NVIDIA DOCA DMA Copy Application Guide
# Table of contents

- **Introduction**
- **System Design**
- **Application Architecture**
- **DOCA Libraries**
- **Compiling the Application**
  - Compiling All Applications
  - Compiling Only the Current Application
  - Troubleshooting
- **Running the Application**
  - Application Execution
  - Command Line Flags
  - Troubleshooting
- **Application Code Flow**
- **References**
This guide provides an example of a DMA Copy implementation on top of NVIDIA® BlueField® DPU.

**Introduction**

DOCA DMA (direct memory access) Copy application transfers files (data path), up to the maximum supported size by the hardware, between the DPU and the x86 host using the DOCA DMA Library which provides an API to copy data between DOCA buffers using hardware acceleration, supporting both local and remote memory.

DOCA DMA allows complex memory copy operations to be easily executed in an optimized, hardware-accelerated manner.

**System Design**

DOCA DMA Copy is designed to run on the instances of the BlueField DPU and x86 host. The DPU application must be the first to spawn as it opens the DOCA Comch server between the two sides on which all the necessary DOCA DMA library configuration files (control path) are transferred.
Application Architecture

DOCA DMA Copy runs on top of DOCA DMA to read/write directly from the host's memory without any user/kernel space context switches, allowing for a fast memory copy.

Flow:

1. The two sides initiate a short negotiation in which the file size and location are determined.

2. The host side creates the export descriptor with `doса_mmap_export_pci()` and sends it with the local buffer address and length on the Comch to the DPU side application.
3. The DPU side application uses the received export descriptor to create a remote memory map locally with `doca_mmap_create_from_export()` and the host buffer information to create a remote DOCA buffer.

4. From this point on, the DPU side application has all the necessary memory information and the DMA copy can take place.

**DOCA Libraries**

This application leverages the following DOCA libraries:

- **DOCA DMA**
- **DOCA Comch**

Refer to their respective programming guide for more information.

**Compiling the Application**

ℹ️ **Info**

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

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

⚠️ **Tip**

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications page](#).
The sources of the application can be found under the application’s directory: /opt/mellanox/doca/applications/dma_copy/.

### Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

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

**Info**

doca_dma_copy is created under /tmp/build/dma_copy/.

### Compiling Only the Current Application

To directly build only the DMA Copy application:

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

**Info**
Alternatively, one 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_dma_copy` to `true`

2. Run the following compilation commands:

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

**Troubleshooting**

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

**Running the Application**
Application Execution

The DMA Copy application is provided in source form. Therefore, a compilation is required before the application can be executed.

1. Application usage instructions:

Usage: doca_dma_copy [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:
- -f, --file Full path to file to be copied/created after a successful DMA copy
- -p, --pci-addr DOCA Comm Channel device PCI address
- -r, --rep-pci DOCA Comm Channel device representor PCIe address (needed only on DPU)

Info

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

./doca_dma_copy -h
2. CLI example for running the application on the BlueField:

```
./doca_dma_copy -p 03:00.0 -r 3b:00.0 -f received.txt
```

**Note**

Both the DOCA Comch device PCIe address (03:00.0) and the DOCA Comch device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_dma_copy -p 3b:00.0 -f send.txt
```

**Note**

The DOCA Comch device PCIe address, 3b:00.0, should match the address of the desired PCIe device.

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

```
./doca_dma_copy --json [json_file]
```

```
./doca_dma_copy --json ./dma_copy_params.json
```

⚠️ Note

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

---

## Command Line Flags

<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genera l flags</td>
<td>h</td>
<td>help</td>
<td>Print a help synopsis</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>version</td>
<td>Print program version information</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>log-level</td>
<td>Set the log level for the application:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DISABLE=10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CRITICAL=20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ERROR=30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• WARNING=40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• INFO=50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DEBUG=60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• TRACE=70 (requires compilation with TRACE log level support)</td>
<td></td>
</tr>
</tbody>
</table>

"log-level": 60
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td></td>
<td>sdk-log-level</td>
<td>Set the log level for the program:</td>
<td>&quot;sdk-log-level&quot;: 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DISABLE=10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CRITICAL=20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ERROR=30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- WARNING=40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- INFO=50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DEBUG=60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- TRACE=70</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>j</td>
<td>json</td>
<td>Parse all command flags from an input JSON file</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is a mandatory flag.</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>file</td>
<td>Full path to file to be copied/created after a successful copy</td>
<td>&quot;file&quot;: &quot;/tmp/sample.txt&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is a mandatory flag.</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>pci-addr</td>
<td>DOCA Comch device PCIe address.</td>
<td>&quot;pci-addr&quot;: &quot;b1:00.0&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This is a mandatory flag.</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>r</td>
<td>rep-pci</td>
<td>DOCA Comch device representor PCIe address.</td>
<td>&quot;rep-pci&quot;: &quot;b1:02.0&quot;</td>
</tr>
</tbody>
</table>
### Flag Type

<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note**

This is a mandatory flag only on the DPU.

---

#### 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.

---

### Application Code Flow

1. Parse application argument.

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

     ```
     doca_argp_init();
     ```

   2. Register DMA Copy application parameters.
3. Parse the arguments.

1. Initialize Comch endpoint.

   init_cc();

   1. Create Comch endpoint.

   2. Parse user PCIe address for Comch device.

   3. Open Comch DOCA device.

   4. Parse user PCIe address for Comch device representor (on DPU side).

   5. Open Comch DOCA device representor (on DPU side).

   6. Set Comch endpoint properties.

2. Open the DOCA hardware device from which the copy would be made.

   open_dma_device();

   1. Parse the PCIe address provided by the user.

   2. Create a list of all available DOCA devices.

   3. Find the appropriate DOCA device according to specific properties.

   4. Open the device.
3. Create all required DOCA core objects.

create_core_objects();

4. Initiate DOCA core objects.

init_core_objects();

5. Start host/DPU DMA Copy.

1. Host side application:

   host_start_dma_copy();

   1. Start negotiation with the DPU side application for the location and size of the file.

   2. Allocate memory for the DMA buffer.

   3. Export the memory map and send the output (export descriptor) to the DPU side application.

   4. Send the host local buffer memory address and length on the Comch to the DPU side application.

   5. Wait for the DPU to notify that DMA Copy ended.

   6. Close all memory objects.

   7. Clean resources.

2. DPU side application:

dpu_start_dma_copy();
1. Start negotiation with the host side application for file location and size.

2. Allocate memory for the DMA buffer.

3. Receive the export descriptor on the Comch.

4. Create the DOCA memory map for the remote buffer on the host.

5. Receive the host buffer information on the Comch.

6. Create two DOCA buffers, one for the remote (host) buffer and one for the local buffer.

7. Submit the DMA copy task.

8. Send a host message to notify that DMA copy ended.

9. Clean resources.

6. Destroy Comch.

```c
destroy_cc();
```

7. Destroy DOCA core objects.

```c
destroy_core_objects();
```

8. Arg parser destroy.

```c
doca_argp_destroy();
```

**References**

- `/opt/mellanox/doca/applications/dma_copy/`
• /opt/mellanox/doca/applications/dma_copy/dma_copy_params.json

© Copyright 2024, NVIDIA. PDF Generated on 08/15/2024