



NVIDIA DOCA Allreduce Application Guide

Table of contents

Introduction

System Design

Application Architecture

DOCA Libraries

Running the Application

Installation

Application Execution

Command Line Flags

Troubleshooting

Recompiling the Application

Recompiling All Applications

Recompiling Allreduce Application Only

Troubleshooting

Running Application on NVIDIA Converged Accelerator

Compiling and Running Application

Application Code Flow

References

This guide provides a DOCA Allreduce collective operation implementation on top of NVIDIA® BlueField® DPU using UCX.

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:

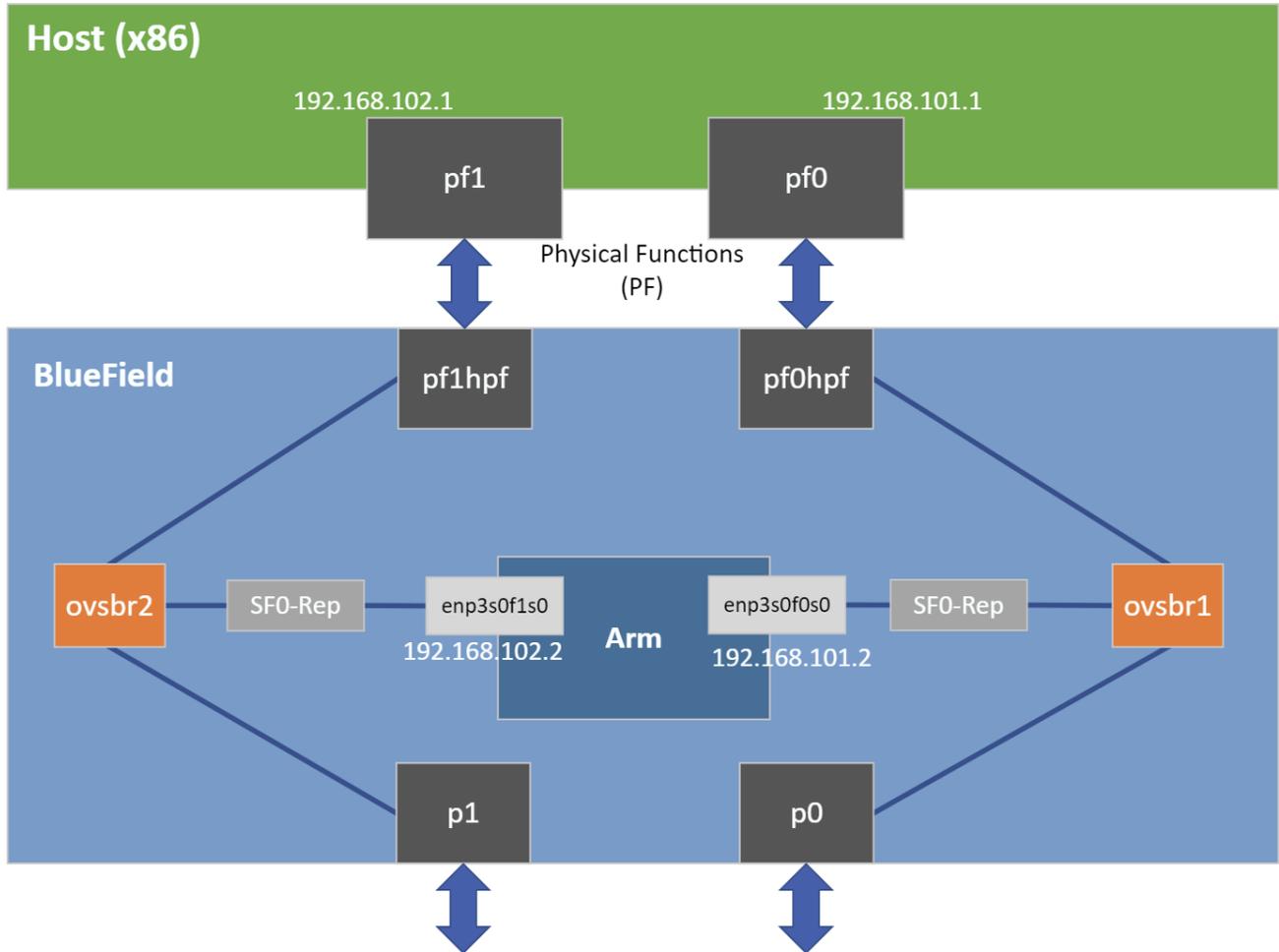
- Offloaded client – processes running on the host which only submit allreduce operation requests to a daemon running on the DPU. The daemon runs on the DPU and performs the allreduce algorithm on behalf of its on-host-clients (offloaded-client).
- Non-offloaded client – processes running on the host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

System Design

The application is designed to measure three metrics:

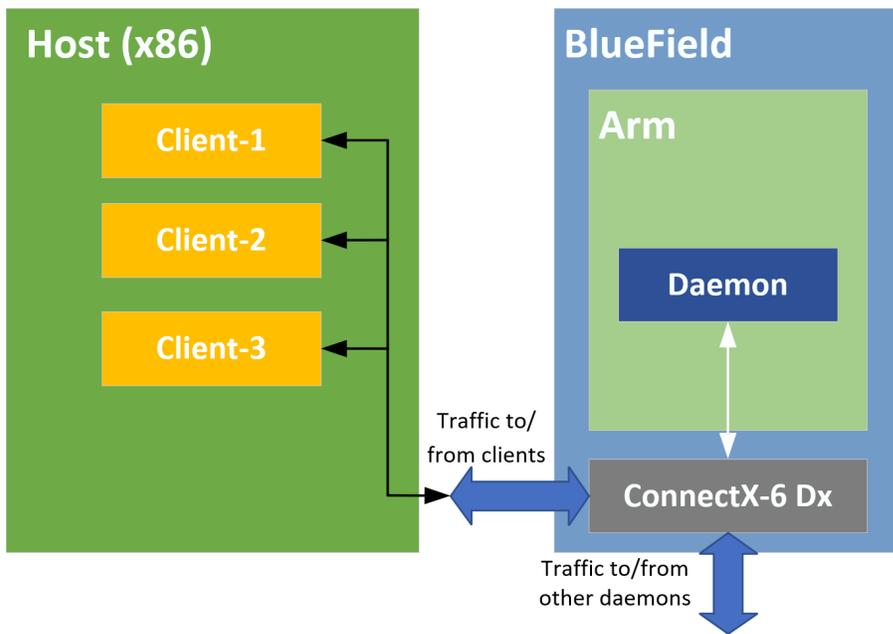
- Communication time taken by offloaded and non-offloaded allreduce operations

- 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.

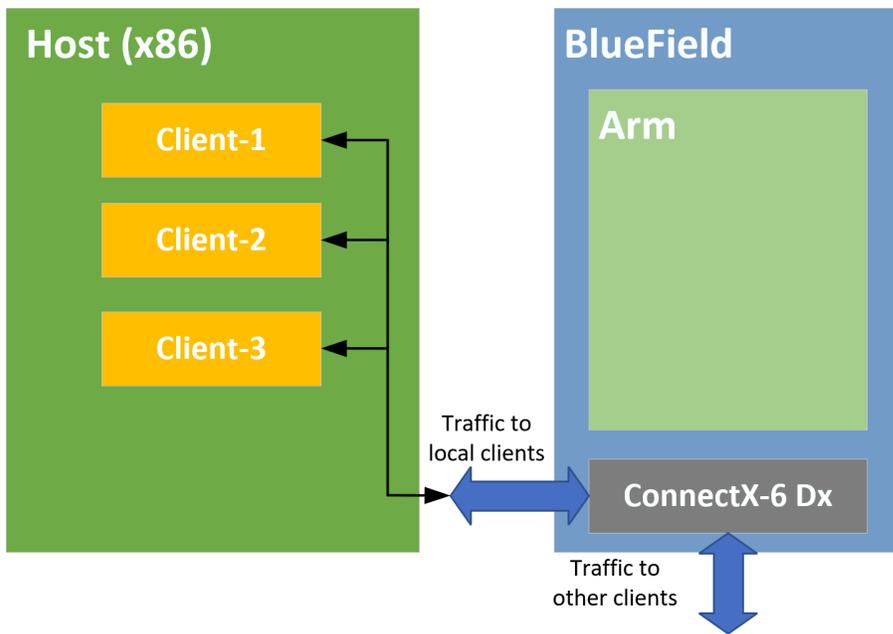


The allreduce implementation is divided into two different types of processes: clients and daemons. Clients are responsible for allocating vectors filled with data and initiating allreduce operations by sending a request with a vector to their daemon. Daemons are responsible for gathering vectors from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



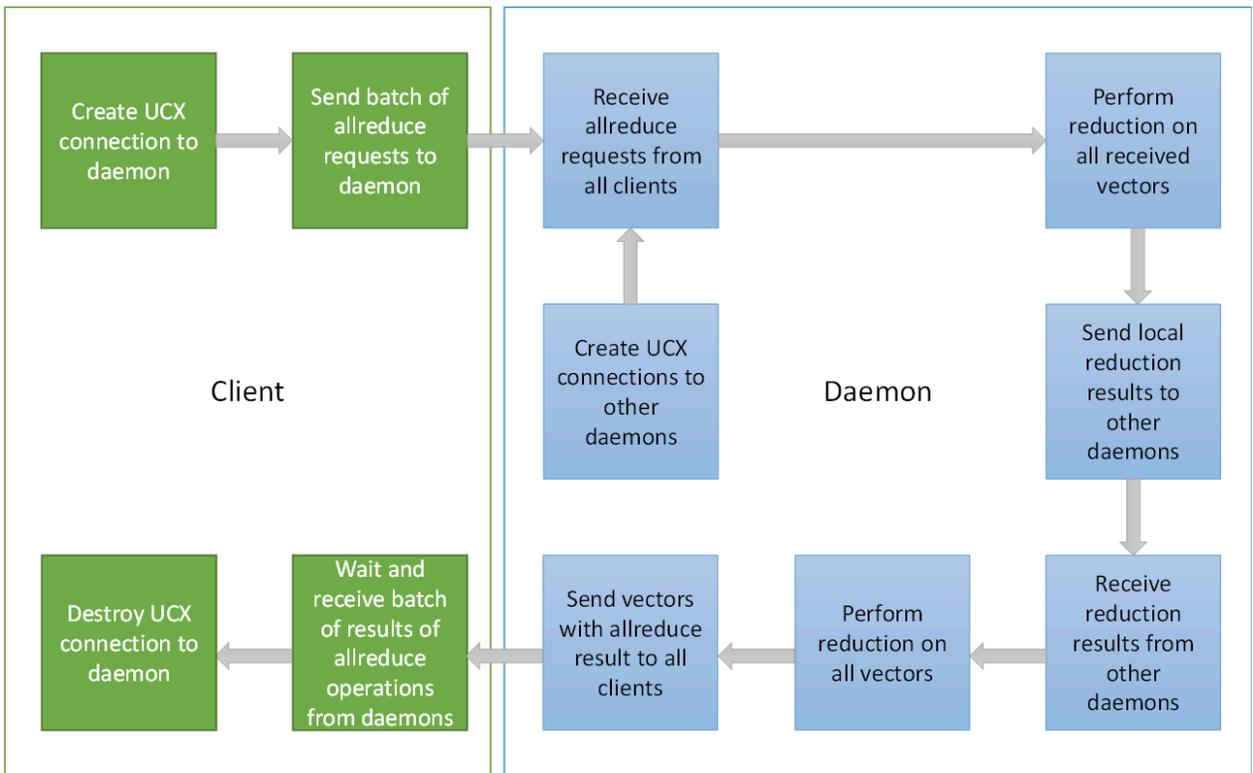
- Non-offloaded mode



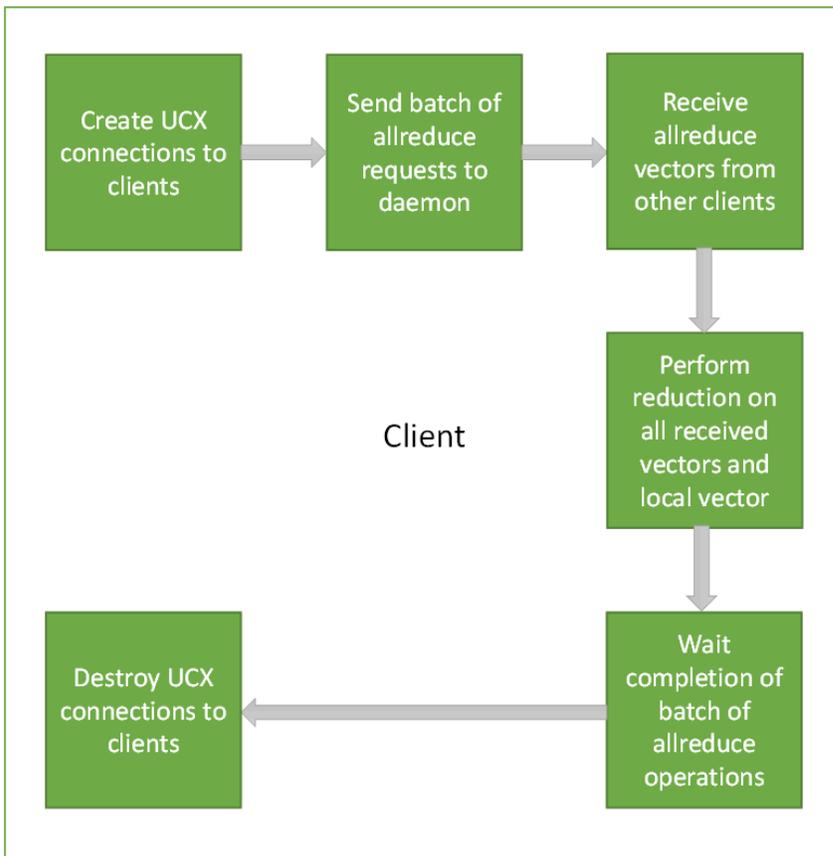
Application Architecture

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.

DOCA Libraries

This application leverages the UCX framework DOCA driver.

Running the Application

Installation

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

Application Execution

The allreduce application is provided in both source and binary forms. The binary is located under

```
/opt/mellanox/doca/applications/allreduce/bin/doca_allreduce.
```

1. Application usage instructions:

```
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 (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
 --sdk-log-level Set the SDK (numeric) log level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR, 40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
 -j, --json <path> Parse all command flags from an input json file

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
```

i Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
/opt/mellanox/doca/applications/allreduce/bin/doca_allrec  
-h
```

i Info

For additional information, refer to section "[Command Line Flags](#)".

2. Configuration steps.

1. All daemons should be deployed before clients. Only after connecting to their peers are daemons able to handle clients.
2. 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:

1. 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_all
-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_all
-r daemon -t 34001 -c 1 -s 100 -o sum -d float -b 16 -i
16
```

2. 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:
/opt/mellanox/doca/applications/allreduce/bin/doca_all
```

```
-r daemon -t 34001 -c 1 -s 100 -o sum -d float -b 16 -i 16
```

3. CLI example for running the daemon on BlueField:

```
/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
```

Notes:

- 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.
- The flag `-c` must be specified for daemon processes only. It indicates how many clients submit their allreduce operations to the daemon.
- The flags `-s`, `-i`, `-b`, and `-d` must be the same for all clients and daemons participating in the allreduce operation.

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.

4. CLI example for running the client on the host:

```
/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
```

5. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
doca_allreduce --json [json_file]
```

For example:

```
cd /opt/mellanox/doca/applications/allreduce/bin
./doca_allreduce --json ./allreduce_client_params.json
```

Note

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the desired PCIe and network addresses required for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description
General flags	<code>h</code>	<code>help</code>	Print a help synopsis
	<code>v</code>	<code>version</code>	Print program version information
	<code>l</code>	<code>log-level</code>	Set the log level for the application: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with <code>TRACE</code> log level support)
	N/A	<code>sdk-log-level</code>	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70
	<code>j</code>	<code>json</code>	Parse all command flags from an input JSON file
Program flags	<code>r</code>	<code>role</code>	Run DOCA UCX allreduce process as either <code>client</code> or <code>daemon</code>
	<code>m</code>	<code>mode</code>	Set allreduce mode. Available types options: <ul style="list-style-type: none"> • <code>offloaded</code> • <code>non-offloaded</code> (valid for client only)
	<code>p</code>	<code>port</code>	Set default destination port of daemons/clients. Used for IPs without a port (see <code>-a</code> flag).

Flag Type	Short Flag	Long Flag/JSON Key	Description
	<code>c</code>	<code>num-clients</code>	Set the number of clients which participate in allreduce operations Note: Valid for daemon only.
	<code>s</code>	<code>size</code>	Set size of vector to perform allreduce for
	<code>d</code>	<code>datatype</code>	Set datatype of vector elements to do allreduce for <ul style="list-style-type: none"> • <code>byte</code> • <code>int</code> • <code>float</code> • <code>double</code>
	<code>o</code>	<code>operation</code>	Set operation to perform between allreduce vectors
	<code>b</code>	<code>batch-size</code>	Set the number of allreduce operations submitted simultaneously. Used for handshakes by daemons.
	<code>i</code>	<code>num-batches</code>	Set the number of batches of allreduce operations. Used for handshakes by daemons.
	<code>t</code>	<code>listen-port</code>	Set listening port of daemon or client
	<code>a</code>	<code>addresses</code>	Set comma-separated list of destination IPv4/IPv6 address and ports optionally of daemons or clients. Format: <code><ip_addr>:[<port>]</code> .

i Info

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

Recompiling the Application

In addition to providing the application in binary form, the installation also includes all of the application sources and compilation instructions so as to allow modifying the sources and recompiling the application. For more information about the applications, as well as development and compilation tips, please refer to the [DOCA Applications](#) page.

The sources of the application can be found under the `/opt/mellanox/doca/applications/allreduce/src` directory.

Recompiling All Applications

The applications are all defined under a single meson project, so the default compilation recompiles all the DOCA applications.

To build all the applications together, run:

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

Info

`doca_allreduce` is created under `/tmp/build/allreduce/src/`.

Recompiling Allreduce Application Only

To build the allreduce application only:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -  
Denable_allreduce=true  
ninja -C /tmp/build
```

Info

`doca_allreduce` is created under
`/tmp/build/allreduce/src/`.

Alternatively, the user can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in

```
/opt/mellanox/doca/applications/meson_options.txt:
```

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

2. Run the following compilation commands:

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

Info

```
doca_allreduce is created under  
/tmp/build/allreduce/src/.
```

Troubleshooting

Please refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

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 [NVIDIA BlueField DPU Scalable Function User Guide](#).

Compiling and Running Application

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:$CPATH  
export  
LD_LIBRARY_PATH=/usr/local/nvidia/lib:/usr/local/nvidia/lib64:  
export PATH=/usr/local/nvidia/bin:/usr/local/cuda/bin:$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:

1. Set the `enable_gpu_support` flag to `true` in `/opt/mellanox/doca/applications/meson_option.txt`.

2. Compile the application sources. Run:

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

`doca_allreduce_gpu` is created under `/tmp/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 same steps as described in section "[Running the Application](#)".

Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register UCX allreduce application parameters.

```
register_allreduce_params();
```

3. Parse all registered parameters.

```
doca_argp_start();
```

2. UCX initialization.

1. Initialize hash table of connections.

```
allreduce_ucx_init();
```

2. Create UCP context.

```
ucp_init();
```

3. Create UCP worker.

```
ucp_worker_create();
```

4. Set AM handler for receiving connection check packets.

```
ucp_worker_set_am_recv_handler();
```

3. Initialization of the allreduce connectivity.

```
communication_init();
```

1. Initialize hash table of allreduce super requests.
2. Set "receive callback" for handshake messages.
3. If daemon or non-offloaded client:
 1. Set AM handler for receiving allreduce requests from clients.

```
allreduce_ucx_am_set_recv_handler();
```

2. Initialize UCX listening function. This creates a UCP listener.

```
allreduce_ucx_listen();
```

4. Initialize all connections.

```
connections_init();
```

1. Go over all destination addresses and connect to each peer.
2. Repeat until a successful send occurs (to check connectivity).

```
ucp_am_send_nbx();  
allreduce_ucx_request_wait();
```

3. Insert the connection to the hash table of connections.

```
allreduce_outgoing_handshake();
```

5. 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();
```

1. Set AM handler to receive allreduce requests from clients.

```
allreduce_ucx_am_set_recv_handler();
```

2. Perform UCP worker progress.

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

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

5. Client:

```
client_run();
```

1. Allocate buffers to store allreduce initial data and results.

```
allreduce_vectors_init();
```

2. Set an AM handler for receiving allreduce results.

```
allreduce_ucx_am_set_recv_handler();
```

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

```
allreduce_barrier();
```

1. Submit a batch of allreduce operations with 0 byte.
 2. Wait for completions.
4. Reset metrics and vectors.

```
allreduce_metrics_init();
```

1. Submit some batches and calculate estimated network time.
2. Allocate matrices to multiply.
3. Estimate how many matrix multiplications could have been performed instead of networking (same time window).

4. Calculate the actual computation time of these matrix multiplications.
5. Reset vectors.
6. Submit a batch of allreduce operations to daemon/peer (depends on mode).
7. 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.
8. Wait for the allreduce operation to complete and calculate time cost.
9. Update metrics.

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

10. Print summary of allreduce benchmarking.

```
allreduce_metrics_print();
```

6. Arg parser destroy.

```
doca_argp_destroy();
```

7. Communication destroy.

1. Clean up connections.

```
allreduce_ucx_disconnect();
```

1. Remove the connection from the hash table of the connections.
2. Close inner UCP endpoint.

```
ucp_ep_close_nbx();
```

3. Wait for the completion of the UCP endpoint closure.
4. Destroy connection.
5. Free connections array.

2. Destroy the hash table of the allreduce super requests.

8. Destroy UCX context.

1. Destroy the hash table of the connections.

```
g_hash_table_destroy();
```

2. If the UCP listener was created, destroy it.

```
ucp_listener_destroy();
```

3. Destroy UCP worker.

```
ucp_worker_destroy();
```

4. Destroy UCP context.

```
ucp_cleanup();
```

References

- `/opt/mellanox/doca/applications/allreduce/src`
- `/opt/mellanox/doca/applications/allreduce/bin/allreduce_client_pa`
- `/opt/mellanox/doca/applications/allreduce/bin/allreduce_daemon_pa`

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 (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes 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 NONINFRINGEMENT, 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 and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2024, NVIDIA. PDF Generated on 01/15/2025