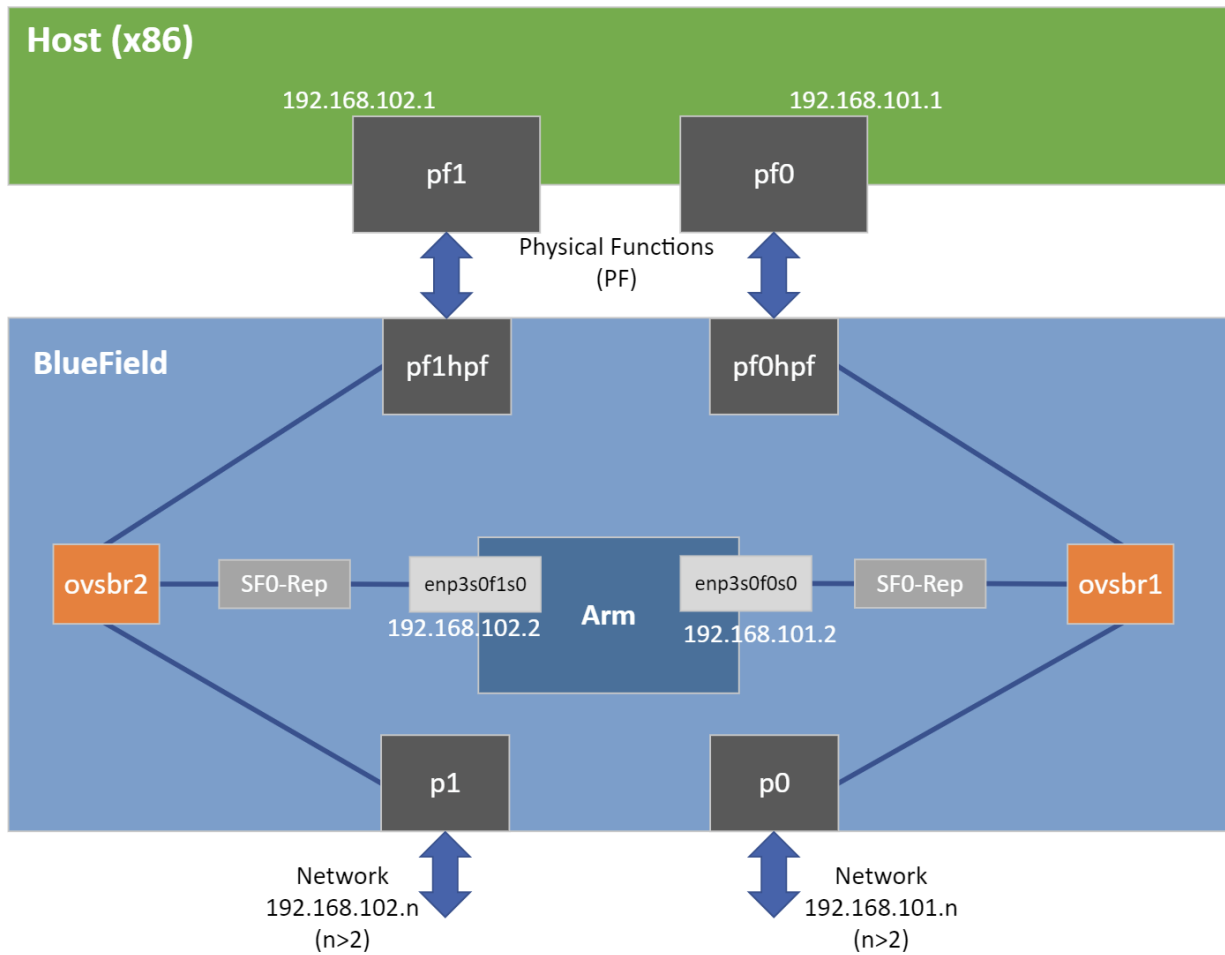
This document describes the following types of allreduce:

- ▶ 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

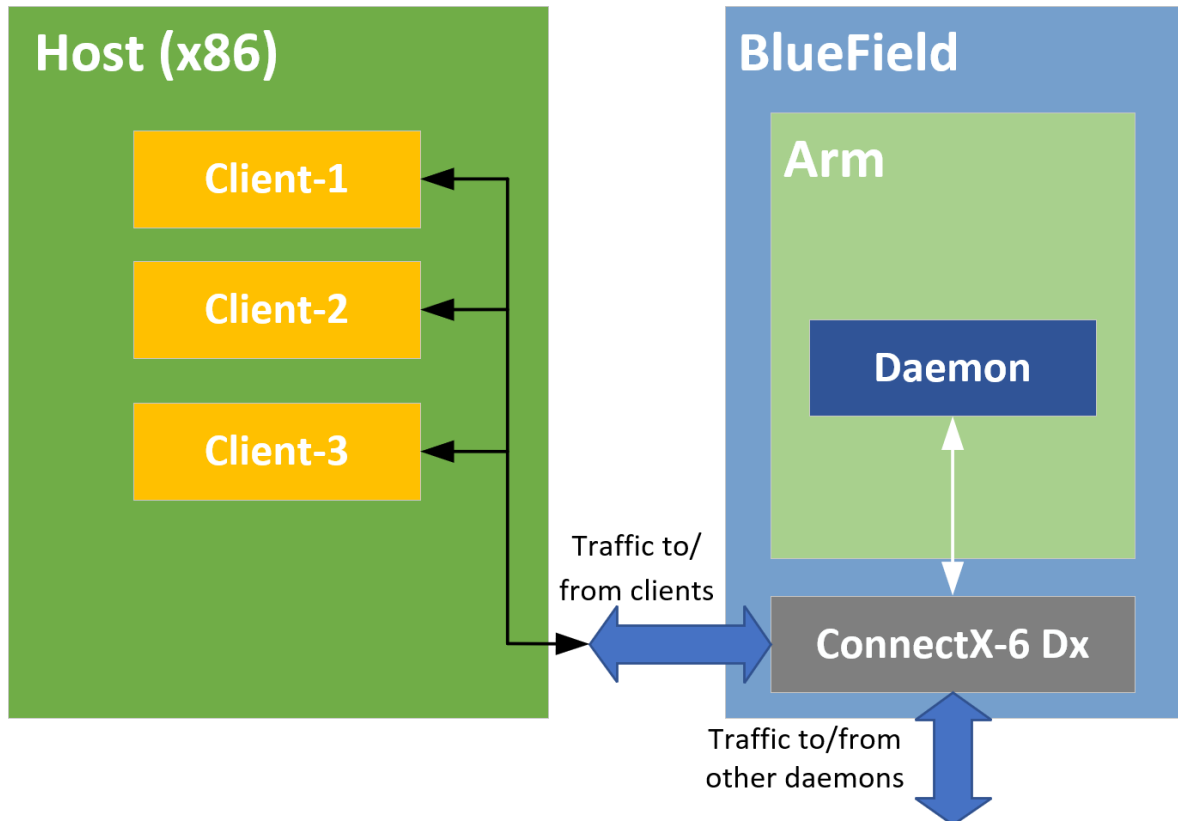
The application is designed to measure three metrics:

- ▶ 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.

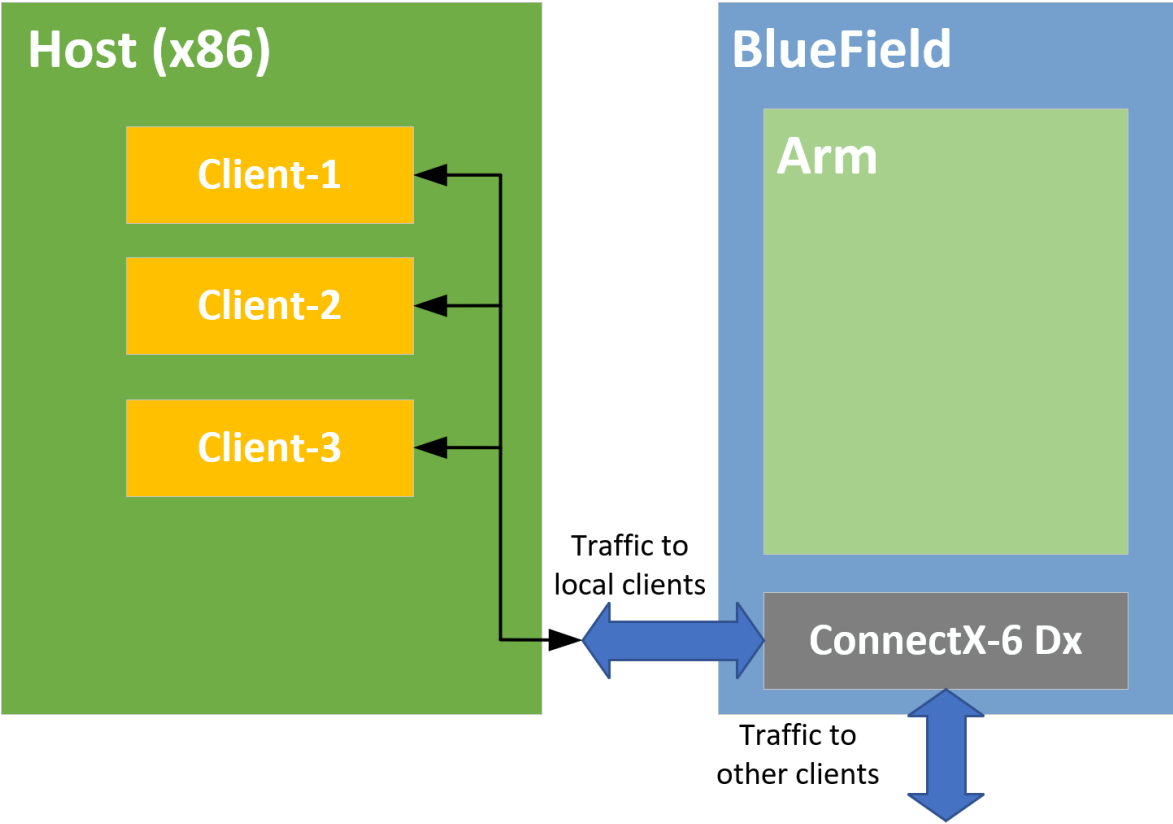


The allreduce implementation is divided into two different types of processes: clients and daemons. Clients are responsible for allocating vectors filled with data and initiating allreduce operations by sending a request with a vector to their daemon. Daemons are responsible for gathering vectors from 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



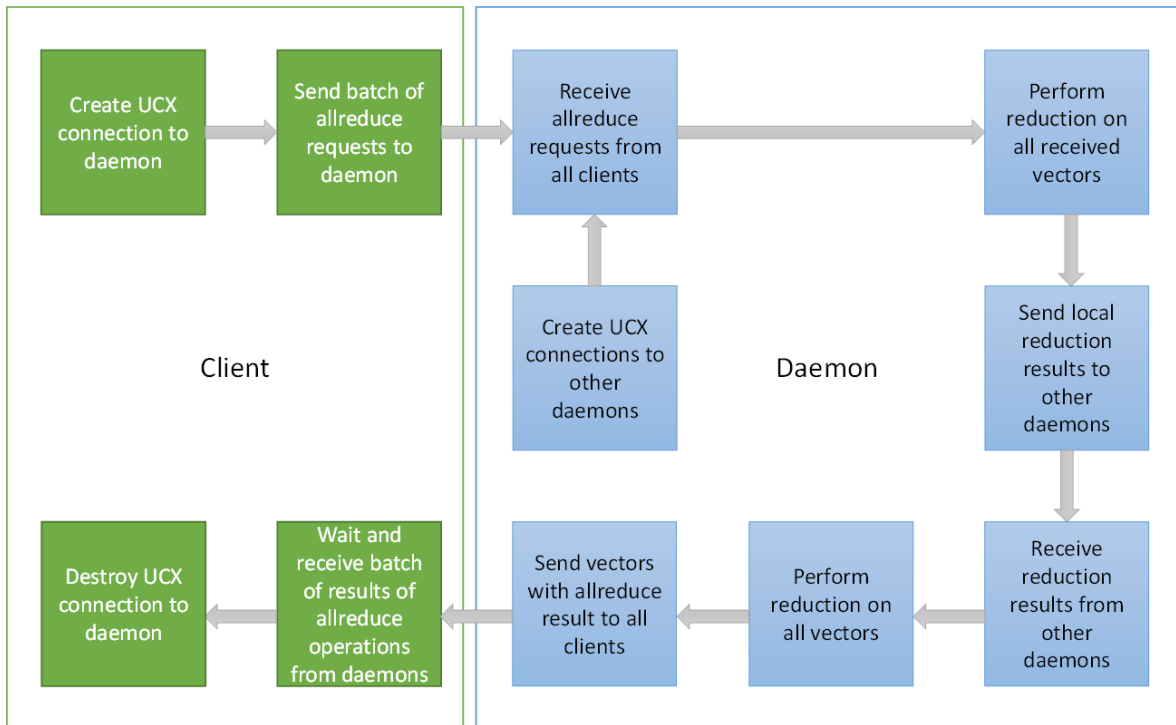
► Non-offloaded mode



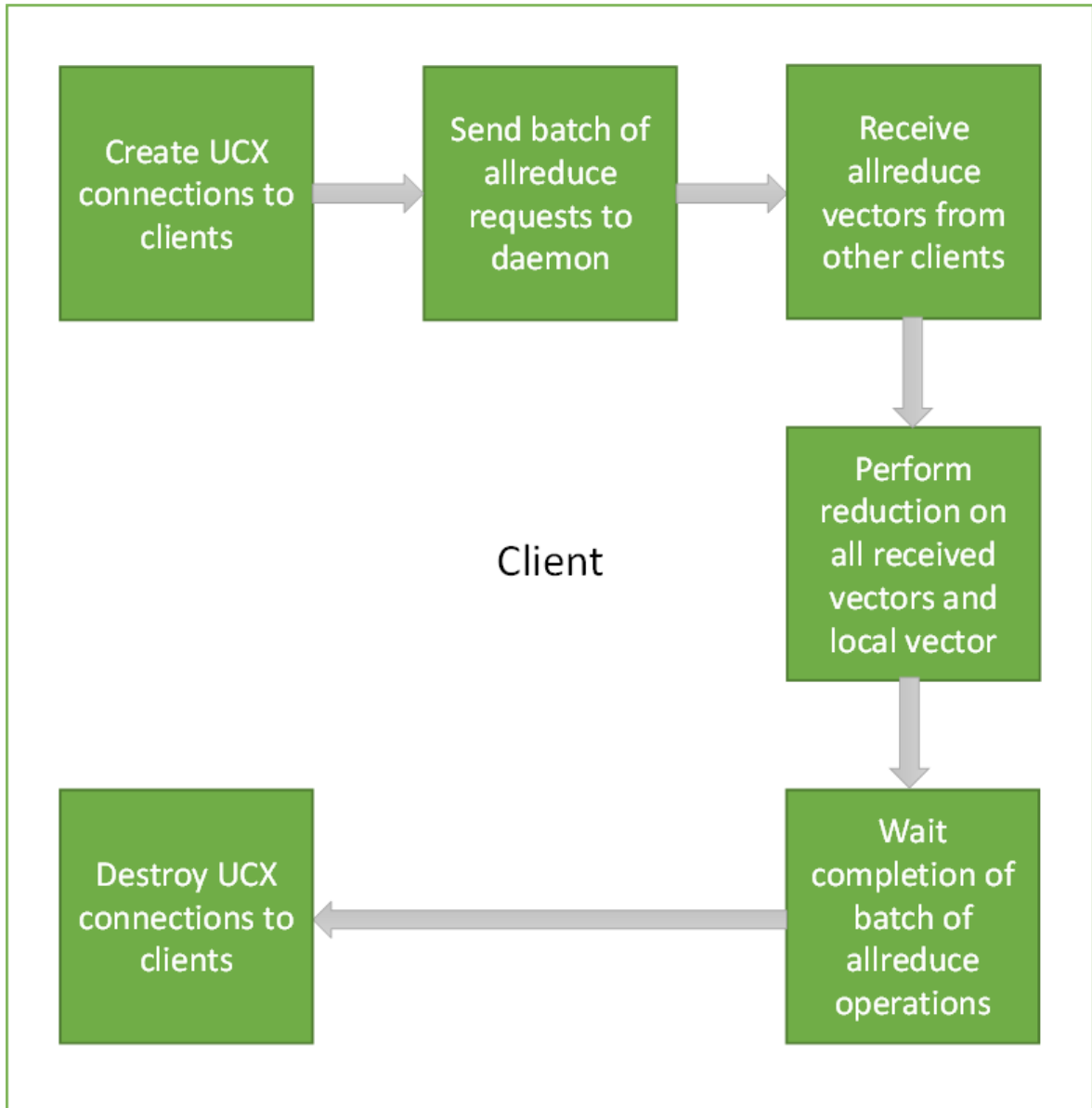
# Chapter 3. Application Architecture

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



- ▶ Non-offloaded mode



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.

```
doca_argp_init();
```

- a). Initialize arg parser resources and register DOCA general parameters.

```
register_allreduce_params();
```

- b). Register UCX allreduce application parameters.

```
doca_argp_start();
```

- c). Parse all registered parameters.

2. UCX initialization.

```
allreduce_ucx_init();
```

- a). Initialize hash table of connections.

```
ucp_init();
```

- b). Create UCP context.

```
ucp_worker_create();
```

- c). Create UCP worker.

```
ucp_worker_set_am_recv_handler();
```

- d). Set AM handler for receiving connection check packets.

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:

```
allreduce_ucx_am_set_recv_handler();
```

- i. Set AM handler for receiving allreduce requests from clients.

```
allreduce_ucx_listen();
```

- ii. Initialize UCX listening function. This creates a UCP listener.

```
connections_init();
```

- d). Initialize all connections.

- i. Go over all destination addresses.

- ii. Connect to each peer. 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();
allreduce_ucx_am_set_recv_handler();
```

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

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

b). Perform UCP worker progress.

c). Invoke callbacks for incoming messages by calling `allreduce_ucx_progress`.

5. Client.

```
client_run();
allreduce_vectors_init();
```

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

```
allreduce_ucx_am_set_recv_handler();
```

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

```
allreduce_barrier();
```

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

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

ii. Wait for completions.

```
allreduce_metrics_init();
```

d). Reset metrics and vectors.

e). Submit some batches and calculate estimated network time.

f). Allocate matrices to multiply.

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

h). Calculate the actual computation time of these matrix multiplications.

```
Do num-batches (flag) times:
    allreduce_vectors_reset();
    allreduce_batch_submit();
    cpu_exploit();
    allreduce_batch_wait();
    allreduce_metrics_calculate();
```

i). Reset vectors.

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

k). 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.

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

m). Update metrics.

```
allreduce_metrics_print();
```

n). 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.

```
ucp_ep_close_nbx();
```

## ii. Close inner UCP endpoint.

## 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.

```
g_hash_table_destroy();
```

## a). Destroy the hash table of the connections.

```
ucp_listener_destroy();
```

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

```
ucp_worker_destroy();
```

## c). Destroy UCP worker.

```
ucp_cleanup();
```

## d). Destroy UCP context.

---

# Chapter 6. Running Application

1. Refer to the following documents:

- ▶ [NVIDIA DOCA Installation Guide](#) 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.

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 the allreduce sample only:

a). Edit the following flags in `/opt/mellanox/doca/applications/meson_option.txt`:

- ▶ Set `enable_all_applications` to `false`
- ▶ Set `enable_allreduce` to `true`

b). Run the commands in step 2.



**Note:** The `doca_allreduce` application is created under `./build/allreduce/src/`.

## Application usage:

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

### DOCA Flags:

<code>-h, --help</code>	Print a help synopsis
<code>-v, --version</code>	Print program version information
<code>-l, --log-level</code>	Set the log level for the program <CRITICAL=0, DEBUG=4>

### Program Flags:

<code>-r, --role</code> or <code>"daemon"</code>	Run DOCA UCX allreduce process as: <code>"client"</code>
<code>-m, --mode &lt;allreduce_mode&gt;</code> (valid for client only)	Set allreduce mode: <code>"offloaded"</code> , <code>"non-offloaded"</code>
<code>-p, --port &lt;port&gt;</code>	Set default destination port of daemons/clients, used for IPs without a port (see <code>'-a'</code> flag)
<code>-c, --num-clients &lt;num_clients&gt;</code>	Set the number of clients which participate in allreduce operations (valid for daemon only)
<code>-s, --size &lt;size&gt;</code>	Set size of vector to do allreduce for
<code>-d, --datatype &lt;datatype&gt;</code>	Set datatype ( <code>"byte"</code> , <code>"int"</code> , <code>"float"</code> , <code>"double"</code> ) of vector elements to do allreduce for
<code>-o, --operation &lt;operation&gt;</code>	Set operation ( <code>"sum"</code> , <code>"prod"</code> ) 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)
-t, --listen-port <listen_port> Set listening port of daemon or client
-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:

- 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
```

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
```

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

```
BlueField> 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

BlueField> 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
```

- ▶ 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
-i 16 -b 128
```



**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
```



**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 User 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: <ul style="list-style-type: none"><li>▶ CRITICAL=0</li><li>▶ ERROR=1</li><li>▶ WARNING=2</li><li>▶ INFO=3</li><li>▶ DEBUG=4</li></ul>
	v	version	Print program version information
	h	help	Print a help synopsis
Program Flags	r	role	Run DOCA UCX allreduce process as either <code>client</code> or <code>daemon</code>
	m	mode	Set allreduce mode. Available types options: <ul style="list-style-type: none"><li>▶ <code>offloaded</code></li><li>▶ <code>non-offloaded</code> (valid for client only)</li></ul>
	p	port	Set default destination port of daemons/clients. Used for IPs without a port (see <code>-a</code> flag).
	c	num-clients	Set the number of clients which



Flag Type	Short Flag	Long Flag/JSON Key	Description
			participate in allreduce operations   <b>Note:</b> 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  <ul style="list-style-type: none"> <li>▶ byte</li> <li>▶ int</li> <li>▶ float</li> <li>▶ double</li> </ul>
	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. References

- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce.c`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_client.c`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_client.h`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_core.c`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_core.h`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_daemon.c`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_daemon.h`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_ucx.c`
- ▶ `/opt/mellanox/doca/applications/allreduce/src/allreduce_ucx.h`
- ▶ `/opt/mellanox/doca/applications/allreduce/bin/allreduce_client_params.json`
- ▶ `/opt/mellanox/doca/applications/allreduce/bin/allreduce_daemon_params.json`

## Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## Trademarks

NVIDIA, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

## Copyright

© 2022 NVIDIA Corporation & affiliates. All rights reserved.