



NVIDIA DOCA DPA All-to-all Application Guide

Application Guide

Table of Contents

Chapter 1. Introduction.....	1
Chapter 2. System Design.....	2
Chapter 3. Application Architecture.....	3
Chapter 4. DOCA Libraries.....	4
Chapter 5. Configuration Flow.....	5
Chapter 6. Dependencies.....	8
Chapter 7. Running the Application.....	9
Chapter 8. Arg Parser DOCA Flags.....	11
Chapter 9. References.....	13

Chapter 1. Introduction

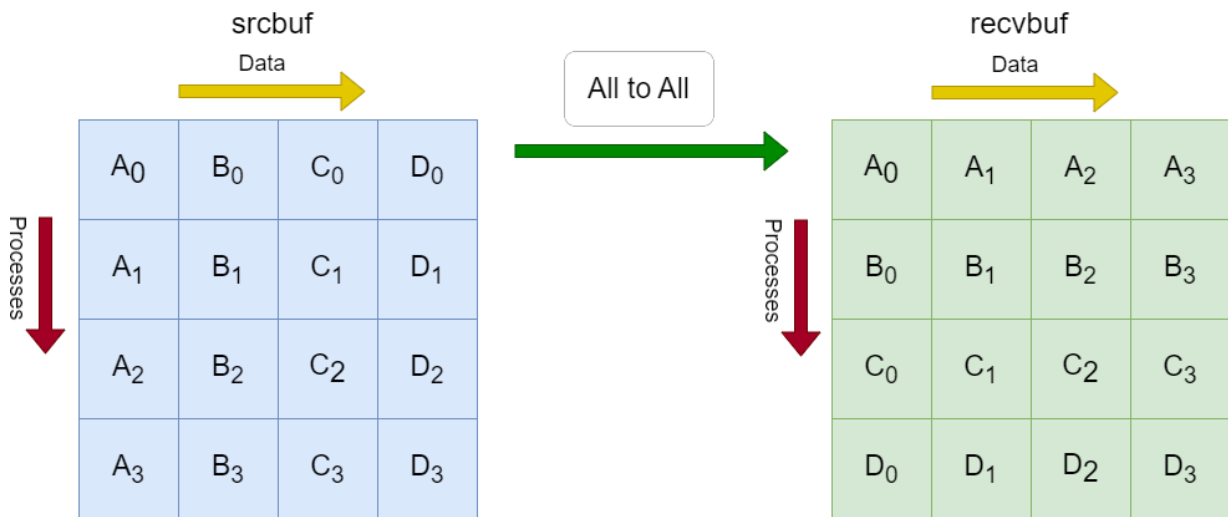
This example shows how the MPI all-to-all collective can be accelerated on the DPA. In an MPI collective, all processes in the same job call the collective routine.

Given a communicator of n ranks, the example performs a collective operation in which all the processes send and receive the same amount of data from all the processes (hence all-to-all).

This document describes how to run the all-to-all example using DOCA DPA.

Chapter 2. System Design

All-to-all is a message passing interface (MPI) method. MPI is a standardized and portable message-passing standard designed to function on parallel computing architectures. An MPI program is one where several processes run in parallel.

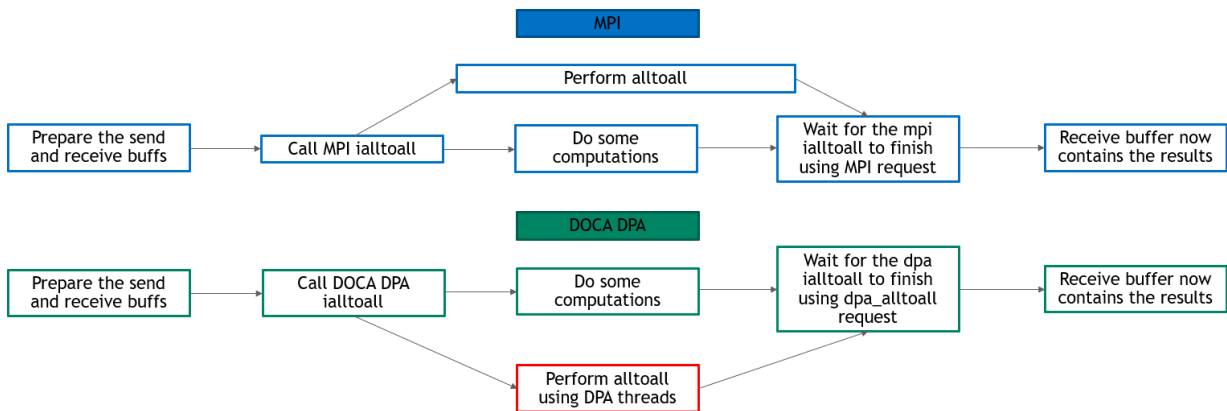


Each process in the diagram divides its local sendbuf into n blocks (4 in this example), each containing sendcount (4 in this example) elements. Process i sends the k -th block of its local sendbuf to process k which places the data in the i -th block of its local recvbuf.

Implementing the all-to-all method using DOCA DPA offloads the copying of the elements from the srcbuf to the recvbufs to the DPA, and leaves the CPU free to perform other computations.

Chapter 3. Application Architecture

The following diagram describes the differences between the host-based all-to-all and DPA all-to-all.



- ▶ In DPA all-to-all, DPA threads perform the all-to-all and the CPU is free to do other computations
- ▶ In host-based all-to-all, the CPU must still perform the all-to-all at some point and is not completely free for other computations

Chapter 4. DOCA Libraries

This application leverages the following DOCA driver:

- ▶ [DPA library](#)

Chapter 5. Configuration Flow

This section lists the application's configuration flow which includes different FlexIO functions and wrappers.

1. Initialize MPI.

```
MPI_Init(&argc, &argv);
```

2. Parse application argument.

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

```
doca_argp_init();
```

b). Register the application's parameters.

```
register_all_to_all_params();
```

c). Parse the arguments.

```
doca_argp_start();
```

- i. The `msgsize` parameter is the size of the sendbuf and recvbuf (in bytes). It must be in multiples of an integer and at least the number of processes times an integer size.
- ii. The `devices_param` parameter is the names of the InfiniBand devices to use (must support DPA). It can include up to two devices names.

d). Only let the first process (of rank 0) parse the parameters to then broadcast them to the rest of the processes.

3. Check and prepare the needed resources for the `all_to_all` call:

- ▶ Check the number of processes (maximum is 16).
- ▶ Check the `msgsize`. It must be in multiples of integer size and at least the number of processes times integer size.
- ▶ Allocate the sendbuf and recvbuf according to `msgsize`.

4. Prepare the resources required to perform the all-to-all method using DOCA DPA:

a). Initialize DOCA DPA context:

i. Open DOCA DPA device (DOCA device that supports DPA).

```
open_dpa_device();
```

ii. Create DOCA DPA context using the opened device.

```
doca_dpa_create();
```

- b). Create the required events for the all-to-all: One completion event for the kernel launch (wait location CPU and update location DPA) and kernel events (wait location remote and update location DPA) as the number of processes.

```
create_dpa_a2a_events() {
    doca_dpa_event_create(doca_dpa, DOCA_DPA_EVENT_ACCESS_DPA,
        DOCA_DPA_EVENT_ACCESS_CPU, DOCA_DPA_EVENT_WAIT_DEFAULT, &comp_event, 0);
    for (i = 0; i < resources->num_ranks; i++)
        doca_dpa_event_create(doca_dpa, DOCA_DPA_EVENT_ACCESS_REMOTE,
            DOCA_DPA_EVENT_ACCESS_DPA, DOCA_DPA_EVENT_WAIT_DEFAULT, &(kernel_events[i]),
            0);
}
```

- c). Create DOCA DPA worker (for the endpoints).

```
doca_dpa_worker_create();
```

- d). Prepare DOCA DPA endpoints:

- i. Create DOCA DPA endpoints as the number of processes/ranks.

```
for (i = 0; i < resources->num_ranks; i++)
    doca_dpa_ep_create();
```

- ii. Connect the local process' endpoints to the other processes' endpoints.

```
connect_dpa_a2a_endpoints();
```

- iii. Export the endpoints to DOCA DPA device endpoints (so they can be used by the DPA) and copy them to DPA heap memory.

```
for (int i = 0; i < resources->num_ranks; i++) {
    result = doca_dpa_ep_dev_export();
    doca_dpa_mem_alloc();
    doca_dpa_h2d_memcpy();
}
```

- e). Prepare the memory required to perform the all-to-all method using DOCA DPA. This includes creating memory handlers for the sendbuf and recvbuf, getting the other processes' recvbufs handlers, and copying these memory handlers and their remote keys and the events' handlers to the DPA heap memory.

```
prepare_dpa_a2a_memory();
```

5. Launch the `alltoall_kernel` using DOCA DPA kernel launch with all the required parameters:

- a). Every MPI rank launches a kernel of up to `MAX_NUM_THREADS`. This example defines `MAX_NUM_THREADS` as 16.

- b). Launch `alltoall_kernel` using `kernel_launch`.

```
doca_dpa_kernel_launch();
```

- c). Using the `doca_dpa_dev_put_signal_nb()` function, copy the relevant sendbuf to the correct recvbuf (according to the process' rank) for every OS process. Remember that multithreading is also used inside of the DPA threads.

```
for (i = thread_rank; i < num_ranks; i += num_threads)
    doca_dpa_dev_put_signal_nb();
```

- d). Wait until the `alltoall_kernel` has finished.

```
doca_dpa_event_wait_until();
```



Note: Add an MPI barrier after waiting for the event to make sure that all of the processes have finished executing the `alltoall_kernel`.

```
MPI_Barrier();
```


After the `alltoall_kernel` is finished, the `recvbuf` of all the processes now contain the expected output of the all-to-all method.

6. Destroy the `a2a_resources`:

- a). Free all the DOCA DPA memories.

```
doca_dpa_mem_free();
```

- b). Unregister all the DOCA DPA host memories.

```
doca_dpa_mem_unregister();
```

- c). Destroy all the DOCA DPA endpoints.

```
doca_dpa_ep_destroy();
```

- d). Destroy the DOCA DPA worker.

```
doca_dpa_worker_destroy();
```

- e). Destroy all the DOCA DPA events.

```
doca_dpa_event_destroy();
```

- f). Destroy the DOCA DPA context.

```
doca_dpa_destroy();
```

- g). Close the DOCA device.

```
doca_dev_close();
```

Chapter 6. Dependencies

- ▶ BlueField-3 or later
- ▶ Ubuntu 18.04/20.04/22.04 on the host (x86)
- ▶ Open MPI version 4.1.5rc2 or greater (included in DOCA installation)

Chapter 7. 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 `doca_dpa_all_to_all` binary is located under `/opt/mellanox/doca/applications/dpa_all_to_all/bin/doca_dpa_all_to_all`. To build all the applications together, run:

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

3. To build only the `dpa_all_to_all` application:

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

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

b). Run the commands in step 2.



Note: `doca_dpa_all_to_all` is created under `./build/dpa_all_to_all/src/host/`.

Application usage:

```
Usage: doca_dpa_all_to_all [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:

```
-m, --msgsize <Message size> The message size - the size of the sendbuf  
and recvbuf (in bytes). Must be in multiples of integer size. Default is size  
of one integer times the number of processes.  
-d, --devices <IB device names> IB devices names that supports DPA,  
separated by comma without spaces (max of two devices). If not provided then a  
random IB device will be chosen.
```

4. Pre-run setup:

- ▶ Make sure that MPI is installed in your setup (`openmpi` is provided as part of the `doca-tools` metapackage). Do not forget to update your `LD_LIBRARY_PATH` and `PATH` environment variable to include MPI. For example, if MPI is installed under `/usr/mpi/gcc/openmpi-4.1.5rc2/` then run:

```
export PATH=/usr/mpi/gcc/openmpi-4.1.5rc2/bin:$PATH
export LD_LIBRARY_PATH=/usr/mpi/gcc/openmpi-4.1.5rc2/lib:$LD_LIBRARY_PATH
```

5. CLI example for running the application. Remember, this is an MPI program, so use `mpirun` to run the application (with the `-np` flag to specify the number of processes to run). The same command works for running the application on the host (x86) and the BlueField (Arm cores).

- ▶ The following runs the DPA all-to-all application with 8 processes using the default message size (the number of processes, which is 8, times the size of 1 integer) with a random InfiniBand device:

```
mpirun -np 8 /opt/mellanox/doca/applications/dpa_all_to_all/bin/
doca_dpa_all_to_all
```

- ▶ The following runs DPA all-to-all application with 8 processes, with 128 bytes as the message size and with `mlx5_0` and `mlx5_1` as the IB devices:

```
mpirun -np 8 /opt/mellanox/doca/applications/dpa_all_to_all/bin/
doca_dpa_all_to_all -m 128 -d "mlx5_0,mlx5_1"
```



Note: The application supports running with a maximum of 16 processes. If you try to run with more processes, then an error is printed and the application exits.

6. To run `doca_all_to_all_dpa` using a JSON file:


```
doca_dpa_all_to_all --json [json_file]
```

For example:

```
cd /opt/mellanox/doca/applications/dpa_all_to_all/bin
./doca_dpa_all_to_all --json ./dpa_all_to_all_params.json
```

Chapter 8. Arg Parser DOCA Flags

Refer to [NVIDIA DOCA Arg Parser Programming Guide](#) for more information.

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
General flags	l	log-level	Sets the log level for the application: <ul style="list-style-type: none"> ▶ CRITICAL=20 ▶ ERROR=30 ▶ WARNING=40 ▶ INFO=50 ▶ DEBUG=60 	<code>"log-level": 60</code>
	v	version	Prints program version information	N/A
	h	help	Prints a help synopsis	N/A
Program flags	m	msgsize	The message size. The size of the sendbuf and recvbuf (in bytes). Must be in multiples of an integer. The default is size of 1 integer times the number of processes.	<code>"msgsize": -1</code> <div style="border: 1px solid gray; background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p> Note: The value -1 is a placeholder to use the default size, which is only known at run time (because it depends on the number</p> </div>

Flag Type	Short Flag	Long Flag/ JSON Key	Description	JSON Content
	d	device	InfiniBand devices names that support DPA, separated by comma without spaces (max of two devices). If <code>NOT_SET</code> then a random InfiniBand device is chosen.	<div style="background-color: #cccccc; padding: 2px;">[REDACTED] of processes).</div> <pre>"devices": "NOT_SET"</pre>

Chapter 9. References

- ▶ `/opt/mellanox/doca/applications/all_to_all_dpa/src`
- ▶ `/opt/mellanox/doca/applications/all_to_all_dpa/bin/
all_to_all_dpa_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

© 2023 NVIDIA Corporation & affiliates. All rights reserved.