DOCA Tools
## Table of contents

**NVIDIA DOCA Bench**  
DOCA Bench Sample Invocations  46

**NVIDIA DOCA Capabilities Print Tool**  86

**NVIDIA DOCA Comm Channel Admin Tool**  104

**NVIDIA DPA Tools**  108

  - NVIDIA DOCA DPACC Compiler  109
  - NVIDIA DOCA DPA Execution Unit Management Tool  134
  - NVIDIA DOCA DPA GDB Server Tool  147
  - NVIDIA DOCA DPA PS Tool  161
  - NVIDIA DOCA DPA Statistics Tool  163

**NVIDIA DOCA PCC Counter Tool**  169

**NVIDIA DOCA Socket Relay**  173
This is an overview of the set of tools provided by DOCA and their purpose.

**Introduction**

DOCA tools are a set of executables/scripts that are needed to produce inputs to some of the DOCA libraries and applications.

All tools are installed with DOCA, as part of the doca-tools package, and can either be directly accessed from the terminal or can be found at /opt/mellanox/doca/tools. Refer to NVIDIA DOCA Installation Guide for Linux for more information.

![Info](image)

For questions, comments, and feedback, please contact us at DOCA-Feedback@exchange.nvidia.com.

**Tools**

**DOCA Bench**

CLI name: `doca_bench`

DOCA Bench is a tool that allows a user to evaluate the performance of DOCA applications, with reasonable accuracy for real-world applications. It provides a flexible architecture to evaluate multiple features in series with multi-core scaling to provide detailed throughput and latency analysis.

**Capabilities Print Tool**

CLI name: `doca_caps`

DOCA Capabilities Print tool is used to print the available devices and their representor devices (in the DPU), all their capabilities, and the available DOCA libraries.
DPA Tools

DOCA DPA tools are a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

PCC Counter

CLI name: pcc_counters.sh

DOCA PCC Counter tool is used to print PCC-related hardware counters. The output counters help debug the PCC user algorithm embedded in the DOCA PCC application.

Socket Relay

CLI name: doca_socket_relay

DOCA Socket Relay tool allows Unix Domain Socket (AF_UNIX family) server applications to be offloaded to Bluefield while communication between the two sides is proxied by DOCA Comm Channel.
NVIDIA DOCA Bench

Introduction

NVIDIA DOCA Bench allows users to evaluate the performance of DOCA applications, with reasonable accuracy for real-world applications. It provides a flexible architecture to evaluate multiple features in series with multi-core scaling to provide detailed throughput and latency analysis.

This tool can be used to evaluate the performance of multiple DOCA operations, gain insight into each stage in complex DOCA operations and understand how items such as buffer sizing, scaling, and GGA configuration affect throughput and latency.

Feature Overview

DOCA Bench is designed as a unified testing tool for all BlueField accelerators. It, therefore, provides these major features:

- BlueField execution, utilizing the Arm cores and GGAs "locally"
- Host (x86) execution, utilizing x86 cores and the GGAs on the BlueField over PCIe
- Support for following DOCA/DPU features:
  - DOCA AES GCM
  - DOCA Comch
  - DOCA Compress
  - DOCA DMA
  - DOCA EC
  - DOCA Eth
  - DOCA RDMA
DOCA Tools

- **DOCA SHA**

- Multi-core/multi-thread support

- Schedule executions based on time, job counts, etc.

- Ability to construct complex pipelines with multiple GGAs (where data moves serially through the pipeline)

- Various data sources (random data, file data, groups of files, etc.)

- Remote memory operations
  - Use data location on the host x86 platform as input to GGAs

- Comprehensive output to screen or CSV

- Query function to report supported software and hardware feature

- Sweeping of parameters between a start and end value, using a specific increment each time

- Specific attributes can be set per GGA instance, allowing fine control of GGA operation

**Installation**

DOCA Bench is installed and available in both DOCA-for-Host and DOCA BlueField Arm packages. It is located under the `/opt/mellanox/doca/tools` folder.

**Prerequisites**

DOCA 2.7.0 and higher.

**Granular Build Support**

DOCA Bench supports a granular build environment which allows users to determine which DOCA libraries are installed on any target system. During initialization, DOCA Bench probes all available and supported DOCA libraries, and provides the ability to test
those libraries. For example, if the DOCA SHA library is not present then DOCA Bench does not allow SHA to be tested.

DOCA Bench provides a query system where device capabilities can be queried to see if the library is indeed installed and supported (under the "installed : yes / no" section of each library). Please see section "Queries" for details.

**Operating Modes**

DOCA Bench measures performance of either throughput (bandwidth) or latency.

**Throughput Measurements**

In this mode, DOCA Bench measures the maximum performance of a given pipeline (see "Core Principles"). At the end of the execution, a short summary along with more detailed statistics is presented:

```
Aggregate stats
  Duration: 3000049 micro seconds
  Enqueued jobs: 17135128
  Dequeued jobs: 17135128
  Throughput: 5712042 Operations/s
  Ingress rate: 063.832 Gib/s
  Egress rate: 063.832 Gib/s
```

**Latency Measurements**

Latency is the measurement of time taken to perform a particular operation. In this instance, DOCA Bench measures the time taken between submitting a job and receiving a response.

DOCA Bench provides two different types of latency measurement figures:

- Bulk latency mode – attempts to submit a group of jobs in parallel to gain maximum throughput, while reporting latency as the time between the first job submitted in the group and the last job received.
• Precision latency mode – used to ensure that only one job is submitted and measured before the next job is scheduled.

**Bulk Latency**

This latency mode effectively runs the pipelines at full rate, trying to maintain the maximum throughput of any pipeline while also recording latency figures for jobs submitted.

To record latency, while operating at the pipeline's maximum throughput, users must place the latency figures inside groups or "buckets" (rather than record each individual job latency). Using this method, users can avoid the large memory and CPU overheads associated with recording millions of latency figures per second (which would otherwise significantly reduce the performance).

As each pipeline operation is different, and therefore has different latency characteristics, the user can supply the boundaries of the latency measure. DOCA Bench internally creates 100 buckets, of which the user can specify the starting value and the width or size of each bucket. The first and last bucket have significance:

• The first bucket contains all jobs that executed faster than the starting period

• The last bucket contains a count of jobs that took longer than the maximum time allowed

The command line option `--latency-bucket-range` is used to supply two values representing the starting time period of the first bucket, and the width of each sequential bucket. For example, `--latency-bucket-range 10us,100us` would start with the lowest bucket measuring <10μs response times, then 100 buckets which are 100μs wide, and a final bucket for results taking longer than 10010μs.

The report generated by bulk mode visualizes the latency data in two methods:

1. A bar graph is provided to visually show the spread of values across the range specified by the `--latency-bucket-range` option:
2. A breakdown of the number of jobs per bucket is presented. This example shortens the output to show that the majority of values lie between 27000ns and 31000ns.

<table>
<thead>
<tr>
<th>Bucket Range</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[25000ns -&gt; 25999ns]</td>
<td>0</td>
</tr>
<tr>
<td>[26000ns -&gt; 26999ns]</td>
<td>0</td>
</tr>
<tr>
<td>[27000ns -&gt; 27999ns]</td>
<td>128</td>
</tr>
<tr>
<td>[28000ns -&gt; 28999ns]</td>
<td>2176</td>
</tr>
<tr>
<td>[29000ns -&gt; 29999ns]</td>
<td>1152</td>
</tr>
<tr>
<td>[30000ns -&gt; 30999ns]</td>
<td>128</td>
</tr>
<tr>
<td>[31000ns -&gt; 31999ns]</td>
<td>0</td>
</tr>
<tr>
<td>[32000ns -&gt; 32999ns]</td>
<td>0</td>
</tr>
<tr>
<td>[33000ns -&gt; 33999ns]</td>
<td>128</td>
</tr>
<tr>
<td>[34000ns -&gt; 34999ns]</td>
<td>0</td>
</tr>
<tr>
<td>[35000ns -&gt; 35999ns]</td>
<td>0</td>
</tr>
</tbody>
</table>

**Precision Latency**

This latency mode operates on a single job at a time. At the cost of greatly reduced throughput, this allows the minimum latency to be precisely recorded. As shown below, the statistics generated are precise and include various fields such as min, max, median, and percentile values.

**Aggregate stats**

<table>
<thead>
<tr>
<th>Stat</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>1878 ns</td>
</tr>
<tr>
<td>max</td>
<td>4956 ns</td>
</tr>
<tr>
<td>median</td>
<td>2134 ns</td>
</tr>
<tr>
<td>mean</td>
<td>2145 ns</td>
</tr>
</tbody>
</table>
Core Principles

The following subsections elaborate on principles which are essential to understand how DOCA Bench operates.

Host or BlueField Arm Execution

Whether executing DOCA Bench on an x86 host or BlueField Arm, the behavior of DOCA Bench is identical. The performance measured is dependent on the environment.

Info

Only execution on x86 hosts is supported.

Pipelines

DOCA Bench is a highly flexible tool, providing the ability to configure how and what operations occur and in what order. To accomplish this, DOCA Bench uses a pipeline of operations, which are termed "steps". These steps can be a particular function (e.g., Ethernet receive, SHA hash generation, data compression). Therefore, a pipeline of steps can accomplish a number of sequential operations. DOCA Bench can measure the throughput performance or latency of these pipelines, whether running on single or multiple cores/threads.

<table>
<thead>
<tr>
<th>Percentile</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>90th</td>
<td>2243</td>
</tr>
<tr>
<td>95th</td>
<td>2285</td>
</tr>
<tr>
<td>99th</td>
<td>2465</td>
</tr>
<tr>
<td>99.9th</td>
<td>3193</td>
</tr>
<tr>
<td>99.99th</td>
<td>4487</td>
</tr>
</tbody>
</table>
Warm-up Period

To ensure correct measurement, the pipelines must be run "hot" (i.e., any initial memory, caches, and hardware subsystems must be running prior to actual performance measurements begin). This is known as the "warm-up" period and, by default, runs approximately 250 jobs through the pipeline before starting measurements.

Defaults

DOCA Bench has a large number of parameters but, to simplify execution, only a few must be supplied to commence a performance measurement. Therefore various parameters have defaults which should be sufficient for most cases. To fine tune performance, users should pay close attention to any default parameters which may affect their pipeline's operation.

Optimizing Performance
To obtain maximum performance, a certain amount of tuning is required for any given environment. While outside the scope of this documentation, it is recommended for users to:

- Avoid using CPU 0 as most OS processes and interrupt request (IRQ) handlers are scheduled to execute on this core
- Enable CPU/IRQ isolation in the kernel boot parameters to remove kernel activities from any cores they wish to execute performance tests on
- On hosts, ensure to not cross any non-uniform memory access (NUMA) regions when addressing the BlueField
- Understand the memory allocation requirements of scenarios, to avoid over-allocating or running into near out-of-memory situations

**Supported BlueField Feature Matrix**

DOCA Bench can be executed on both host and BlueField Arm environments, and can target BlueField networking platforms.

The following table shows which operations are possible using either DOCA Bench. It also provides two columns showing whether remote memory can be used as an input or output to that operation. For example, DMA operations on the BlueField Arm can access remote memory as an input to pull memory from the host into the BlueField Arm).

<table>
<thead>
<tr>
<th></th>
<th>BlueField-2 Networking Platform</th>
<th>BlueField-3 Networking Platform</th>
<th>Execute on Host Side</th>
<th>Execute on BlueField Arm</th>
<th>Remote Memory as Input Allowed?</th>
<th>Remote Memory as Output Allowed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>doca_compress::compress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_compress::decompress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>doca_dma</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>BlueField-2 Networking Platform</td>
<td>BlueField-3 Networking Platform</td>
<td>Execute on Host Side</td>
<td>Execute on BlueField Arm</td>
<td>Remote Memory as Input Allowed?</td>
<td>Remote Memory as Output Allowed?</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td>---------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>doca_ec::create</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_ec::recover</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_ec::update</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_sha</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_rdma::send</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_rdma::receive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_aes_gcm::encrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_aes_gcm::decrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_cc::client_producer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_cc::client_consumer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_eth::rx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doca_eth::tx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Input remote memory is not supported for lz4 decompression.
Remote Operations

A subset of BlueField operations have a remote element, whether this is an RDMA connection, Ethernet connectivity, or memory residing on an x86 host. All these operations require an agent to be present on the far side to facilitate the benchmarking of that particular feature.

In DOCA Bench, this agent is an additional standalone application called the "companion app". It provides the remote benchmarking facilities and is part of the standard DOCA Bench installation.

The following diagram provides an overview of the function and communications between DOCA Bench and the companion app:

![Diagram](image)

In this particular setup, the BlueField executes "DOCA Bench" while the host (x86) executes the companion App.

DOCA Bench also acts as the controller of the tests, instructing the companion app to perform the necessary operations as required. There is an out-of-band communications channel operating between the two applications that utilizes either standard TCP/IP sockets or a DOCA ComCh channel (depending on the test scenario/user preferences).

⚠️ Warning

DOCA Bench tool is not intended to be used in a production deployment. If choosing to do so, please be aware that the out-of-band communications might contain sensitive information and thus
CPU Core and Thread Selection

**Note**

Selection of the correct CPU cores and threads has a significant impact on the performance or latency obtained. Read this section carefully.

A key requirement to scaling any application is the number of CPU cores or threads allocated to any given activity. DOCA Bench provides the ability to specify the numbers of cores, and the number of threads to be created per core, to maximize the number of jobs submitted to a given pipeline.

The following care should be given when selecting the number of CPU's or threads:

- Threads that are on cores located on distant NUMA regions (i.e., not the same NUMA region the BlueField is connected to) will experience lower performance and higher latency

- Core 0 is often most used by the OS and should be avoided

- Standard Linux Kernel installations allow the OS to move processes on any CPU core resulting in unexpected drops in performance, or higher latency, due to process switching

The selection of CPU cores is provided through the \(--\)core-mask, \(--\)core-list, \(--\)core-count parameters, while thread selection is made via the \(--\)threads-per-core parameter.

**Device Selection**

When executing from a host (x86) environment DOCA Bench can target one or more BlueField devices within an installed environment. When executing from the BlueField
Arm, the target is always the local BlueField.

The default method of targeting a given BlueField from either the host or the BlueField Arm is using the --device or -A parameters, which can be provided as:

- Device PCIe address (i.e., 03:00.0);
- Device IB name (mlx5_0); or
- Device interface name (ens4f0)

From the BlueField Arm environment, DOCA Bench should be targeted at the local PCIe address (i.e., --device 03:00.0) or the IB device name (i.e., mlx5_0).

### Input Data Selection and Sizing of Jobs

DOCA Bench supports different methods of supplying data to jobs and providing information on the amount of data to process per job. These are referred to as "Data Providers".

#### Input Data Selection

The following subsections provide the modes available to provide data for input into any operation.

**File**

A single file is used as input to the operation. The contents of the file are not important for certain operations (e.g., DMA, SHA, etc.) but must be valid and specific for others (e.g., decompress, etc). The data may be used multiple times and repeated if the operations required more data than the single file contains. For more information on how file data is handled in complex operations, see section "Command-line Parameters".

**File Sets**

File sets are a group of files that are primarily used for structured data. The data in the file set is effectively a list of files, separated by a new line that is used sequentially as input data for jobs. Each file pointed to by the file set would have its entire contents read...
into a single buffer. This is useful for operations that require structured data (i.e., a complete valid block of data, such as decompression or AES).

Random Data

Random data is provided when the actual data required for the given operation is not specific (e.g., DMA).

Note

The use of random data for certain operations may reduce the maximum performance obtained. For example, compressing random data results in lower performance than compressing actual file data (due to the lack of repeating patterns in random data).

Job Sizing

Each job in DOCA Bench consists of three buffers: An original input buffer, an output, and an intermediate buffer.

The input buffer is provided by the data provider for the first step in the pipeline to use, after which the following steps use the output and intermediate buffers (can be sized by using --job-output-buffer-size) in a ping-pong fashion. This means, the pipeline can always start with the same deterministic data while allowing for each step to provide its newly generated output data to be used as input to the next step.

The input buffer is specified in one of two ways: using uniform-job-size to make every input buffer the exact same size, or using a file set to size each buffer based on the size of the selected input data file(s). Users should ensure the data generated by each step in the pipeline will fit in the provided output buffer.
Controlling Test Duration

DOCA Bench has a variety of ways to control the length of executing tests—whether based on data or time limit.

Limit to Specific Number of Seconds

Using the --run-limit-seconds or -s parameter ensures that the execution continues for a specific number of seconds.

Limited Through Total Number of Jobs

It may be desirable to measure a specific number of jobs passing through a pipeline. The --run-limit-jobs or -J parameter is used to specify the exact number of jobs submitted to the pipeline and allowed to complete before execution finishes.

GGA-specific Attributes

As DOCA Bench supports a wide range of both GGA and software based DOCA libraries, the ability to fine tune their invocation is important. Command-line parameters are generally used for configuration options that apply to all aspects of DOCA Bench, without being specific to a particular DOCA library.

Attributes are the method of providing configuration options to a particular DOCA Library, whilst some shared attributes exist the majority of libraries have specific attributes designed to control their specific behavior.

For example, the attribute doca_ec.data_block_count allows you to set the data block count for the DOCA EC library, whilst the attribute doca_sha.algorithm controls the selection of the SHA algorithm.

For a full list of support attributes, see the "Command-line Parameters" section.
DOCA Bench allows users to specify a series of operations to be performed and then scale that workload across multiple CPU cores/threads to get an estimation of how that workload performs and some insight into which stage(s), if any, cause performance problems for them. The user can then modify various configuration properties to explore how issues can be tuned to better serve their need.

When running, DOCA Bench creates a number of execution threads with affinities to the specific CPU specified by the user. Each thread creates, uniquely for themselves, a jobs pool (with job data initialized by a data provider) and a pipeline of workload steps.

**CPU Core and Thread Count Configuration**

There are many factors involved when carrying out performance tests, one of these is the CPU selection:

- The user should consider NUMA regions when selecting which cores to use, as using a CPU which is distant from the device under test can impact the performance achievable

- The user may also wish to avoid core 0 as this is typically the default core for kernel interrupt handlers.

**Note**

CPU core selection has an impact on the total memory footprint of the test. See section "Test Memory Footprint" for more details.

---

--core-mask
**Default value:** 0x02

Core mask is the simplest way to specify which cores to use but is limited in that it can only specify up to 32 CPUs (0-31). Usage example: `--core-mask 0xF001` selects CPU cores 0, 12, 13, 14, and 15.

---

**--core-list**

Core list can specify any/all CPU cores in a given system as a list, range, or combination of the two. Usage example: `--core-list 0,3,6-10` selects CPU cores 0, 3, 6, 7, 8, 9, and 10.

---

**--core-count**

The user can select the first N cores from a given core set (list or mask) if desired. Usage example: `--core-count N`.

---

**Info**

Sweep testing is supported. See section "Sweep Tests" for more details.

---

**--threads-per-core -t**

To test the impacts of contention within a single CPU core, the user can specify this value so that instead of only one thread being created per core, N threads are created with their affinity mask set to the given core for each core selected. For example, 3 cores and 2 threads per core create 6 threads total.

---

**Info**
Sweep testing is supported. See section "Sweep Tests" for more details.

Device Configuration

The test requires the use of at least one BlueField to execute. With remote system testing, a second device may be required.

--device -A

Specify the device to use from the perspective of the system under test. The value can be for any one of either the device PCIe address (e.g., 03:00.0), the device IB device name (e.g., mlx5_0), or the device interface name (e.g., ens4f0).

--representor -R

This option is used only when performing remote memory operations between a BlueField device and its host using DOCA Comch. This is typically automated by the companion connection string but exists for some developer debug use-cases.

Info

This option used to be important before the companion connection string property was introduced but now is rarely used.

Input Data and Buffer Size Configuration
DOCA Bench supports multiple methods of acquiring data to use to initialize job buffers. The user can also configure the output/intermediate buffers associated with each job.

---

**Info**

Input data and buffer size configuration has an impact on the total memory footprint of the test. See section "Test Memory Footprint" for more details.

---

--data-provider -I

DOCA Bench supports several different input data sources:

- file
- file-set
- random-data

**File Data Provider**

The file data provider produces uniform/non-structured data buffers by using a single input file. The input data is stripped and or repeated to fill each data buffer as required, returning back to the start of the file each time it is exhausted to collect more data. This is desirable when the performance of the component(s) under test is meant to show different performance characteristics depending on the input data supplied.

For example, `doca_dma` and `doca_sha` would execute in constant time regardless of the input data. Whereas `doca_compress` would be faster with data with more duplication and slower for truly random data and would produce different output depending on the input data.

**Example 1 - Small Input File with Large Buffers**

Given a small input data (i.e., smaller than the data buffer size), the file contents are repeated until the buffer is filled and then continue onto the next buffer(s). So, if the input file contained the data `012345` and the user requested two 20-byte buffers, the buffers would appear as follows:
Example 2 – Large Input File with Smaller Buffers

Given a large input data (i.e., greater than the data buffer size), the file contents are distributed across the data buffers. If the input file contained the data 0123456789abcdef and the user requested three 12-byte buffers, the buffers would appear as follows:

- 0123456789ab
- cdef01234567
- 89abcdef0123

File Set Data Provider

The file set data provider produces structured data. The file set input file itself is a file containing one or more filenames (relative to the input "command working directory (cwd)" not relative to the file set file). Each file listed inside the file set would have its entire contents used as a job buffer. This is useful for operations where the data must be a complete valid data block for the operation to succeed like decompression with doca_compress or decryption with doca_aes.

Example – File Set and Its Contents

Given a file set in the "command working directory (cwd)" referring to data_1.bin and data_2.bin (one file name per line), and data_1.bin contains 33 bytes and data_2.bin contains 69 bytes, then the data required by the buffers would be filled with these two files in a round-robin manner until the buffers are full. Unlike uniform (non-structured) data each task can have different lengths.
Random-data Data Provider

The random data data provider provides uniform (non-structured) data from a random data source. Each buffer will have unique (pseudo) random bytes of content.

--data-provider-job-count

Default value: 128

Each thread in DOCA Bench has its own allocation of job data buffers to avoid memory contention issues. Users may select how many jobs should be created per thread using this parameter.

Info

Sweep testing is supported. See section "Sweep Tests" for more details.

--data-provider-input-file

For data providers which use an input file, the filename can be specified here. The filename is relative to the input_cwd.

Info

Sweep testing is supported. See section "Sweep Tests" for more details.
--uniform-job-size

Specify the size of uniform input buffers (in bytes) that should be created.

⚠️ **Note**

Does not apply and should not be specified when using structured data input sources.

⚠️ **Info**

Sweep testing is supported. See section "Sweep Tests" for more details.

--job-output-buffer-size

**Default value:** 16384

Specify the size of output/intermediate buffers (in bytes). Each job has 3 buffers: immutable input buffer and two output/intermediate buffers. This allows for a pipeline to mutate the data an infinite number of times throughout the pipeline, while allowing for it to be reset and re-used at the end and allowing any step to use the new mutated data created by the previous step.

--input-cwd -i

To ease configuration management, the user may opt to use a separate folder for the input data for a given scenario outside of the DOCA build/install directory.
**Tip**

It is recommended to use relative file paths for the input files.

**Example 1 – Running DOCA Bench from Current Working Directory**

Considering a user executing DOCA Bench from `/home/bob/doca/build`, values specified in `--data-provider-input-file` and filenames within a file set would search relative to the shell's "command working directory (cwd)". `/home/bob/doca/build`. Their command might look something like:

```bash
doca_bench --data-provider file-set --data-provider-input-file my_file_set.txt
```

And assuming `my_file_set.txt` contains `data_1.bin`, the files that would be loaded by DOCA Bench after path resolution would be:

- `/home/bob/doca/build/my_file_set.txt`
- `/home/bob/doca/build/data_1.bin`

**Example 2 – Running DOCA Bench from Another Directory**

Considering the user executed that same test from one level up. Something like:

```bash
build/doca_bench --data-provider file-set --data-provider-input-file build/my_file_set.txt
```

The files to be loaded would be:

- `/home/bob/doca/build/my_file_set.txt`
- `/home/bob/doca/data_1.bin`
Notice how both files were loaded relative to the "command working directory (cwd)" and the data file was not loaded relative to the file set.

**Example 3 – Example 2 Revisited Using input-cwd**

The user can solve this easily by keeping all input files in a single directory and then referring to that directory using the parameter input-cwd. In this case, the command like may look something like:

```
build/doca_bench --data-provider file-set --data-provider-input-file my_file_set.txt --input-cwd build
```

Note that the value for --data-provider-input-file also changed to be relative to the new "command working directory (cwd)".

The files loaded this time are back to being what is expected:

- `/home/bob/doca/build/my_file_set.txt`
- `/home/bob/doca/build/data_1.bin`

**Test Execution Control**

DOCA Bench supports multiple test modes and run execution limits to allow the user to configure the test type and duration.

**--mode**

**Default value:** throughput

Select which type of test is to be performed.

**Throughput Mode**
Throughput mode is optimized to increase the volume of data processed in a given period with little or no regard for latency impact. Throughput mode tries to keep each component under test as busy as possible. A summary of the bandwidth and job execution rate are provided as output.

**Bulk-latency Mode**

Bulk latency mode strikes a balance between throughout and latency, submitting a batch of jobs and waiting for them all to complete to measure the latency of each job. This mode uses a bucketing mechanism to allow DOCA Bench to handle many millions of jobs worth of results. DOCA Bench keeps a count of the number of jobs that complete within each bucket to allow it to run for long periods of time. A summary of the distribution of results with an ASCII histogram of the results are provided as output. The latency reported is the time taken between the first job submission (for a batch of jobs) until the final job response is received (for that same batch of jobs).

**Precision-latency Mode**

Precision latency mode executes one job at a time to allow DOCA Bench to calculate the minimum possible latency of the jobs. This causes the components which can process many jobs in parallel to be vastly underutilized and so greatly reduces bandwidth. As this mode records every result individually, it should not be used to execute more than several thousand jobs. Precision latency mode requires 8 bytes of storage for each result, so be mindful of the memory overhead of the number of jobs to be executed.

A statistical analysis including minimum, maximum, mean, median and some percentiles of the latency value are provided as output.

**--latency-bucket-range**

**Default value:** 100ms, 10ms

Only applicable to bulk-latency mode. Allows the user to specify the starting value of the buckets, and the width of each bucket. There are 100 buckets of the given size and an
under flow and overflow bucket for results that fall outside of the central range.

For example:

```
--latency-bucket-range 10us,100us
```

This would start with the lowest bucket measuring <10μs response times, then 100 buckets which are 100μs wide, and a final bucket for results taking longer than >10010μs.

## Blocking Mode

DOCA supports two methods of waiting on completion of tasks:

- Busy-wait (or polling) mode
- Notification-driven mode

---

### Info

Refer to "DOCA SDK Architecture" documentation for more information.

---

By default, DOCA Bench uses the busy-wait to ensure maximum bandwidth (and low latency) for any given pipeline and its tasks with high utilization of any allocated CPU resources.

As with all high-performance software, utilizing GGAs or hardware accelerators, performance is usually CPU-bound at smaller packet sizes (i.e., at smaller payload sizes, the CPU spends a long time generating tasks and dealing with completions). For larger packet sizes, the CPU submits less tasks, as each task contains more data, therefore it may easily submit more data than the GGA or hardware accelerator can accept, resulting
in periods where the CPU is busy-waiting on completions before being able to submit further tasks.

**Info**

To execute any tests using an "notification-driven mode", use the options detailed in the following subsections.

**--use-blocking-mode**

This option causes DOCA Bench to use the "notification-drive mode" method of waiting on task completion.

**Note**

At smaller packet sizes, the benchmark may still be CPU bound.

**--record-cpu-usage**

If specified, this option reports CPU statistics for any CPU cores DOCA Bench is executing on. This provides guidance on how much CPU time is returned, and thus available to other processes or threads, should the "notification-driven" mode be active.

**Note**

Short duration tests may not result in sufficient produced data to generate CPU usage statistics.
The statistics provided include min, max, median, and mean values for the CPU usage. Also included are a number of percentile results, showing 90th, 95th, and a number of 99th percentile values. Example output:

<table>
<thead>
<tr>
<th>CPU Usage stats</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>min: 25%</td>
<td></td>
</tr>
<tr>
<td>max: 50%</td>
<td></td>
</tr>
<tr>
<td>median: 50%</td>
<td></td>
</tr>
<tr>
<td>mean: 45.8333%</td>
<td></td>
</tr>
<tr>
<td>90th %ile: 50%</td>
<td></td>
</tr>
<tr>
<td>95th %ile: 50%</td>
<td></td>
</tr>
<tr>
<td>99th %ile: 50%</td>
<td></td>
</tr>
<tr>
<td>99.9th %ile: 50%</td>
<td></td>
</tr>
<tr>
<td>99.99th %ile: 50%</td>
<td></td>
</tr>
</tbody>
</table>

**Execution Limits**

By default, a test runs forever. This is typically undesirable so the user can specify a limit to the test.

**Note**

Precision-latency mode only supports job limited execution.

```
--run-limit-seconds -s
```

Runs the test for N seconds as specified by the user.

```
--run-limit-jobs -J
```
Runs the test until at least N jobs have been submitted, then allowing in-flight jobs to complete before exiting. More jobs than N may be executed based on batch size.

```bash
--run-limit-bytes -b
```

Runs the test until at least N bytes of data have been submitted, then allowing in-flight jobs to complete before exiting. More data may be processed than desired if the limit is not a multiple of the job input buffer size.

**Gather/Scatter Support**

Gather support involved breaking incoming input data from a single buffer into multiple buffers, which are "gathered" into a single gather list. Currently only gather is supported.

```bash
--gather-value
```

**Default value: 1**

Specifies the partitioning of input data from a single buffer into a gather list. The value can be specified in two flavors:

- `--gather-value 4` – splits input buffers into 4 parts as evenly as possible with odd bytes in the last segment

- `--gather-value 4KiB` – splits buffers after each 4KB of data. See `doca_bench/utility/byte_unit.hpp` for the list of possible units.

**Stats Output**

```bash
--rt-stats-interval
```

By default, DOCA Bench emits the results of an iteration once it completes. The user can ask for transient snapshots of the stats as the test progresses by providing the `--rt-stats-`
interval argument with a value representing the number of milliseconds between stat
prints. The end-result of the run is still displayed as normal.

Note

This may produce a large amount of console output.

--csv-output-file

DOCA Bench can produce an output file as part of its execution which can contain stats
and the configuration values used to produce that stat. This is enabled by specifying the --
csv-output-file argument with a file path as the value. Providing a value for this argument
enables CSV stats output (in addition to the normal console output). When performing a
sweep test, one line per iteration of the sweep test is populated.

By default, the CSV output contains every possible value. The user can tune this by
applying a filter.

--csv-stats

Provide one or more filters (positive or negative) to tune which stats are displayed. The
value for this argument is a comma-separated list of filter strings. Negative filters start
with a minus sign ('-').

Example 1 – Emit Only Statistical Values (No Configuration Values)

```
--csv-stats "stats.*"
```

Note
Example 2 – Emit Statistical Values and Some Configuration Values (Remove Attribute Values)

```
--csv-stats "stats.*,~attribute*"
```

**--csv-append-mode**

**Default:** false

When enabled, DOCA Bench appends to a CSV file if it exists or creates a new one. It is assumed that all invocation uses the exact same set of output values. This is not verified by DOCA Bench. The user must ensure that all tests that append to the CSV use the same set of output values.

**--csv-separate-dynamic-values**

A special case which creates a non-standard CSV file. All values that are not supported by sweep tests are reported only once first, then a new line of headers for values emitted during the test, then a row for each test result. This is reserved for an internal use case and should not be relied upon by anyone else.

**--enable-environment-information**

Instructs DOCA Bench to collect some detailed system information as part of the test startup procedure which are then made available for output in the CSV. These also gather the same details from the companion side if the companion is in use.

The quotes around the * prevent the shell from interpreting it as a wild card for filenames in the command.
Remote Memory Testing

Some libraries (e.g., `doca_dma`) support the use of remote memory. To enable this, the user can specify one or both of the remote memory flags `--use-remote-input-buffers` and `--use-remote-output-buffers`. This tells DOCA Bench to use the companion to create a remote mmap. This remote mmap is then used to create buffers that are submitted to the component under test.

---

**Warning**

This collection can take a long time (up to a few minutes in some circumstances) to complete, so it is not recommended unless you know you need it.

---

**Note**

These flags should be used with caution and an understanding that if the underlying components under test can support this scenario, there is no automated checking. It is user responsibility to ensure these are used appropriately.

---

**--use-remote-input-buffers**

Specifies that the memory used for the initial immutable job input buffers into a pipeline should be backed by an mmap on the remote side.

---

**Note**
--use-remote-output-buffers

Specifies that all output and translation buffers in use are backed by an mmap on the remote side.

Note

Requires the companion app to be configured.

Network Options

--mtu-size

For use with doca_rdma. Value is an enum: 256B 512B 1KB 2KB 4KB or raw_eth.

--receive-queue-size

For use with doca_rdma. Configure the RDMA RQ size independently of the SQ size.

--send-queue-size

For use with doca_rdma. Configure the RDMA SQ size independently of the RQ size.
DOCA Lib Configuration Options

--task-pool-size

Default value: 1024

Configure the maximum task pool size used when libraries initialize task pools.

Pipeline Configuration

DOCA Bench is based on a pipeline of operations, This allow for complex test scenarios where multiple components are tested in parallel. Currently only a single chain of operations in a pipeline is supported (but scaled across multiple cores or threads), future versions will allow for varied pipeline’s per CPU core.

A pipeline is described as a series of steps. All steps have a few general characteristics:

- Step type: doca_dma, doca_sha, doca_compress, etc.
- An operation category – transformative or non-transformative
- An input data category – structured or non structured

Individual step types may also have some additional metadata information or configuration as defined on a per step basis.

Metadata examples:

- doca_compress requires an operation type: compress or decompress
- doca_aes requires an operation type: encrypt or decrypt
- doca_ec requires an operation type: create, recover or update
- doca_rdma requires a direction: send, receive or bidir
Configuration examples:

- --pipeline-steps doca_dma
- --pipeline-steps doca_compress::compress,doca_compress::decompress

**--pipeline-steps**

Define the step(s) (comma-separated list) to be executed by each thread of the test.

The following is the list of supported steps:

- doca_compress::compress
- doca_compress::decompress
- doca_dma
- doca_ec::create
- doca_ec::recover
- doca_ec::update
- doca_sha
- doca_rdma::send
- doca_rdma::receive
- doca_rdma::bidir
- doca_aes_gcm::encrypt
- doca_aes_gcm::decrypt
- doca_cc::client_producer
- doca_cc::client_consumer
- doca_eth::rx
- doca_eth::tx
Some modules may be unavailable if they were not compiled as part of DOCA when DOCA Bench was compiled.

--attribute

Some of the options are very niche or specific to a single step/mmo type, so they are defined simply as attributes instead of a unique command-line argument.

The following is the list of supported options:

- doption.mmp.log_qp_depth
- doption.mmo.log.num_qps
- doption.companion_app.path
- doca_compress.algorithm
- doca_ec.matrix_count
- doca_ec.data_block_count
- doca_ec.redundancy_block_count
- doca_sha.algorithm
- doca_rdma.gid-index
- doca_eth.max_burst_size
- doca_eth.l3_chksum_offload
- doca_eth.l4_chksum_offload
--warm-up-jobs

**Default value:** 100

Warm-up serves two purposes:

- Firstly, it runs N tasks in a round robin fashion to get the data path code, tasks memory, and tasks data buffers memory into the CPU caches before the measurement of the test begins.

- Secondly, it uses `doca_task_try_submit` instead of `doca_task_submit` to validate the jobs. This validation is not desirable during the proper hot path as it costs time revalidating the task each execution.

The user should ensure their warmup count equals or exceeds the number of tasks being used per thread (see `--data-provider-job-count`).

### Companion Configuration

Some tests require a remote system to function. For this purpose, DOCA Bench comes bundled with a companion application (this application is installed as part of the DOCA-for-Host or BlueField packages). The companion is responsible for providing services to DOCA Bench such as creating a `doca_mmap` on the remote side and exporting it for use with remote operations like `doca_dma/doca_sha`, or other `doca_libs` that support remote memory input buffers. DOCA Bench can also provide remote worker processes for libraries that require them such as `doca_rdma` and `doca_cc`. The companion is enabled by providing the `--companion-connection-string` argument. Companion remote workers are enabled by providing either of the arguments `--companion-core-list` or `--companion-core-mask`.

![Info]

DOCA Bench requires that an SSH key is configured to allow the user specified to SSH without a password to the remote system using the
The companion connection may also specify the no-launch option.

⚠️ **Warning**

This is reserved for expert developer use.

The user may also specify a path to a specific companion binary to allow them to test companion binaries not in the default install path using the following command:

```
--attribute doption.companion_app.path=/tmp/my_doca_build/tools/bench/doca_bench_companion
```

⚠️ **Warning**

This is reserved for expert developer use.

`--companion-connection-string`

Specifies the details required to establish a connection to and execute the companion process.

- Example of running DOCA Bench from the host side using the BlueField for the remote side using `doca_comch` as the communications method:

```
--companion-connection-string
"proto=dcc,mode=DPU,user=bob,addr=172.17.0.1,dev=03:00.0,rep=d8:00.0"
```
- Example of running DOCA Bench from the BlueField side using the host for the remote side using `doca_comch` as the communications method:

```
--companion-connection-string "proto=dcc,mode=host,user=bob.addr=172.17.0.1.dev=d8:00.0"
```

- Example of running DOCA Bench on one host with the companion on another host using TCP as the communications method:

```
--companion-connection-string "proto=tcp,user=bob.addr=172.17.0.1.port=12345.dev=d8:00.0"
```

⚠️ **Note**

For `doca_rdma` only.

---

**--companion-core-list**

Works the same way as `--core-list` but defines the cores to be used on the companion side.

⚠️ **Note**

Must be at least as large as the `--core-list`.

---

**--companion-core-mask**

Works the same way as `--core-mask` but defines the cores to be used on the companion side.
Sweep Tests

--sweep

DOCA Bench supports executing a set of tests based on a number of value ranges. For example, to understand the performance of multi-threading, the user may wish to run the same test for various CPU core counts. They may also wish to vary more than one aspect of the test. Providing one or more --sweep parameters activates sweep test mode where every combination of values is tested with a single invocation of DOCA Bench.

The following is a list of the supported sweep test options:

- core-count
- data-provider-input-file
- data-provider-job-count
- gather-value
- mtu-size
- receive-queue-size
- send-queue-size
- threads-per-core
- task-pool-size
- uniform-job-size

Note

Must be at least as large as the --core-mask.
Sweep test argument values take one of three forms:

- `--sweep param,start_value,end_value,+N`
- `--sweep param,start_value,end_value,*N`
- `--sweep param,value1,...,valueN`

Sweep core count and input file example:

```bash
--sweep core-count,1,8,*2
--sweep data-provider-input-file,file1.bin,file2.bin
```

This would sweep cores 1-8, inclusive, multiplying the value each time as 1,2,4,8 and two different input files resulting in a cumulative 8 test cases:

<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Core Count</th>
<th>Input File</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>file1.bin</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>file1.bin</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>file1.bin</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>file1.bin</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>file2.bin</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>file2.bin</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>file2.bin</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>file2.bin</td>
</tr>
</tbody>
</table>
Queries

Device Capabilities

DOCA Bench allows the querying of a device to report which step types are available as well as information of valid configuration options for each step. A device must be specified:

```
tools/bench/doca_bench --device 03:00.0 --query device-capabilities
```

For each supported library, this would report:

- **Capable** – if that library is enabled in DOCA Bench at compile time (if not capable, installing the library would not make it become available to bench)

- **Installed** – if the library is installed on the machine executing the query (if not installed, installing it would make it available to bench)

- **Library wide attributes**

- **A list of supported task types (~= step name)**
  - If the task type is supported
  - Task specific attributes/capabilities

```
doca_compress:
   Capable: yes
   Installed: yes
   Tasks:
      compress::deflate:
         Supported: no
      compress::lz4:
         Supported: no
      compress::lz4_stream:
         Supported: no
      decompress::deflate:
         Supported: yes
         Max buffer length: 134217728
```
decompress::lz4:
  Supported: yes
  Max buffer length: 134217728

decompress::lz4_stream:
  Supported: yes
  Max buffer length: 134217728

**Supported Sweep Attributes**

Shows the possible parameters that can be used with the sweep test parameter

```
tools/bench/doca_bench --query sweep-properties
```

Example output:

```
Supported query properties: [
  core-count
  threads-per-core
  uniform-job-size
  task-pool-size
  data-provider-job-count
  gather-value
  mtu-size
  send-queue-size
  receive-queue-size
  doption.mmo.log_qp_depth
  doption.mmo.log_num_qps
  doca_rdma.transport-type
  doca_rdma.gid-index
]
```

**Test Memory Footprint**
DOCA Bench allocates memory for all the tasks required by the test based on the input buffer size, output/intermediate buffer size, number of cores, number of threads, and number of jobs in use. All jobs contain an input buffer, an output buffer, and an intermediate buffer. The input buffer is immutable and sized based on the data provider in use. The output and intermediate buffers are sized based on the users specification or automatically calculated at the users request. For a library which produces the same amount of output as it consumes (e.g., `doca_dma`), typically the user should set the buffers all to the same size to make things as efficient as possible.

The memory footprint for job buffers can be calculated as: 
\[(\text{number-of-tasks}) \times (\text{number-of-cores}) \times (\text{number-of-threads-per-core}) \times (\text{input-buffer-size} + (\text{output/intermediate-buffer-size} \times 2))\]. For a 1KB job with the default of 32 jobs, 1 core, and 1 core per thread, the memory footprint would be 96KB.

For sweep testing and structured data input, it can be difficult to pick a suitable output buffer size so the user may choose to specify 0 and have DOCA Bench try all the tasks once to calculate the required output buffer sizes. This only has a cost in terms of time taken to perform the calculation. After this, there is no difference between auto-sizing and manually sizing the jobs output buffers.

**Note**

When running DOCA Bench on the BlueField and on some host OSs, it may be necessary to increase the limit of how much memory the process can acquire. Consult your OS's documentation for details of how to do this.

**DOCA Bench Sample Invocations**

**Overview**

This guide provides examples of various invocations of the tool to help provide guidance and insight into it and the feature under test.
DOCA Eth Receive Sample

- This test invokes DOCA Bench to run in Ethernet receive mode, configured to receive Ethernet frames of size 1500 bytes.

- The test runs for 3 seconds using a single core and use a maximum burst size of 512 frames.

- The test runs in the default throughput mode, with throughput figures displayed at the end of the test run.

- The companion application uses 6 cores to continuously transmit Ethernet frames of size 1500 bytes until it is stopped by DOCA Bench.

Command Line

doca_bench --core-mask 0x02 \
    --pipeline-steps doca_eth::rx \
    --device b1:00.1 \

--data-provider random-data \ 
--uniform-job-size 1500 \ 
--run-limit-seconds 3 \ 
--attribute doca_eth.max-burst-size=512 \ 
--companion-connection-string \ 
proto=tcp,addr=10.10.10.10,port=12345,user=bob,dev=ens4f1np1 \ 
--attribute \ 
dooption.companion_app.path=/opt/mellanox/doca/tools/doca_bench_companion \ 
--companion-core-list 6 \ 
--job-output-buffer-size 1500 \ 
--mtu-size raw_eth

Results Output

[main] doca_bench : 2.7.0084
[main] release build
+++ + + + + + + + + + + + + + + + + + + + + + + + + +
DOCA bench supported modules: [doca_comm_channel, doca_compress, doca_dma, doca_ec, doca_eth, doca_sha, doca_comch, doca_rdma, doca_aes_gcm]
+++ + + + + + + + + + + + + + + + + + + + + + + + + +
DOCA bench configuration
Static configuration: [
  Attributes: [doca_eth.l4-chksum-offload:false, doca_eth.max-burst-size:512,
  dooption.companion_app.path:/opt/mellanox/doca/tools/doca_bench_companion, doca_eth.l3-chksum-offload:false]
  Companion configuration: [
    Device: ens4f1np1
    Remote IP address: "bob@10.10.10.10"
    Core set: [6]
  ]
] Pipelines: [
  Steps: [
    name: "doca_eth::rx"
    attributes: []
  ]
  Use remote input buffers: no
  Use remote output buffers: no
  Latency bucket_range: 10000ns-110000ns
]
Run limits: [ 
    Max execution time: 3seconds
    Max jobs executed: -- not configured --
    Max bytes processed: -- not configured --
]
Data provider: [ 
    Name: "random-data"
    Job output buffer size: 1500
]
Device: "b1:00.1"
Device representor: "-- not configured --"
Warm up job count: 100
Input files dir: "-- not configured --"
Output files dir: "-- not configured --"
Core set: [1]
Benchmark mode: throughput
Warnings as errors: no
CSV output: [ 
    File name: -- not configured --
    Selected stats: []
    Deselected stats: []
    Separate dynamic values: no
    Collect environment information: no
    Append to stats file: no
]
Test permutations: [ 
    Attributes: []
    Uniform job size: 1500
    Core count: 1
    Per core thread count: 1
    Task pool size: 1024
    Data provider job count: 128
    MTU size: ETH_FRAME
    SQ depth: -- not configured --
    RQ depth: -- not configured --
    Input data file: -- not configured --
]
[main] Initialize framework...
[main] Start execution...
Preparing...
EAL: Detected CPU lcores: 36
EAL: Detected NUMA nodes: 4
Results Overview

As a single core is specified, there is a single section of statistics output displayed.

DOCA Eth Send Sample

- This test invokes DOCA Bench to run in Ethernet send mode, configured to transmit Ethernet frames of size 1500 bytes
- Random data is used to populate the Ethernet frames
- The test runs for 3 seconds using a single core and uses a maximum burst size of 512 frames
- L3 and L4 checksum offloading is not enabled
• The test runs in the default throughput mode, with throughput figures displayed at the end of the test run

• The companion application uses 6 cores to continuously receive Ethernet frames of size 1500 bytes until it is stopped by DOCA Bench

**Command Line**

doca_bench --core-mask 0x02
  --pipeline-steps doca_eth::tx
  --device b1:00.1
  --data-provider random-data
  --uniform-job-size 1500
  --run-limit-seconds 3
  --attribute doca_eth.max-burst-size=512
  --attribute doca_eth.l4-chksum-offload=false
  --attribute doca_eth.l3-chksum-offload=false
  --companion-connection-string
  proto=tcp,addr=10.10.10.10,port=12345,user=bob,dev=ens4f1np1
  --attribute
doption.companion_app.path=/opt/mellanox/doca/tools/doca_bench_companion
     --companion-core-list 6
  --job-output-buffer-size 1500

**Results Output**

```bash
[main] doca_bench : 2.7.0084
[main] release build
+++++++++++++++
DOCA bench supported modules: [doca_comm_channel, doca_compress, doca_dma, doca_ec,
doca_eth, doca_sha, doca_comch, doca_rdma, doca_aes_gcm]
+++++++++++++++

DOCA bench configuration
Static configuration: [
   Attributes: [doca_eth.l4-chksum-offload=false, doca_eth.max-burst-size=512,
dooption.companion_app.path=/opt/mellanox/doca/tools/doca_bench_companion, doca_eth.l3-chksum-offload=false]
```
Companion configuration: [
  Device: ens4f1np1
  Remote IP address: "bob@10.10.10.10"
  Core set: [6]
]

Pipelines: [
  Steps: [
    name: "doca_eth::tx"
    attributes: []
  ]
  Use remote input buffers: no
  Use remote output buffers: no
  Latency bucket_range: 10000ns-110000ns
]

Run limits: [
  Max execution time: 3seconds
  Max jobs executed: -- not configured --
  Max bytes processed: -- not configured --
]

Data provider: [
  Name: "random-data"
  Job output buffer size: 1500
]

Device: "b1:00.1"
Device representor: "-- not configured --"
Warm up job count: 100
Input files dir: "-- not configured --"
Output files dir: "-- not configured --"
Core set: [1]
Benchmark mode: throughput
Warnings as errors: no

CSV output: [
  File name: -- not configured --
  Selected stats: []
  Deselected stats: []
  Separate dynamic values: no
  Collect environment information: no
  Append to stats file: no
]

Test permutations: [
  Attributes: []
  Uniform job size: 1500
  Core count: 1
Results Overview

As a single core is specified, there is a single section of statistics output displayed.

Host-side AES-GCM Decrypt Sample

- This test invokes DOCA Bench on the x86 host side to run the AES-GM Decryption step

- A file-set file is used to indicate which file is to be decrypted. The content of the file-set file lists the filename to be decrypted.
• The key to be used for the encryption and decryption is specified using the `doca_aes_gcm.key-file` attribute. This contains the key to be used.

• It will run until 5000 jobs have been processed

• It runs in the precision-latency mode, with latency and throughput figures displayed at the end of the test run

• A core mask is specified to indicate that cores 12, 13, 14, and 15 are to be used for this test

**Command Line**

doca_bench --mode precision-latency \
  --core-mask 0xf000 \
  --warm-up-jobs 32 \
  --device 17:00.0 \
  --data-provider file-set \
  --data-provider-input-file aes_64_128.fileset \
  --run-limit-jobs 5000 \
  --pipeline-steps doca_aes_gcm::decrypt \
  --attribute doca_aes_gcm.key-file='aes128.key' \
  --job-output-buffer-size 80

**Results Output**

```
[main] Completed! tearing down...
Worker thread[0](core: 12) stats:
  Duration: 10697 micro seconds
  Enqueued jobs: 5000
  Dequeued jobs: 5000
  Throughput: 000.467 MOperations/s
  Ingress rate: 000.265 Gib/s
  Egress rate: 000.223 Gib/s
Worker thread[1](core: 13) stats:
  Duration: 10700 micro seconds
  Enqueued jobs: 5000
  Dequeued jobs: 5000
```
Results Overview

Since a core mask is specified but no core count, then all cores in the mask are used.

There is a section of statistics displayed for each core used as well as the aggregate statistics.
BlueField-side AES-GCM Encrypt Sample

- This test invokes DOCA Bench on the BlueField side to run the AES-GM encryption step

- A text file of size 2KB is the input for the encryption stage

- The key to be used for the encryption and decryption is specified using the `doca_aes_gcm.key` attribute

- It runs until 2000 jobs have been processed

- It runs in the bulk-latency mode, with latency and throughput figures displayed at the end of the test run

- A single core is specified with 2 threads

Command Line

doca_bench --mode bulk-latency \
    --core-list 3 \
    --threads-per-core 2 \
    --warm-up-jobs 32 \
    --device 03:00.0 \
    --data-provider file \
    --data-provider-input-file plaintext_2k.txt \
    --run-limit-jobs 2000 \
    --pipeline-steps doca_aes_gcm::encrypt \
    --attribute doca_aes_gcm.key="0123456789abcdef0123456789abcdef" \
    --uniform-job-size 2048 \
    --job-output-buffer-size 4096

Results Output

[main] Completed! tearing down...
Worker thread[0](core: 3) stats:
  Duration: 501 micro seconds
  Enqueued jobs: 2048
  Dequeued jobs: 2048
  Throughput: 004.082 MOperations/s
  Ingress rate: 062.279 Gib/s
  Egress rate: 062.644 Gib/s

Worker thread[1](core: 3) stats:
  Duration: 466 micro seconds
  Enqueued jobs: 2048
  Dequeued jobs: 2048
  Throughput: 004.386 MOperations/s
  Ingress rate: 066.922 Gib/s
  Egress rate: 067.314 Gib/s

Aggregate stats
  Duration: 501 micro seconds
  Enqueued jobs: 4096
  Dequeued jobs: 4096
  Throughput: 008.163 MOperations/s
  Ingress rate: 124.558 Gib/s
  Egress rate: 125.287 Gib/s

Latency report:
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...
  ...

---------------------------------------------------------------------
[<10000ns]: 0
.. OUTPUT RETRACTED (SHORTENED) ..
[26000ns -> 26999ns]: 0
[27000ns -> 27999ns]: 128
[28000ns -> 28999ns]: 2176
[29000ns -> 29999ns]: 1152
[30000ns -> 30999ns]: 128
[31000ns -> 31999ns]: 0
[32000ns -> 32999ns]: 0
[33000ns -> 33999ns]: 128
[34000ns -> 34999ns]: 0
Results Overview

Since a single core is specified, there is a single section of statistics output displayed.

Host-side AES-GCM Encrypt and Decrypt Sample

- This test invokes DOCA Bench on the host side to run 2 AES-GM steps in the pipeline, first to encrypt a text file and then to decrypt the associated output from the encrypt step.

- A text file of size 2KB is the input for the encryption stage.

- The `input-cwd` option instructs DOCA Bench to look in a different location for the input file, in the parent directory in this case.

- The key to be used for the encryption and decryption is specified using the `doca_aes_gcm.key-file` attribute, indicating that the key can be found in the specified file.

- It runs until 204800 bytes have been processed.

- It runs in the default throughput mode, with throughput figures displayed at the end of the test run.

.. OUTPUT RETRACTED (SHORTENED) ..

[>110000ns]: 0
Command Line

doca_bench --core-mask 0xf00 \ 
  --core-count 1 \ 
  --warm-up-jobs 32 \ 
  --device 17:00.0 \ 
  --data-provider file \ 
  --input-cwd ../. \ 
  --data-provider-input-file plaintext_2k.txt \ 
  --run-limit-bytes 204800 \ 
  --pipeline-steps doca_aes_gcm::encrypt,doca_aes_gcm::decrypt \ 
  --attribute doca_aes_gcm.key-file='aes128.key' \ 
  --uniform-job-size 2048 \ 
  --job-output-buffer-size 4096

Results Output

Executing...
Worker thread[0](core: 8) [doca_aes_gcm::encrypt>>doca_aes_gcm::decrypt] started...
Worker thread[0] Executing 32 warm-up tasks using 32 unique tasks
Cleanup...
[main] Completed! tearing down...
Aggregate stats
  Duration:  79 micro seconds
  Enqueued jobs:  214
  Dequeued jobs:  214
  Throughput:  002.701 MOperations/s
  Ingress rate:  041.214 Gib/s
  Egress rate:  041.214 Gib/s

Results Overview

Since a single core is specified, there is a single section of statistics output displayed.
Host-side SHA with CSV Output File Sample

- This test invokes DOCA Bench on the host side to execute the SHA operation using the SHA256 algorithm and to create a CSV file containing the test configuration and statistics.

- A list of 1 core is provided with a count of 2 threads per core.

Command Line

doca_bench --core-mask 2 \ 
   --threads-per-core 2 \ 
   --pipeline-steps doca_sha \ 
   --device d8:00.0 \ 
   --data-provider random-data \ 
   --uniform-job-size 2048 \ 
   --job-output-buffer-size 2048 \ 
   --run-limit-seconds 3 \ 
   --attribute doca_sha.algorithm=sha256 \ 
   --warm-up-jobs 100 \ 
   --csv-output-file /tmp/sha_256_test.csv

Results Output

Executing...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [1] started...
WT[1] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...
Stats for thread[0](core: 1)
   Duration: 3000064 micro seconds
   Enqueued jobs: 3713935
   Dequeued jobs: 3713935
   Throughput: 001.238 MOperations/s
   Ingress rate: 018.890 Gib/s
Results Overview

As a single core has been specified with a thread count of 2, there are statistics displayed for each thread as well as the aggregate statistics.

It can also be observed that 2 threads are started on core 1 with each thread executing the warm-up jobs.

The contents of the /tmp/sha_256_test.csv are shown below. It can be seen that the configuration used for the test and the associated statistics from the test run are listed:

```
ts.output.throughput.bytes, stats.input.throughput.rate, stats.output.throughput.rate
[doca_sha], 0, 0, 10000, 1000, 3, "random-data", 2048, d8:00.0, 100, [1], throughput, 0, "sha256", 2048, 1, 2, 1024, 128, 1 fragments, 7471270, 7471270, 15301160960, 23910931
```
Host-side SHA with CSV Appended Output File Sample

- This test invokes DOCA Bench on the Host side to execute the SHA operation using the SHA512 algorithm and to create a csv file containing the test configuration and statistics.

- The command is repeated with the added option of csv-append-mode. This instructs DOCA Bench to append the test run statistics to the existing csv file.

- A list of 1 core is provided with a count of 2 threads per core.

Command Line

1. Create the initial /tmp/sha_512_test.csv file:

```
doca_bench --core-list 2 \
    --threads-per-core 2 \
    --pipeline-steps doca_sha \
    --device d8:00.0 \
    --data-provider random-data \
    --uniform-job-size 2048 \
    --job-output-buffer-size 2048 \
    --run-limit-seconds 3 \
    --attribute doca_sha.algorithm=sha512 \
    --warm-up-jobs 100 \
    --csv-output-file /tmp/sha_512_test.csv
```

2. The second command is:

```
doca_bench --core-list 2 \
    --threads-per-core 2 \
    --pipeline-steps doca_sha \
    --device d8:00.0 \
    --data-provider random-data \
    --uniform-job-size 2048 \
    --job-output-buffer-size 2048 \
    --run-limit-seconds 3 \
    --attribute doca_sha.algorithm=sha512 \
    --warm-up-jobs 100 \
```

This causes DOCA Bench to append the configuration and statistics from the second command run to the /tmp/sha_512_test.csv file.

Results Output

This is a snapshot of the results output from the first command run:

Executing...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [1] started...
WT[1] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...
Stats for thread[0](core: 2)
  Duration: 3015185 micro seconds
  Enqueued jobs: 3590717
  Dequeued jobs: 3590717
  Throughput: 001.191 MOperations/s
  Ingress rate: 018.171 Gib/s
  Egress rate: 000.568 Gib/s
Stats for thread[1](core: 2)
  Duration: 3000203 micro seconds
  Enqueued jobs: 3656044
  Dequeued jobs: 3656044
  Throughput: 001.219 MOperations/s
  Ingress rate: 018.594 Gib/s
  Egress rate: 000.581 Gib/s
Aggregate stats
  Duration: 3015185 micro seconds
  Enqueued jobs: 7246761
  Dequeued jobs: 7246761
  Throughput: 002.403 MOperations/s
  Ingress rate: 036.673 Gib/s
  Egress rate: 001.146 Gib/s
This is a snapshot of the results output from the second command run:

```
Executing...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [1] started...
WT[1] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...
Stats for thread[0](core: 2)
  Duration: 3000072 micro seconds
  Enqueued jobs: 3602562
  Dequeued jobs: 3602562
  Throughput: 001.201 MOperations/s
  Ingress rate: 018.323 Gib/s
  Egress rate: 000.573 Gib/s
Stats for thread[1](core: 2)
  Duration: 3000062 micro seconds
  Enqueued jobs: 3659148
  Dequeued jobs: 3659148
  Throughput: 001.220 MOperations/s
  Ingress rate: 018.611 Gib/s
  Egress rate: 000.582 Gib/s
Aggregate stats
  Duration: 3000072 micro seconds
  Enqueued jobs: 7261710
  Dequeued jobs: 7261710
  Throughput: 002.421 MOperations/s
  Ingress rate: 036.934 Gib/s
  Egress rate: 001.154 Gib/s
```

**Results Overview**

Since a single core has been specified with a thread count of 2, there are statistics displayed for each thread as well as the aggregate statistics.

It can also be observed that 2 threads are started on core 1 with each thread executing the warm-up jobs.
The contents of the /tmp/sha_256_test.csv, after the first command has been run, are shown below. It can be seen that the configuration used for the test and the associated statistics from the test run are listed:

```
cfg.companion.connection_string,cfg.pipeline.steps,cfg.pipeline.use_remote_input_buffers,cfg.pipeline.use-size,cfg.send-queue-size,cfg.receive-queue-size,cfg.data-provider-input-file,cfg.attribute.mmo.log_qp_depth,cfg.attribute.mmo.log_num_qps,stats.input.job_count,stats.output.job,
[doca_sha],0,0,10000,1000,3,,random-data,2048,d8:00.0,,100,
[2],throughput,0,,,,sha512,2048,1,2,1024,128,1 fragments,,,,,,,7246761,7246761,14841366528,46385004
```

The contents of the /tmp/sha_256_test.csv, after the second command has been run, are shown below. It can be seen that a second entry has been added detailing the configuration used for the test and the associated statistics from the test run:

```
cfg.companion.connection_string,cfg.pipeline.steps,cfg.pipeline.use_remote_input_buffers,cfg.pipeline.use-size,cfg.send-queue-size,cfg.receive-queue-size,cfg.data-provider-input-file,cfg.attribute.mmo.log_qp_depth,cfg.attribute.mmo.log_num_qps,stats.input.job_count,stats.output.job,
[doca_sha],0,0,10000,1000,3,,random-data,2048,d8:00.0,,100,
[2],throughput,0,,,,sha512,2048,1,2,1024,128,1 fragments,,,,,,,7261710,7261710,14871982080,46480678
```

**BlueField-side SHA with Transient Statistics Sample**

- This test invokes DOCA Bench on the BlueField side to execute the SHA operation using the SHA1 algorithm and to display statistics every 2000 milliseconds during the test run

- A list of 3 cores is provided with a count of 2 threads per core and a core-count of 1

- The core-count instructs DOCA Bench to use the first core number in the core list, in this case core number 2

**Command Line**
```bash
doca_bench --core-list 2,3,4 \
   --core-count 1 \
   --threads-per-core 2 \
   --pipeline-steps doca_sha \
   --device 03:00.0 \
   --data-provider random-data \
   --uniform-job-size 2048 \
   --job-output-buffer-size 2048 \
   --run-limit-seconds 3 \
   -attribute doca_sha.algorithm=sha1 \
   --warm-up-jobs 100 \
   --rt-stats-interval 2000
```

## Results Output

Executing...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [1] started...
WT[1] Executing 100 warm-up tasks using 100 unique tasks

Stats for thread[0](core: 2)
- Duration: 965645 micro seconds
- Enqueued jobs: 1171228
- Dequeued jobs: 1171228
- Throughput: 001.213 MOperations/s
- Ingress rate: 018.505 Gib/s
- Egress rate: 000.181 Gib/s

Stats for thread[1](core: 2)
- Duration: 965645 micro seconds
- Enqueued jobs: 1171754
- Dequeued jobs: 1171754
- Throughput: 001.213 MOperations/s
- Ingress rate: 018.514 Gib/s
- Egress rate: 000.181 Gib/s

Aggregate stats
- Duration: 965645 micro seconds
- Enqueued jobs: 2342982
- Dequeued jobs: 2342982
- Throughput: 002.426 MOperations/s
Ingress rate: 037.019 Gib/s
Egress rate: 000.362 Gib/s
Stats for thread[0](core: 2)
  Duration: 2968088 micro seconds
  Enqueued jobs: 3653691
  Dequeued jobs: 3653691
  Throughput: 001.231 MOperations/s
  Ingress rate: 018.783 Gib/s
  Egress rate: 000.183 Gib/s
Stats for thread[1](core: 2)
  Duration: 2968088 micro seconds
  Enqueued jobs: 3689198
  Dequeued jobs: 3689198
  Throughput: 001.243 MOperations/s
  Ingress rate: 018.965 Gib/s
  Egress rate: 000.185 Gib/s
Aggregate stats
  Duration: 2968088 micro seconds
  Enqueued jobs: 7342889
  Dequeued jobs: 7342889
  Throughput: 002.474 MOperations/s
  Ingress rate: 037.748 Gib/s
  Egress rate: 000.369 Gib/s
Cleanup...
[main] Completed! tearing down...
Stats for thread[0](core: 2)
  Duration: 3000122 micro seconds
  Enqueued jobs: 3694128
  Dequeued jobs: 3694128
  Throughput: 001.231 MOperations/s
  Ingress rate: 018.789 Gib/s
  Egress rate: 000.184 Gib/s
Stats for thread[1](core: 2)
  Duration: 3000089 micro seconds
  Enqueued jobs: 3751128
  Dequeued jobs: 3751128
  Throughput: 001.250 MOperations/s
  Ingress rate: 019.079 Gib/s
  Egress rate: 000.186 Gib/s
Aggregate stats
  Duration: 3000122 micro seconds
  Enqueued jobs: 7445256
  Dequeued jobs: 7445256
  Throughput: 002.482 MOperations/s
Results Overview

Although a core list of 3 cores has been specified, the core-count value of 1 instructs DOCA Bench to use the first entry in the core list.

It can be seen that as a thread-count of 2 has been specified, there are 2 threads created on core 2.

A transient statistics interval of 2000 milliseconds has been specified, and the transient statistics per thread can be seen, as well as the final aggregate statistics.

Host-side Local DMA with Core Sweep Sample

- This test invokes DOCA Bench to execute a local DMA operation on the host
- It specifies that a core sweep should be carried out using core counts of 1, 2, and 4 using the option `--sweep core-count,1,4,*2`
- Test output is to be saved in a CSV file `/tmp/dma_sweep.csv` and a filter is applied so that only statistics information is recorded. No configuration information is to be recorded.

Command Line

doca_bench --core-mask 0xff \  
   --sweep core-count,1,4,*2 \  
   --pipeline-steps doca_dma \  
   --device d8:00.0 \  
   --data-provider random-data \  
   --uniform-job-size 2048 \  
   --job-output-buffer-size 2048 \  
   --run-limit-seconds 5 \  
   --csv-output-file /tmp/dma_sweep.csv \
Results Overview

Test permutations: [
  Attributes: []
  Uniform job size: 2048
  Core count: 1
  Per core thread count: 1
  Task pool size: 1024
  Data provider job count: 128
  MTU size: -- not configured --
  SQ depth: -- not configured --
  RQ depth: -- not configured --
  Input data file: -- not configured --
-----------------------------
  Attributes: []
  Uniform job size: 2048
  Core count: 2
  Per core thread count: 1
  Task pool size: 1024
  Data provider job count: 128
  MTU size: -- not configured --
  SQ depth: -- not configured --
  RQ depth: -- not configured --
  Input data file: -- not configured --
-----------------------------
  Attributes: []
  Uniform job size: 2048
  Core count: 4
  Per core thread count: 1
  Task pool size: 1024
  Data provider job count: 128
  MTU size: -- not configured --
  SQ depth: -- not configured --
  RQ depth: -- not configured --
  Input data file: -- not configured --
]

[main] Initialize framework...
[main] Start execution...
Preparing permutation 1 of 3...
Executing permutation 1 of 3...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup permutation 1 of 3...
Aggregate stats
  Duration:   5000191 micro seconds
  Enqueued jobs:  22999128
  Dequeued jobs:  22999128
  Throughput:  004.600 MOperations/s
  Ingress rate:  070.185 Gib/s
  Egress rate:  070.185 Gib/s
Preparing permutation 2 of 3...
Executing permutation 2 of 3...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [1] started...
WT[1] Executing 100 warm-up tasks using 100 unique tasks
Cleanup permutation 2 of 3...
Stats for thread[0](core: 0)
  Duration:   5000066 micro seconds
  Enqueued jobs:  14409794
  Dequeued jobs:  14409794
  Throughput:  002.882 MOperations/s
  Ingress rate:  043.975 Gib/s
  Egress rate:  043.975 Gib/s
Stats for thread[1](core: 1)
  Duration:   5000188 micro seconds
  Enqueued jobs:  14404708
  Dequeued jobs:  14404708
  Throughput:  002.881 MOperations/s
  Ingress rate:  043.958 Gib/s
  Egress rate:  043.958 Gib/s
Aggregate stats
  Duration:   5000188 micro seconds
  Enqueued jobs:  28814502
  Dequeued jobs:  28814502
  Throughput:  005.763 MOperations/s
  Ingress rate:  087.932 Gib/s
  Egress rate:  087.932 Gib/s
Preparing permutation 3 of 3...
Executing permutation 3 of 3...
Data path thread [1] started...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Data path thread [3] started...
WT[3] Executing 100 warm-up tasks using 100 unique tasks
Cleanup permutation 3 of 3...
[main] Completed! tearing down...
Stats for thread[0](core: 0)
  Duration: 5000092 micro seconds
  Enqueued jobs: 7227025
  Dequeued jobs: 7227025
  Throughput: 001.445 MOperations/s
  Ingress rate: 022.055 Gib/s
  Egress rate: 022.055 Gib/s
Stats for thread[1](core: 1)
  Duration: 5000081 micro seconds
  Enqueued jobs: 7223269
  Dequeued jobs: 7223269
  Throughput: 001.445 MOperations/s
  Ingress rate: 022.043 Gib/s
  Egress rate: 022.043 Gib/s
Stats for thread[2](core: 2)
  Duration: 5000047 micro seconds
  Enqueued jobs: 7229678
  Dequeued jobs: 7229678
  Throughput: 001.446 MOperations/s
  Ingress rate: 022.063 Gib/s
  Egress rate: 022.063 Gib/s
Stats for thread[3](core: 3)
  Duration: 5000056 micro seconds
  Enqueued jobs: 7223037
  Dequeued jobs: 7223037
  Throughput: 001.445 MOperations/s
  Ingress rate: 022.043 Gib/s
  Egress rate: 022.043 Gib/s
Aggregate stats
  Duration: 5000092 micro seconds
  Enqueued jobs: 28903009
  Dequeued jobs: 28903009
  Throughput: 005.780 MOperations/s
  Ingress rate: 088.203 Gib/s
  Egress rate: 088.203 Gib/s
Results Overview

The output gives a summary of the permutations being carried out and then proceeds to display the statistics for each of the permutations.

The CSV output file contents can be seen to contain only statistics information. Configuration information is not included.

There is an entry for each of the sweep permutations:

```
stats.input.job_count,stats.output.job_count,stats.input.byte_count,stats.output.byte_count,stats.input.thr
22999128,22999128,47102214144,47102214144,070.185 Gib/s,070.185 Gib/s,4.599650 MOperations/s
28814502,28814502,59012100096,59012100096,087.932 Gib/s,087.932 Gib/s,5.762683 MOperations/s
28903009,28903009,59193362432,59193362432,088.203 Gib/s,088.203 Gib/s,5.780495 MOperations/s
```

Host-side Local DMA with Job Size Sweep Sample

This test invokes DOCA Bench to execute a local DMA operation on the host.

It specifies that a uniform job size sweep should be carried out using job sizes 1024 and 2048 using the option --sweep uniform-job-size,1024,2048.

Test output is to be saved in a CSV file /tmp/dma_sweep_job_size.csv and collection of environment information is enabled.

Command Line

```
doca_bench --core-mask 0xff
    --core-count 1
    --pipeline-steps doca_dma
    --device d8:00.0
    --data-provider random-data
    --sweep uniform-job-size,1024,2048
    --job-output-buffer-size 2048
```
Results Overview

Test permutations: [
    Attributes: []
    Uniform job size: 1024
    Core count: 1
    Per core thread count: 1
    Task pool size: 1024
    Data provider job count: 128
    MTU size: -- not configured --
    SQ depth: -- not configured --
    RQ depth: -- not configured --
    Input data file: -- not configured --
    -----------------------------
    Attributes: []
    Uniform job size: 2048
    Core count: 1
    Per core thread count: 1
    Task pool size: 1024
    Data provider job count: 128
    MTU size: -- not configured --
    SQ depth: -- not configured --
    RQ depth: -- not configured --
    Input data file: -- not configured --
]

[main] Initialize framework...
[main] Start execution...
Preparing permutation 1 of 2...
Executing permutation 1 of 2...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup permutation 1 of 2...
Aggregate stats
    Duration: 5000083 micro seconds
    Enqueued jobs: 23645128
Results Overview

The output gives a summary of the permutations being carried out and then proceeds to display the statistics for each of the permutations.

The CSV output file contents can be seen to contain statistics information and the environment information.

There is an entry for each of the sweep permutations.
BlueField-side Remote DMA Sample

- This test invokes DOCA Bench to execute a remote DMA operation on the host
- It specifies the companion connection details to be used on the host and that remote output buffers are to be used

Command Line

doca_bench --core-list 12 \ 
   --pipeline-steps doca_dma \ 
   --device 03:00.0 \ 
   --data-provider random-data \ 
   --uniform-job-size 2048 \ 
   --job-output-buffer-size 2048 \ 
   --use-remote-output-buffers \ 
   --companion-connection-string \ 
   proto=tcp,port=12345,mode=host,dev=17:00.0,user=bob,addr=10.10.10.10 \ 
   --run-limit-seconds 5

Results Overview

Executing...
Worker thread[0](core: 12) [doca_dma] started...
Worker thread[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...
Aggregate stats
   Duration: 5000073 micro seconds
   Enqueued jobs: 32202128
   Dequeued jobs: 32202128
Results Overview

None.

Compress BlueField-side Sample

Note

This test is relevant for BlueField-2 only.

- This test invokes DOCA Bench to run compression using random data as input
- The compression algorithm specified is "deflate"

Command Line

doca_bench  --core-list 2 \ 
    --pipeline-steps doca_compress::compress \ 
    --device 03:00.0 \ 
    --data-provider random-data \ 
    --uniform-job-size 2048 \ 
    --job-output-buffer-size 4096 \ 
    --run-limit-seconds 3 \ 
    --attribute doca_compress.algorithm="deflate"
Result Output

Executing...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...

Aggregate stats
  Duration:  3000146 micro seconds
  Enqueued jobs: 5340128
  Dequeued jobs: 5340128
  Throughput:  001.780 MOperations/s
  Ingress rate:  027.160 Gib/s
  Egress rate:  027.748 Gib/s

Results Overview

None

BlueField-side Decompress LZ4 Sample

- This test invokes DOCA Bench to run decompression using random data as input
- This test specifies a data provider of file set which contains the filename of an LZ4 compressed file
- Remote input buffers are specified to be used for the input jobs
- It specifies the companion connection details to be used on the host for the remote input buffers

Command Line

doca_bench --core-list 12 \
--pipeline-steps doca_compress::decompress \ 
--device 03:00.0 \ 
--data-provider file-set \ 
--data-provider-input-file lz4_compressed_64b_buffers.fs \ 
--job-output-buffer-size 4096 \ 
--run-limit-seconds 3 \ 
--attribute doca_compress.algorithm="lz4" \ 
--use-remote-output-buffers \ 
--companion-connection-string proto=tcp,port=12345,mode=host,dev=17:00.0,user=bob,addr=10.10.10.10

Results Output

Executing...
Worker thread[0](core: 12) [doca_compress::decompress] started...
Worker thread[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup...
[main] Completed! tearing down...
Aggregate stats
  Duration: 3000043 micro seconds
  Enqueued jobs: 15306128
  Dequeued jobs: 15306128
  Throughput: 005.102 MOperations/s
  Ingress rate: 003.155 Gib/s
  Egress rate: 002.433 Gib/s

Results Comment

None

Host-side EC Creation in Bulk Latency Mode Sample

- This test invokes DOCA Bench to run the EC creation step.
• It runs in bulk latency mode and specifies the `data` attributes of `data_block_count`, `redundancy_block_count`, and `matrix_type`.

**Command Line**

doca_bench --mode bulk-latency 
  --core-list 12 
  --pipeline-steps doca_ec::create 
  --device 17:00.0 
  --data-provider random-data 
  --uniform-job-size 1024 
  --job-output-buffer-size 1024 
  --run-limit-seconds 3 
  --attribute doca_ec.data_block_count=16 
  --attribute doca_ec.redundancy_block_count=16 
  --attribute doca_ec.matrix_type=cauchy

**Results Output**

Bulk latency output will be similar to that presented in section "BlueField-side Decompress LZ4 Sample".

**Results Comment**

Bulk latency output will be similar to that presented earlier on this page.

**BlueField-side EC Creation in Precision Latency Mode Sample**

• This test invokes DOCA Bench to run the EC creation step

• It runs in precision latency mode and specifies the `data` attributes of `data_block_count`, `redundancy_block_count`, and `matrix_type`
Command Line

doca_bench --mode precision-latency \  
  --core-list 12 \  
  --pipeline-steps doca_ec::create \  
  --device 03:00.0 \  
  --data-provider random-data \  
  --uniform-job-size 1024 \  
  --job-output-buffer-size 1024 \  
  --run-limit-jobs 5000 \  
  --attribute doca_ec.data_block_count=16 \  
  --attribute doca_ec.redundancy_block_count=16 \  
  --attribute doca_ec.matrix_type=cauchy

Results Output

None

Results Comment

Precision latency output will be similar to that presented earlier on this page.

Comch Consumer from Host Side Sample

- This test invokes DOCA Bench in Comch consumer mode using a core-list on host side and BlueField side
- The run-limit is 500 jobs

Command Line
Results Output

[main] Completed! tearing down...
Aggregate stats
  Duration: 1415 micro seconds
  Enqueued jobs: 500
  Dequeued jobs: 500
  Throughput: 000.353 MOperations/s
  Ingress rate: 000.000 Gib/s
  Egress rate: 010.782 Gib/s

Results Comment

The aggregate statistics show the test completed after 500 jobs were processed.

Host-side Comch Producer Sample

- This test invokes DOCA Bench in Comch producer mode using a core-mask on the host side and BlueField side

- The run-limit is 1000 jobs

Command Line

./doca_bench --core-list 4 --warm-up-jobs 32 --pipeline-steps doca_comch::consumer --device ca:00.0 --data-provider random-data --run-limit-jobs 500 --core-count 1 --uniform-job-size 4096 --job-output-buffer-size 4096 --companion-connection-string proto=tcp,mode=dpu,dev=03:00.0,user=bob,addr=10.10.10.10,port=12345 --attribute dopt.companion_app.path=<path to DPU doca_bench_companion application location> --data-provider-job-count 256 --companion-core-list 12
doca_bench --core-list 4
    --warm-up-jobs 32
    --pipeline-steps doca_comch::producer
    --device ca:00.0
    --data-provider random-data
    --run-limit-jobs 500
    --core-count 1
    --uniform-job-size 4096
    --job-output-buffer-size 4096
    --companion-connection-string
    proto=tcp,mode=dpu,dev=03:00.0,user=bob,addr=10.10.10.10,port=12345
    --attribute dopt.companion_app.path=<path to DPU doca_bench_companion location>
    --data-provider-job-count 256
    --companion-core-list 12

Results Overview

[main] Completed! tearing down...
Aggregate stats
    Duration: 407 micro seconds
    Enqueued jobs: 500
    Dequeued jobs: 500
    Throughput: 001.226 MOperations/s
    Ingress rate: 037.402 Gib/s
    Egress rate: 000.000 Gib/s

Results Comment

The aggregate statistics show the test completed after 500 jobs were processed.

Host-side RDMA Send Sample
• This test invokes DOCA Bench in RDMA send mode using a core-list on the send and receive side

• The send queue size is configured to 50 entries

**Command Line**

```bash
doca_bench --pipeline-steps doca_rdma::send \
    --device d8:00.0 \
    --data-provider random-data \
    --uniform-job-size 2048 \
    --job-output-buffer-size 2048 \
    --run-limit-seconds 3 \
    --send-queue-size 50 \
    --companion-connection-string proto=tcp,addr=10.10.10.10,port=12345,user=bob,dev=ca:00.0 \
    --companion-core-list 12 \
    --core-list 12
```

**Results Output**

Test permutations: [
  Attributes: []
  Uniform job size: 2048
  Core count: 1
  Per core thread count: 1
  Task pool size: 1024
  Data provider job count: 128
  MTU size: -- not configured --
  SQ depth: 50
  RQ depth: -- not configured --
  Input data file: -- not configured --
]

**Results Comment**
The configuration output shows the send queue size configured to 50.

**Host-side RDMA Receive Sample**

- This test invokes DOCA Bench in RDMA receive mode using a core-list on the send and receive side
- The receive queue size is configured to 100 entries

**Command Line**

```bash
doca_bench --pipeline-steps doca_rdma::receive \
  --device d8:00.0 \ 
  --data-provider random-data \ 
  --uniform-job-size 2048 \ 
  --job-output-buffer-size 2048 \ 
  --run-limit-seconds 3 \ 
  --receive-queue-size 100 \ 
  --companion-connection-string proto=tcp,addr=10.10.10.10,port=12345,user=bob,dev=ca:00.0 \ 
  --companion-core-list 12 \ 
  --core-list 12
```

**Results Output**

Test permutations: [
  Attributes: []
  Uniform job size: 2048
  Core count: 1
  Per core thread count: 1
  Task pool size: 1024
  Data provider job count: 128
  MTU size: -- not configured --
  SQ depth: -- not configured --
  RQ depth: 100
  Input data file: -- not configured --
]
Results Overview

The configuration output shows the receive queue size configured to 100.
NVIDIA DOCA Capabilities Print Tool

This document provides instruction on the usage of the DOCA Capabilities Print Tool.

Introduction

This tool is used to print all the available DOCA libraries and devices. For each DOCA device, the tool prints its representor devices and the capabilities it supports in each DOCA library.

Prerequisites

DOCA 2.6.0 and higher.

Description

This tool can be executed on the host or Arm sides.

The following capabilities are supported by this tool:

- DOCA device list – print the PCIe device of every available DOCA device and its capabilities
- DOCA representor device list – for every DOCA device, print the PCIe device of every available DOCA representor device and its capabilities
- DOCA library list – print the available DOCA libraries supported by the running OS and their availability for specific OSs
- DOCA library capabilities – for every DOCA device, print the capabilities it supports in every DOCA library

Execution
To print all the available DOCA devices and their capabilities, run:

```
/opt/mellanox/doca/tools/doca_caps --list-devs
```

**Info**

Printing the capabilities of a specific DOCA device can be done using the `--pci-addr` flag.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-devs
PCI: 0000:03:00.0
  ibdev_name         mlx5_0
  iface_name         p0
  mac_addr           94:6d:ae:5c:9e:04
  ipv4_addr          0.0.0.0
  ipv6_addr          fe80:0000:0000:0000:966d:aeff:fe5c:9e04
  gid_table_size     255
  GID[0]             fe80:0000:0000:0000:966d:aeff:fe5c:9e04
PCI: 0000:03:00.1
  ibdev_name         mlx5_1
  iface_name         p1
  mac_addr           94:6d:ae:5c:9e:05
  ipv4_addr          0.0.0.0
  ipv6_addr          fe80:0000:0000:0000:966d:aeff:fe5c:9e05
  gid_table_size     255
  GID[0]             fe80:0000:0000:0000:966d:aeff:fe5c:9e05
PCI: 0000:03:00.0
  ibdev_name         mlx5_2
  iface_name         enp3s0f0s0
  mac_addr           02:c6:d0:fd:56:d7
  ipv4_addr          0.0.0.0
  ipv6_addr          fe80:0000:0000:0000:00c6:d0ff:fe6d:56d7
  gid_table_size     255
  GID[0]             fe80:0000:0000:0000:00c6:d0ff:fe6d:56d7
```
To print all the available DOCA representor devices and their capabilities, run:

```
/opt/mellanox/doca/tools/doca_caps --list-rep-devs
```

**Info**

This command is available only on the Arm side.

**Info**

Printing the representor list of a specific DOCA device can be done using the `--pci-addr` flag.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-rep-devs
PCI: 0000:03:00.0
    representor-PCI: 0000:3b:00.0
        pci_func_type  PF
        hotplug        no
        vuid           MT2308XZ0BN0MLNXS0D0F0
    representor-PCI: 0000:3b:00.0
```
To print all the supported DOCA libraries by the OS and their availability status, run:

```
/opt/mellanox/doca/tools/doca_caps --list-libs
```

**Info**

Different OSs may support different DOCA libraries.

Example output:

```
/opt/mellanox/doca/tools/doca_caps --list-libs
  common                   installed
  aes_gcm                  installed
  apsh                     installed
  argp                     installed
  cc                       installed
  comm_channel             installed
  compress                 installed
  dma                      installed
  dpa                      installed
```
To print all the capabilities for all the available libraries, that have capabilities, for every DOCA device, run:

```
/opt/mellanox/doca/tools/doca_caps
```

---

**Info**

Printing the capabilities of one specific DOCA device can be done using the `--pci-addr` flag.

---

**Info**

Printing the capabilities of one specific DOCA library can be done using the `--lib` flag.

---

Example output:

```
/opt/mellanox/doca/tools/doca_caps
PCI: 0000:03:00.0
```
common
  mmap_export_pci supported
  mmap_create_from_export_pci supported
  hotplug_manager unsupported
  rep_filter_all supported
  rep_filter_net supported
  rep_filter_emulated unsupported
aes_gcm
  task_encrypt supported
  task_encrypt_get_max_iv_len 12
  task_encrypt_tag_96 supported
  task_encrypt_tag_128 supported
  task_encrypt_128b_key supported
  task_encrypt_256b_key supported
  task_encrypt_max_buf_size 2097152
  task_encrypt_max_list_buf_num_elem 128
  task_decrypt supported
  task_decrypt_get_max_iv_len 12
  task_decrypt_tag_96 supported
  task_decrypt_tag_128 supported
  task_decrypt_128b_key supported
  task_decrypt_256b_key supported
  task_decrypt_max_buf_size 2097152
  task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
cc
  server supported
  client supported
  max_name_len 120
  max_msg_size 4080
  max_recv_queue_size 8192
  max_send_tasks 8192
  maxclients 512
  consumer supported
  consumer_max_num_tasks 65536
  consumer_max_buf_size 2097152
  producer supported
  producer_max_num_tasks 65536
  producer_max_buf_size 2097152
comm_channel
  max_service_name_len 120
  max_message_size 4080
  max_send_queue_size 8192
  max_recv_queue_size 8192
service_max_num_connections 512
compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0
task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536
dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536
dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time 12
erasure_coding
task_galois_mul supported
task_create supported
task_update supported
task_recover supported
max_block_size 1048576
max_buf_list_len 128
eth
rxq_cyclic_cpu unsupported
rxq_cyclic_gpu supported
rxq_managed_mempool_cpu unsupported
rxq_managed_mempool_gpu supported
rxq_regular_cpu unsupported
rxq_regular_gpu supported
rxq_max_recv_buf_list_len 32
rxq_max_packet_size 16384
rxq_max_burst_size 32768
txq_regular_cpu unsupported
txq_regular_gpu supported
txq_max_send_buf_list_len 48
txq_max_lso_header_size 256
txq_txq_max_lso_msg_size 262144
txq_l3_chksum_offload supported
<table>
<thead>
<tr>
<th>Feature</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>txq_l4_chksum_offload</td>
<td>supported</td>
</tr>
<tr>
<td>txq_wait_on_time_type</td>
<td>unsupported</td>
</tr>
<tr>
<td>flow_ct</td>
<td></td>
</tr>
<tr>
<td>flow_ct</td>
<td>supported</td>
</tr>
<tr>
<td>ipsec</td>
<td></td>
</tr>
<tr>
<td>task_sa_create</td>
<td>supported</td>
</tr>
<tr>
<td>task_sa_destroy</td>
<td>supported</td>
</tr>
<tr>
<td>nvrd_transport</td>
<td></td>
</tr>
<tr>
<td>task_write</td>
<td>supported</td>
</tr>
<tr>
<td>rc_max_src_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>dc_max_src_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>pcc</td>
<td></td>
</tr>
<tr>
<td>pcc</td>
<td>unsupported</td>
</tr>
<tr>
<td>pcc_np</td>
<td>unsupported</td>
</tr>
<tr>
<td>min_num_threads</td>
<td>0</td>
</tr>
<tr>
<td>max_num_threads</td>
<td>0</td>
</tr>
<tr>
<td>rdma</td>
<td></td>
</tr>
<tr>
<td>task_send</td>
<td>supported</td>
</tr>
<tr>
<td>task_send_imm</td>
<td>supported</td>
</tr>
<tr>
<td>task_read</td>
<td>supported</td>
</tr>
<tr>
<td>task_write</td>
<td>supported</td>
</tr>
<tr>
<td>task_write_imm</td>
<td>supported</td>
</tr>
<tr>
<td>task_atomic_cmp_swp</td>
<td>supported</td>
</tr>
<tr>
<td>task_atomic_fetch_add</td>
<td>supported</td>
</tr>
<tr>
<td>task_receive</td>
<td>supported</td>
</tr>
<tr>
<td>rc_transport_type</td>
<td>supported</td>
</tr>
<tr>
<td>dc_transport_type</td>
<td>unsupported</td>
</tr>
<tr>
<td>rc_task_receive_get_max_dst_buf_list_len</td>
<td>31</td>
</tr>
<tr>
<td>dc_task_receive_get_max_dst_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>task_remote_net_sync_event_get</td>
<td>supported</td>
</tr>
<tr>
<td>task_remote_net_sync_event_notify_set</td>
<td>supported</td>
</tr>
<tr>
<td>task_remote_net_sync_event_notify_add</td>
<td>supported</td>
</tr>
<tr>
<td>max_send_queue_size</td>
<td>32768</td>
</tr>
<tr>
<td>max_recv_queue_size</td>
<td>32768</td>
</tr>
<tr>
<td>max_send_buf_list_len</td>
<td>13</td>
</tr>
<tr>
<td>max_message_size</td>
<td>1073741824</td>
</tr>
<tr>
<td>sha</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha256</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha512</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha1_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha256_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha512_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>max_list_num_elem</td>
<td>0</td>
</tr>
</tbody>
</table>
max_src_buf_size 0
sha1_min_dst_buf_size 0
sha256_min_dst_buf_size 0
sha512_min_dst_buf_size 0
sha1_partial_hash_block_size 0
sha256_partial_hash_block_size 0
sha512_partial_hash_block_size 0
PCI: 0000:03:00.1
common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported
rep_filter_all supported
rep_filter_net supported
rep_filter_emulated unsupported
aes_gcm
task_encrypt supported
task_encrypt_get_max_iv_len 12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported
task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size 2097152
task_encrypt_max_list_buf_num_elem 128
task_decrypt supported
task_decrypt_get_max_iv_len 12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536
c
server supported
client supported
max_name_len 120
max_msg_size 4080
max_recv_queue_size 8192
max_send_tasks 8192
max_clients 512
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152
producer                                      supported
producer_max_num_tasks                        65536
producer_max_buf_size                         2097152

comm_channel
max_service_name_len                          120
max_message_size                              4080
max_send_queue_size                           8192
max_recv_queue_size                           8192
service_max_num_connections                   512

compress
task_compress_deflate                         unsupported
task_compress_deflate_get_max_buf_size        0
task_compress_deflate_get_max_buf_list_len    0

task_decompress_deflate                       supported

task_decompress_deflate_get_max_buf_size      2097152

task_decompress_deflate_get_max_buf_list_len  128

task_decompress_lz4                           supported

task_decompress_lz4_get_max_buf_size          2097152

task_decompress_lz4_get_max_buf_list_len      128

max_num_tasks                                 65536

dma

task_memcpy                                   supported
max_buf_size                                  2097152
max_buf_list_len                              64
max_num_tasks                                 65536

dpa

dpa                                           supported
max_threads_per_kernel                        128
kernel_max_run_time                           12

erasure_coding

task_galois_mul                               supported

task_create                                   supported

task_update                                   supported

task_recover                                  supported
max_block_size                                1048576
max_buf_list_len                              128

eth
	rxq_cyclic_cpu                                unsupported
rxq_cyclic_gpu                                 supported

rxq_managed_mempool_cpu                       unsupported
rxq.managed.mempool.gpu                        supported
rxq_regular_cpu                               unsupported
rxq_regular_gpu                               supported
rxq_max_recv_buf_list_len                     32
rxq_max_packet_size: 16384
rxq_max_burst_size: 32768
txq_regular_cpu: unsupported
txq_regular_gpu: supported
rxq_max_send_buf_list_len: 48
rxq_max_lso_header_size: 256
txqTxq max lso msg size: 262144
rxq_l3_checksum offload: supported
txq_l4_checksum offload: supported
txq_wait_on_time_type: unsupported

flow_ct
flow_ct: supported

ipsec
task_sa_create: supported
task_sa_destroy: supported

nvrd_transport
nvrd_transport: supported
task_write: supported
rc_max_src_buf_list_len: 0
dc_max_src_buf_list_len: 0

pcc
pcc: unsupported
pcc_np: unsupported
min_num_threads: 0
max_num_threads: 0

rdma
rma: supported
task_send: supported
task_send_imm: supported
task_read: supported
task_write: supported
task_write_imm: supported
task_atomic_cmp_swap: supported
task_atomic_fetch_add: supported
task_receive: supported
task_send_imm: supported
dc_transport_type: unsupported
task_receive_get_max_dst_buf_list_len: 31
dc_task_receive_get_max_dst_buf_list_len: 0
task_remote_net_sync_event_get: supported
task_remote_net_sync_event_notify_set: supported
task_remote_net_sync_event_notify_add: supported
max_send_queue_size: 32768
max_recv_queue_size: 32768
max_send_buf_list_len: 13
max_message_size: 1073741824
sha
  sha1 unsupported
  sha256 unsupported
  sha512 unsupported
  sha1_partial unsupported
  sha256_partial unsupported
  sha512_partial unsupported
  max_list_num_elem 0
  max_src_buf_size 0
  sha1_min_dst_buf_size 0
  sha256_min_dst_buf_size 0
  sha512_min_dst_buf_size 0
  sha1_partial_hash_block_size 0
  sha256_partial_hash_block_size 0
  sha512_partial_hash_block_size 0

PCI: 0000:03:00.0

common
  mmap_export_pci supported
  mmap_create_from_export_pci supported
  hotplug_manager unsupported
  rep_filter_all unsupported
  rep_filter_net unsupported
  rep_filter_emulated unsupported

aes_gcm
  task_encrypt supported
  task_encrypt_get_max_iv_len 12
  task_encrypt_tag_96 supported
  task_encrypt_tag_128 supported
  task_encrypt_128b_key supported
  task_encrypt_256b_key supported
  task_encrypt_max_buf_size 2097152
  task_encrypt_max_list_buf_num_elem 128
  task_decrypt supported
  task_decrypt_get_max_iv_len 12
  task_decrypt_tag_96 supported
  task_decrypt_tag_128 supported
  task_decrypt_128b_key supported
  task_decrypt_256b_key supported
  task_decrypt_max_buf_size 2097152
  task_decrypt_max_list_buf_num_elem 128
  max_num_tasks 65536

cc
  server unsupported
  client supported
max_name_len  120
max_msg_size  4080
max_recv_queue_size  8192
max_send_tasks  8192
max_clients  0
consumer  supported
consumer_max_num_tasks  65536
consumer_max_buf_size  2097152
producer  supported
producer_max_num_tasks  65536
producer_max_buf_size  2097152
comm_channel
max_service_name_len  120
max_message_size  4080
max_send_queue_size  8192
max_recv_queue_size  8192
service_max_num_connections  0
compress
task_compress_deflate  unsupported
task_compress_deflate_get_max_buf_size  0
task_compress_deflate_get_max_buf_list_len  0
task_decompress_deflate  supported
task_decompress_deflate_get_max_buf_size  2097152
task_decompress_deflate_get_max_buf_list_len  128
task_decompress_lz4  supported
task_decompress_lz4_get_max_buf_size  2097152
task_decompress_lz4_get_max_buf_list_len  128
max_num_tasks  65536
dma
task_memcpy  supported
max_buf_size  2097152
max_buf_list_len  64
max_num_tasks  65536
dpa
dpa  supported
max_threads_per_kernel  128
kernel_max_run_time  12
erasure_coding
task_galois_mul  supported
task_create  supported
task_update  supported
task_recover  supported
max_block_size  1048576
max_buf_list_len  128
<table>
<thead>
<tr>
<th>Function</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>eth</td>
<td></td>
</tr>
<tr>
<td>rxq_cyclic_cpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_cyclic_gpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_managed_mempool_cpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_managed_mempool_gpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_regular_cpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_regular_gpu</td>
<td>supported</td>
</tr>
<tr>
<td>rxq_max_recv_buf_list_len</td>
<td>32</td>
</tr>
<tr>
<td>rxq_max_packet_size</td>
<td>16384</td>
</tr>
<tr>
<td>rxq_max_burst_size</td>
<td>32768</td>
</tr>
<tr>
<td>txq_regular_cpu</td>
<td>supported</td>
</tr>
<tr>
<td>txq_regular_gpu</td>
<td>supported</td>
</tr>
<tr>
<td>txq_max_send_buf_list_len</td>
<td>48</td>
</tr>
<tr>
<td>txq_max_lso_header_size</td>
<td>256</td>
</tr>
<tr>
<td>txq_txq_max_lso_msg_size</td>
<td>262144</td>
</tr>
<tr>
<td>txq_l3_chksum_offload</td>
<td>supported</td>
</tr>
<tr>
<td>txq_l4_chksum_offload</td>
<td>supported</td>
</tr>
<tr>
<td>txq_wait_on_time_type</td>
<td>unsupported</td>
</tr>
<tr>
<td>flow_ct</td>
<td></td>
</tr>
<tr>
<td>flow_ct</td>
<td>unsupported</td>
</tr>
<tr>
<td>ipsec</td>
<td></td>
</tr>
<tr>
<td>task_sa_create</td>
<td>unsupported</td>
</tr>
<tr>
<td>task_sa_destroy</td>
<td>unsupported</td>
</tr>
<tr>
<td>nvrd_transport</td>
<td></td>
</tr>
<tr>
<td>task_write</td>
<td>supported</td>
</tr>
<tr>
<td>rc_max_src_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>dc_max_src_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>pcc</td>
<td></td>
</tr>
<tr>
<td>pcc</td>
<td>unsupported</td>
</tr>
<tr>
<td>pcc_np</td>
<td>unsupported</td>
</tr>
<tr>
<td>min_num_threads</td>
<td>0</td>
</tr>
<tr>
<td>max_num_threads</td>
<td>0</td>
</tr>
<tr>
<td>rdma</td>
<td></td>
</tr>
<tr>
<td>task_send</td>
<td>supported</td>
</tr>
<tr>
<td>task_send_imm</td>
<td>supported</td>
</tr>
<tr>
<td>task_read</td>
<td>supported</td>
</tr>
<tr>
<td>task_write</td>
<td>supported</td>
</tr>
<tr>
<td>task_write_imm</td>
<td>supported</td>
</tr>
<tr>
<td>task_ATOMIC_CMP_SWAP</td>
<td>supported</td>
</tr>
<tr>
<td>task_atomic_fetch_add</td>
<td>supported</td>
</tr>
<tr>
<td>task_receive</td>
<td>supported</td>
</tr>
<tr>
<td>rc_transport_type</td>
<td>supported</td>
</tr>
<tr>
<td>dc_transport_type</td>
<td>unsupported</td>
</tr>
<tr>
<td>rc_task_receive_get_max_dst_buf_list_len</td>
<td>31</td>
</tr>
</tbody>
</table>
dc_task_receive_get_max_dst_buf_list_len  0
task_remote_net_sync_event_get supported
task_remote_net_sync_event_notify_set supported
task_remote_net_sync_event_notify_add supported
max_send_queue_size  32768
max_recv_queue_size  32768
max_send_buf_list_len  13
max_message_size  1073741824

sha
sha1 unsupported
sha256 unsupported
sha512 unsupported
sha1_partial unsupported
sha256_partial unsupported
sha512_partial unsupported
max_list_num_elem  0
max_src_buf_size  0
sha1_min_dst_buf_size  0
sha256_min_dst_buf_size  0
sha512_min_dst_buf_size  0
sha1_partial_hash_block_size  0
sha256_partial_hash_block_size  0
sha512_partial_hash_block_size  0

PCI: 0000:03:00.1

common
mmap_export_pci supported
mmap_create_from_export_pci supported
hotplug_manager unsupported
rep_filter_all unsupported
rep_filter_net unsupported
rep_filter_emulated unsupported

aes_gcm

task_encrypt supported
task_encrypt_get_max_iv_len  12
task_encrypt_tag_96 supported
task_encrypt_tag_128 supported
task_encrypt_128b_key supported
task_encrypt_256b_key supported
task_encrypt_max_buf_size  2097152
task_encrypt_max_list_buf_num_elem  128
task_decrypt supported
task_decrypt_get_max_iv_len  12
task_decrypt_tag_96 supported
task_decrypt_tag_128 supported
task_decrypt_128b_key supported
task_decrypt_256b_key supported
task_decrypt_max_buf_size 2097152
task_decrypt_max_list_buf_num_elem 128
max_num_tasks 65536

cc
server unsupported
client supported
max_name_len 120
max_msg_size 4080
max_recv_queue_size 8192
max_send_tasks 8192
max_clients 0
consumer supported
consumer_max_num_tasks 65536
consumer_max_buf_size 2097152
producer supported
producer_max_num_tasks 65536
producer_max_buf_size 2097152

comm_channel
max_service_name_len 120
max_message_size 4080
max_send_queue_size 8192
max_recv_queue_size 8192
service_max_num_connections 0

compress
task_compress_deflate unsupported
task_compress_deflate_get_max_buf_size 0
task_compress_deflate_get_max_buf_list_len 0
task_decompress_deflate supported
task_decompress_deflate_get_max_buf_size 2097152
task_decompress_deflate_get_max_buf_list_len 128
task_decompress_lz4 supported
task_decompress_lz4_get_max_buf_size 2097152
task_decompress_lz4_get_max_buf_list_len 128
max_num_tasks 65536

dma
task_memcpy supported
max_buf_size 2097152
max_buf_list_len 64
max_num_tasks 65536

dpa
dpa supported
max_threads_per_kernel 128
kernel_max_run_time                           12
erasure_coding
  task_galois_mul                        supported
  task_create                           supported
  task_update                           supported
  task_recover                          supported
max_block_size                         1048576
max_buf_list_len                        128
eth
  rxq_cyclic_cpu                        supported
  rxq_cyclic_gpu                        supported
  rxq_managed_mempool_cpu               supported
  rxq_managed_mempool_gpu               supported
  rxq_regular_cpu                       supported
  rxq_regular_gpu                       supported
  rxq_max_recv_buf_list_len             32
  rxq_max_packet_size                   16384
  rxq_max_burst_size                    32768
  txq_regular_cpu                       supported
  txq_regular_gpu                       supported
  txq_max_send_buf_list_len             48
  txq_max_lso_header_size               256
  txq_l3_chksum_offload                 supported
  txq_l4_chksum_offload                 supported
  txq_wait_on_time_type                 unsupported
flow_ct
  flow_ct                                unsupported
ipsec
  task_sa_create                        unsupported
  task_sa_destroy                       unsupported
nvrd_transport
  task_write                            supported
  rc_max_src_buf_list_len               0
  dc_max_src_buf_list_len               0
pcc
  pcc                                    unsupported
  pcc_np                                 unsupported
  min_num_threads                       0
  max_num_threads                       0
rdma
  task_send                             supported
  task_send_imm                         supported
  task_read                             supported
<table>
<thead>
<tr>
<th>Feature</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_write</td>
<td>supported</td>
</tr>
<tr>
<td>task_write_imm</td>
<td>supported</td>
</tr>
<tr>
<td>task_atomic_cmp_swap</td>
<td>supported</td>
</tr>
<tr>
<td>task_atomic_fetch_add</td>
<td>supported</td>
</tr>
<tr>
<td>task_receive</td>
<td>supported</td>
</tr>
<tr>
<td>rc_transport_type</td>
<td>supported</td>
</tr>
<tr>
<td>dc_transport_type</td>
<td>unsupported</td>
</tr>
<tr>
<td>rc_task_receive_get_max_dst_buf_list_len</td>
<td>31</td>
</tr>
<tr>
<td>dc_task_receive_get_max_dst_buf_list_len</td>
<td>0</td>
</tr>
<tr>
<td>task_remote_net_sync_event_get</td>
<td>supported</td>
</tr>
<tr>
<td>task_remote_net_sync_event_notify_set</td>
<td>supported</td>
</tr>
<tr>
<td>task_remote_net_sync_event_notify_add</td>
<td>supported</td>
</tr>
<tr>
<td>max_send_queue_size</td>
<td>32768</td>
</tr>
<tr>
<td>max_recv_queue_size</td>
<td>32768</td>
</tr>
<tr>
<td>max_send_buf_list_len</td>
<td>13</td>
</tr>
<tr>
<td>max_message_size</td>
<td>1073741824</td>
</tr>
<tr>
<td>sha</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha256</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha512</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha1_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha256_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>sha512_partial</td>
<td>unsupported</td>
</tr>
<tr>
<td>max_list_num_elem</td>
<td>0</td>
</tr>
<tr>
<td>max_src_buf_size</td>
<td>0</td>
</tr>
<tr>
<td>sha1_min_dst_buf_size</td>
<td>0</td>
</tr>
<tr>
<td>sha256_min_dst_buf_size</td>
<td>0</td>
</tr>
<tr>
<td>sha512_min_dst_buf_size</td>
<td>0</td>
</tr>
<tr>
<td>sha1_partial_hash_block_size</td>
<td>0</td>
</tr>
<tr>
<td>sha256_partial_hash_block_size</td>
<td>0</td>
</tr>
<tr>
<td>sha512_partial_hash_block_size</td>
<td>0</td>
</tr>
</tbody>
</table>
NVIDIA DOCA Comm Channel Admin Tool

This document provides instructions on the usage of the DOCA Comm Channel Admin Tool.

**Introduction**

The Comm Channel Admin Tool is used to print a snapshot of DOCA Comch (comm channel) connections:

- On the BlueField Arm side, it includes DOCA Comch servers and their current connection information

- On the host side, it includes all active client connections and the server they are connected to

- Only client-to-server control channels are reported; fast path producer/consumer channels are not.

**Prerequisites**

The Comm Channel Admin Tool is for Linux only and requires an up-to-date BFB bundle or DOCA host packages of at least 2.7, which include in the Resource dump binary.

**Description and Execution**

The Comm Channel Admin Tool can be executed on the host or Arm CPUs. By default, the tool scans all available PCIe slots to detect supported DOCA devices and reports any Comch information available.

The tool can be run on BlueField Arm or x86 host using the following command:
Sample Output from BlueField Arm

On the BlueField Arm side, any active DOCA Comch servers are be reported:

The following information is available:

- **Server Name** – the name assigned to the server
- **PID** – the Linux process ID of the application which created the server
- **Connections** – the number of connections active on the server out of the total allowed (e.g., 2/512 means 2 active connections of a maximum of 512)
- **PCIe** – the PCIe address of the device which the server has been detected on
- **Interface Name** – the interface name associated with the PCIe address

```
<table>
<thead>
<tr>
<th>Server name</th>
<th>PID</th>
<th>Connections</th>
<th>PCIe</th>
<th>Interface Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>comch1</td>
<td>1898009</td>
<td>2/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch3</td>
<td>1898011</td>
<td>1/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch6</td>
<td>1898014</td>
<td>3/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch2</td>
<td>1898010</td>
<td>1/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch7</td>
<td>1898015</td>
<td>1/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch5</td>
<td>1898013</td>
<td>4/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch8</td>
<td>1898016</td>
<td>2/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
<tr>
<td>comch4</td>
<td>1898012</td>
<td>0/512</td>
<td>0000:03:00:0</td>
<td>p0</td>
</tr>
</tbody>
</table>
```

**Note**
Connections may also be displayed on the BlueField Arm like on x86. This occurs if SF ports are detected here. The interface name associated with the PCIe address indicates the SF port.

Sample Output from x86

The x86 host cannot run DOCA Comch servers. Therefore, individual client connections are reported:

<table>
<thead>
<tr>
<th>Server name</th>
<th>PID</th>
<th>PCIe</th>
<th>Interface Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>comch6</td>
<td>299693</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch3</td>
<td>299688</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch2</td>
<td>299687</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch5</td>
<td>299689</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch5</td>
<td>299692</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch7</td>
<td>299696</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch5</td>
<td>299690</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch6</td>
<td>299694</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch8</td>
<td>299697</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch6</td>
<td>299695</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch8</td>
<td>299698</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch1</td>
<td>299686</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch5</td>
<td>299691</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
<tr>
<td>comch1</td>
<td>299685</td>
<td>0000:3b:00.0</td>
<td>ens1f0np0</td>
</tr>
</tbody>
</table>

The following information is available:
• Server Name – the name of the BlueField Arm server that a client has connected to
• PID – the Linux process ID of the application running a DOCA Comch client
• PCIe – the PCIe address of the BlueField networking platform which the destination server is running on
• Interface Name – the interface name associated with the PCIe address
NVIDIA DPA Tools

Introduction

DPA tools are a set of executables that enable the DPA application developer and the system administrator to manage and monitor DPA resources and to debug DPA applications.

DPA Tools

DPACC Compiler

CLI name: dpacc

DPACC is a high-level compiler for the DPA processor. It compiles code targeted for the DPA processor into an executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the DPA executable. This DPA program can be linked with the host application to generate a host executable where the DPA code is invoked through the FlexIO runtime API.

DPA EU Management Tool

CLI name: dpaeumgmt

This tool allows users to manage the DPA's EUs which are the basic resource of the DPA. The tool enables the resource control of EUs to optimize the usage of computation resources of the DPA. Using this tool, users may query, create, and destroy EU partitions and groups, thus ensuring proper EU allocation between devices.

DPA GDB Server Tool
CLI name: dpa-gdbserver

The DPA GDB Server tool enables debugging FlexIO DEV programs.

**DPA PS Tool**

CLI name: dpa-ps

This tool allows users to monitor running DPA processes and threads.

**DPA Statistic Tool**

CLI name: dpa-statistics

This tool allows users to monitor and obtain statistics on thread execution per running DPA process and thread.

**NVIDIA DOCA DPACC Compiler**

This document describes DOCA DPACC compiler and instructions about DPA toolchain setup and usage.

**Introduction**

DPACC is a high-level compiler for the DPA processor which compiles code targeted for the data-path accelerator (DPA) processor into a device executable and generates a DPA program.

The DPA program is a host library with interfaces encapsulating the device executable. This DPA program is linked with the host application to generate a host executable. The host executable can invoke the DPA code through FlexIO runtime API.
DPACC uses DPA compiler (dpa-clang) to compile code targeted for DPA. dpa-clang is part of the DPA toolchain package which is an LLVM-based cross-compiling bare-metal toolchain. It provides Clang compiler, LLD linker targeting DPA architecture, and other utilities.

### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>DPA as present on the BlueField DPU</td>
</tr>
<tr>
<td>Host</td>
<td>CPU that launches the device code to run on the DPA</td>
</tr>
<tr>
<td>Device function</td>
<td>Any C function that runs on the DPA</td>
</tr>
<tr>
<td>DPA global function</td>
<td>Device function that is the point of entry when offloading any work on DPA</td>
</tr>
<tr>
<td>Host compiler</td>
<td>Compiler used to compile the code targeting the host CPU</td>
</tr>
<tr>
<td>Device compiler</td>
<td>Compiler used to compile code targeting the DPA</td>
</tr>
<tr>
<td>DPA program</td>
<td>Host library that encapsulates the DPA device executable (.elf) and host stubs which are used to access the device executable</td>
</tr>
</tbody>
</table>

### Offloading Work on DPA
To invoke a DPA function from host, the following things are required:

- **DPA device code** – C programs, targeted to run on the DPA. DPA device code may contain one or more entry functions.

- **Host application code** – the corresponding host application. For more information, refer to DPA Subsystem documentation.

- **Runtime** – FlexIO or DOCA DPA library provides the runtime

The generated DPA program, when linked with a host application results in a host executable which also contains the device executable. The host application oversees loading the device executable on the device.

## DPACC Predefined Macros

DPACC predefines the following macros:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DPA</strong></td>
<td>Defined when compiling device code file</td>
</tr>
<tr>
<td><strong>NV_DPA</strong></td>
<td>Defined to the target DPA hardware identifier macros. See Architecture Macros for more details.</td>
</tr>
<tr>
<td><strong>DPA_MAJOR</strong></td>
<td>Defined to the major version number of DPACC</td>
</tr>
<tr>
<td>Macro</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>DPA_MINOR</strong></td>
<td>Defined to the minor version number of DPACC</td>
</tr>
<tr>
<td><strong>DPA_PATCH</strong></td>
<td>Defined to the patch version number of DPACC</td>
</tr>
</tbody>
</table>

Writing DPA Applications

DPA device code is a C code with some restrictions and special definitions.

FlexIO or DOCA-DPA APIs provide interfaces to DPA.

Language Support

The DPA is programmed using a subset of the C11 language standard. The compiler documents any constructs that are not available. Language constructs, where available, retain their standard definitions.

Restrictions on DPA Code

- Use of C thread local storage is not allowed for any variables
- Identifiers with _dpacc prefix are reserved by the compiler. Use of such identifiers may result in an error or undefined behavior
- DPA processor does not have native floating-point support; use of floating point operations is disabled

DPA RPC Functions

A remote procedure call function is a synchronous call that triggers work in DPA and waits for its completion. These functions return a type uint64_t value. They are annotated with a __dpa_rpc__ attribute.
DPA Global Functions

A DPA global function is an event handler device function referenced from the host code. These functions do not return anything. They are annotated with a \_dpa\_global\_ attribute.

For more information, refer to DPA Subsystem documentation.

Characteristics of Annotated Functions

- Global functions must have \texttt{void} return type and RPC functions must have \texttt{uint64\_t} return type
- Annotated functions cannot accept C pointers and arrays as arguments (e.g., \texttt{void my\_global (int *ptr, int arr[])})
- Annotated functions cannot accept a variable number of arguments
- Inline specifier is not allowed on annotated functions

Handling User-defined Data Types

User-defined data types, when used as global function arguments, require special handling. They must be annotated with a \_dpa\_global\_ attribute.

If the user-defined data type is \texttt{typedef}’d, the \texttt{typedef} statement must be annotated with a \_dpa\_global\_ attribute along the data type itself.

Characteristics of Annotated Types

- They must have a copy of the definition in all translation units where they are used as global function arguments
- They cannot have pointers, variable length arrays, and flexible arrays as members
- Fixed-size arrays as C structure members are supported
These characteristics apply recursively to any user-defined/typedef’d types that are members of an annotated type.

DPACC processes all annotated functions along with annotated types and generates host and device interfaces to facilitate the function launch.

**DPA Intrinsics**

DPA features such as fences and processor-specific instructions are exposed via intrinsics by the DPA compiler. All intrinsics defined in the header file `dpaintrin.h` are guarded by the `DPA_INTRIN_VERSION_USED` macro. The current `DPA_INTRIN_VERSION` is 1.3.

Example:

```c
#define DPA_INTRIN_VERSION_USED (DPA_INTRIN_VERSION(1, 3))
#include <dpaintrin.h>
...
__dpa_thread_writeback_window();  // Fence for write barrier
```

For more information, refer to DPA Subsystem documentation.

**Prerequisites**

<table>
<thead>
<tr>
<th>Package</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host compiler</td>
<td>Compiler specified through <code>hostcc</code> option. Both gcc and clang are supported.</td>
</tr>
</tbody>
</table>

**Note**

Minimum supported version for clang as hostcc is clang 3.8.0.
The default device compiler is the "DPA compiler". Installing the DPACC package also installs the DPA compiler binaries `dpa-clang`, `dpa-ar`, `dpa-nm` and `dpa-objdump`.

**Note**

`dpa-clang` is the only supported device compiler.

Available as part of the DOCA software package. DPA toolchain does not provide C library and corresponding headers. Users are expected to use the C library for DPA from the FlexIO SDK.

**Supported Versions**

- DPACC version 1.8.0
- Refer to [DPA Subsystem](#) documentation for other component versions

**Description**

**DPACC Inputs and Outputs**

DPACC can produce DPA programs in a single command by accepting all source files as input. DPACC also offers the flexibility of producing DPA object files or libraries from input files.

DPA object files contain both host stub objects (DPACC-generated interfaces) and device objects. These DPA object files can later be given to DPACC as input to produce the DPA library.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Option Name</th>
<th>Default Output File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compile input device code files to DPA object files</td>
<td>--compile or -c</td>
<td>.dpa.o appended to the name of each input source file</td>
</tr>
<tr>
<td>Compile and link the input device code files/DPA object files, and produce a DPA program</td>
<td>No specific option</td>
<td>No default name, output file name must be specified</td>
</tr>
<tr>
<td>Compile and build DPA library from input device code files/DPA object files</td>
<td>--gen-libs or -gen-libs</td>
<td>No default name, output library name must be specified</td>
</tr>
</tbody>
</table>

DPACC can accept the following file types as input:

<table>
<thead>
<tr>
<th>Input File Extension</th>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.c</td>
<td>C source file</td>
<td>DPA device code</td>
</tr>
<tr>
<td>.dpa.o</td>
<td>DPA object file</td>
<td>Object file generated by DPACC, containing both host and device objects</td>
</tr>
<tr>
<td>.a</td>
<td>DPA object archive</td>
<td>An archive of DPA object files. User can generate this archive from DPACC-generated DPA objects.</td>
</tr>
</tbody>
</table>

Based on the mode of operations, DPACC can generate the following output files:

<table>
<thead>
<tr>
<th>Output File Type</th>
<th>Input Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPA object file</td>
<td>C source files</td>
</tr>
<tr>
<td>DPA program</td>
<td>C source files, DPA object files, and/or DPA object archives</td>
</tr>
<tr>
<td>DPA library</td>
<td>C source files, DPA object files, and/or DPA object archives</td>
</tr>
</tbody>
</table>

The following provides the commands to generate different kinds of supported output file types for each input file type:
<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>DPACC Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>C source file</td>
<td>DPA program</td>
<td>dpacc -hostcc=gcc in.c -o libprog.a</td>
</tr>
<tr>
<td></td>
<td>DPA object</td>
<td>dpacc -hostcc=gcc in.c -c</td>
</tr>
<tr>
<td></td>
<td>DPA library</td>
<td>dpacc -hostcc=gcc in.c -o lib&lt;name&gt; -gen-libs</td>
</tr>
<tr>
<td>DPA object</td>
<td>DPA program</td>
<td>dpacc -hostcc=gcc in.dpa.o -o libprog.a</td>
</tr>
<tr>
<td></td>
<td>DPA library</td>
<td>dpacc -hostcc=gcc in.dpa.o -o lib&lt;name&gt; -gen-libs</td>
</tr>
<tr>
<td>DPA object archive</td>
<td>DPA program</td>
<td>dpacc -hostcc=gcc in.a -o libprog.a</td>
</tr>
<tr>
<td></td>
<td>DPA library</td>
<td>dpacc -hostcc=gcc in.a -o lib&lt;name&gt; -gen-libs</td>
</tr>
</tbody>
</table>

**DPA Program**

DPACC produces a DPA program in compile-and-link mode. A DPA program is a host library which contains:

- DPACC-generated host stubs which facilitate invoking a DPA global function from the host application

- Device executable, generated by DPACC by compiling input DPA device code

DPA program library must be linked with the host application that contains appropriate runtime APIs to load the device executable onto DPA memory.

**DPA Object**

DPACC produces DPA object files in compile-only mode. A DPA object is an object file for the host machine. In a DPA object, the device object generated by compiling the input device code file is placed inside a specific section of the generated host stubs object. This process is repeated for each input file.
DPA Library

A DPA library is a collection of two individual libraries:

- DPA device library – contains device objects generated from input files
- DPA host library – contains host interface objects corresponding to the device objects in DPA device library

The DPA device library is consumed by DPACC during DPA-program generation and the DPA host library can optionally be linked with other host code and be distributed as the host library. Both libraries are generated as static archives.
DPACC Trajectory

The following diagram illustrates DPACC compile-and-link mode trajectory.
**Modes of Operation**

**Compile-and-link Mode**

This is a one-step mode that accepts C source files or DPA object files and produces the DPA program. Specifying the output library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o myLib1.a -hostcc=gcc  # Takes C sources to produce myLib1.a library
```
Compile-only Mode

This mode accepts C source code and produces .dpa.o object files. These files can be given to DPACC to produce the DPA program. The mode is invoked by the --compile or -c option.

The user can explicitly provide the output object file name using the --output-file or -o option.

Example commands:

```
$ dpacc -c input1.c -hostcc=gcc           # Produces input1.dpa.o
$ dpacc -c input3.c input4.c -hostcc=gcc  # Produces input3.dpa.o and input4.dpa.o
$ dpacc -c input2.c -o myObj.dpa.o -hostcc=gcc # Produces myObj.dpa.o
```

Library Generation Mode

This mode accepts C source files or DPA object files and produces the DPA program. Specifying the output DPA library name is mandatory in this mode.

Example commands:

```
$ dpacc in1.c in2.c -o libdummy1 -hostcc=gcc -gen-libs    # Takes C sources to produce libdummy1_host.a and libdummy_device.a archives
$ dpacc in3.dpa.o in4.dpa.o -o libdummy2 -hostcc=gcc -gen-libs # Takes DPA object files to produce libdummy2_host.a and libdummy2_device.a archives
$ dpacc in1.c in3.dpa.o -o outdir/libdummy3 -hostcc=gcc -gen-libs # Takes C source and DPA object to produce outdir/libdummy3_host.a and outdir/libdummy3_device.a archives
```
Execution

To execute DOCA DPACC compiler:

Usage: dpacc <list-of-input-files> -hostcc=<path> [other options]
Helper Flags:
- h, --help Print help information about DPACC
- V, --version Print DPACC version information
- v, --verbose List the compilation commands generated by this invocation while also executing every command in verbose mode
- dryrun, --dryrun Only list the compilation commands generated by DPACC, without executing them
- keep, --keep Keep all intermediate files that are generated during internal compilation steps in the current directory
- keep-dir, --keep-dir Keep all intermediate files that are generated during internal compilation steps in the given directory
- optf, --options-file <file>,... Include command line options from the specified file

Mandatory Arguments

<table>
<thead>
<tr>
<th>Flag</th>
<th>DPACC Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of one or more input files</td>
<td>All</td>
<td>List of C source files or DPA object file names. Specifying at least one input file is mandatory. A file with an unknown extension is treated as a DPA object file.</td>
</tr>
<tr>
<td>-hostcc, --hostcc &lt;path&gt;</td>
<td>All</td>
<td>Specify the host compiler. This is typically the native compiler present on the host system.</td>
</tr>
</tbody>
</table>

Note
The host compiler used to link the host application with the DPA program must be
## Flag | DPACC Mode | Description
---|---|---
-o, --output-file <file> | Compile-and-link/library generation | Specify name and location of the output file.

### Commonly Used Arguments

**Flag**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-app-name, --app-name &lt;name&gt;</td>
<td>Specify DPA application name for the DPA program. This option is required if multiple DPA programs are part of a host application because each DPA application must have a unique name. Default name is __dpa_a_out.</td>
</tr>
<tr>
<td>-mcpu=&lt;target_cpu&gt;</td>
<td>Specify the target DPA hardware for code generation. See <a href="#">DPA Hardware Architectures</a> for more details. Supported values: nv-dpa-bf3, nv-dpa-cx7</td>
</tr>
<tr>
<td>-flto, --flto</td>
<td>Enable link-time optimization (LTO) for device code. Specify this option during compilation along with an optimization level in devicecc-options.</td>
</tr>
<tr>
<td>-devicecc-options, --devicecc-options &lt;options&gt;,...</td>
<td>Specify the list of options to pass to the device compiler.</td>
</tr>
</tbody>
</table>

### Tip

Use `--help` option for a list of all supported options.
<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-devicelink-options, --devicelink-options &lt;options&gt;,...</td>
<td>Specify the list of options to pass during device linking stage.</td>
</tr>
<tr>
<td>-device-libs, --device-libs '-L&lt;path&gt; -l&lt;name&gt;',...</td>
<td>Specify the list of device libraries including their names (in -l) and their paths (in -L). FlexIO libraries are linked by default.</td>
</tr>
<tr>
<td>-I, --common-include-path &lt;path&gt;,...</td>
<td>Specify include search paths common to host and device code compilation. FlexIO headers paths are included by DPACC by default.</td>
</tr>
</tbody>
</table>
| -o, --output-file <file> | Specify name and location of the output file.  
  - Compile-only mode – name of the output DPA object file. If not specified, .dpa.o is generated for each .c file.  
  - Compiler-and-link mode – name of the output DPA program. This is a mandatory option in compiler-and-link mode.  
  - Library generation mode – name of the output library. This is a mandatory option for this mode. Output files <name>_device.a and <name>_host.a are generated. |
| -hostcc-options, --hostcc-options <options>,... | Specify the list of options to pass to the host compiler. |
| -gen-libs, --gen-libs | Generate a DPA library from input files |
| -ldoca_dpa, --ldoca_dpa | Link with DOCA-DPA libraries |

**Note**

Using machine dependent options (e.g., -mcpu, -march, -mabi) through -devicecc-options to influence compiler code generation is not supported.
The `devicecc-options` option allows passing any option to the device compiler. However, passing options that prevent compilation of the input file may lead to unexpected behavior (e.g., `-devicecc-options="-version"` makes the device compiler print the version and not process input files).

**Note**

Incompatible options that affect DPA global function argument sizes during DPACC invocation and host application compilation may lead to undefined behavior during execution (e.g., passing `-hostcc-options="-fshort-enums"` to DPACC and missing this option when building the host application).

**DPA Hardware Architectures**

The following table mentions the DPA architectures, the associated values supported in the compiler through the `-mcpu` option, and the macros defined by the compiler to identify these architectures.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Value</th>
<th>Macro</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectX-7</td>
<td>nv-dpa-cx7</td>
<td>__NV_DPA_CX7</td>
</tr>
<tr>
<td>BlueField-3</td>
<td>nv-dpa-bf3</td>
<td>__NV_DPA_BF3</td>
</tr>
</tbody>
</table>

Since ConnectX-7 and BlueField-3 share the same DPA hardware, `nv-dpa-cx7` is treated as an alias of `nv-dpa-bf3` by the compiler.

**Architecture Macros**
As described in section "DPA Hardware Architectures", the compiler defines identifier macros for each version of DPA hardware. Each identifier macro has a unique integer value which is strictly greater than that of macros for older DPA CPU models. Known aliases such as BlueField-3 DPA and ConnectX-7 DPA share the same integer value. The macro __NV_DPA is defined to the value of current compilation target. This can be used to write device code specific to a DPA hardware generation as shown in the following:

```c
#if __NV_DPA == __NV_DPA_BF3
// Code for Bluefield-3 here
#elif __NV_DPA > __NV_DPA_BF3
// Code for devices after Bluefield-3 here
#endif
```

### Note

The ordering established by the value of the hardware version identifier macros does not imply an ordering of features supported by hardware. It is the user responsibility to ensure that features used in the code which are specific for a DPA version are actually supported on the hardware.

---

### LTO Usage Guidelines

#### Restrictions

- Only the default linker script is supported with LTO
- Using options -fPIC/-fpic/-shared/-mmodel=large through -devicecc-options is not supported when LTO is enabled
- Fat objects containing both LLVM bitcode and ELF representation are not supported
- Thin LTO is not supported
Compatibility

During compilation, LLVM generates the object as bitcode IR (intermediate representation) when LTO is enabled instead of ELF representation. The bitcode IR generated by the DPA compiler is only guaranteed to be compatible within the same version of DPACC. All objects involved in link-time optimization (enabled with \texttt{-flto}) must be built with the same version of DPACC.

Deprecated Features

- The \texttt{-ldpa} option which links with DOCA-DPA libraries is deprecated and will be removed in future releases. Use the option \texttt{-ldoca_dpa} instead.

Examples

This section provides some common use cases of DPACC and showcases the \texttt{dpacc} command.

Building Libraries

This example shows how to build DPA libraries using DPACC. Libraries for DPA typically contain two archives, one for the host and one for the device.

```
dpacc input.c -hostcc=gcc -o lib<name> -gen-libs -hostcc-options="-fPIC"
```

This command generates the output files \texttt{lib<name>_host.a} and \texttt{lib<name>_device.a}.

The host stub archive can be linked with other host code to generate a shared/static host library.

- Generating a static host library:
Generating a shared host library:

```
ar x lib<name>_host.a          # Extract objects to generate *.o
ar cr lib<name>.a <*src.host.o> *.o  # Generate final static archive with all objects
```

**Linking with DPA Device Library**

The DPA device library generated by DPACC using `-gen-libs` as part of a DPA library can be consumed by DPACC using the `-device-libs` option.

```
dpacc input.c -hostcc=gcc -o libInput.a -device-libs="-L <path-to-library> -l<libName>"
```

**Enabling Link-time Optimizations**

Link-time optimizations can be enabled using `-flto` along with an optimization level specified for device compilation.

```
dpacc input1.c -hostcc=gcc -c -flto -devicecc-options="-O2" dpacc input2.c -hostcc=gcc -c -flto -devicecc-options="-O2" dpacc input1.dpa.o input2.dpa.o -hostcc=gcc -o libInput.a
```

**Including Headers**

This example includes headers for device compilation using `devicecc-options` and host compilation using `hostcc-options`. You may also specify headers for any compilation on both the host and device side using the `-I` option.
Generating Output as Source Code

DPACC provides an option, `-src-output`, to generate the output as host source code. This source can be compiled by the host compiler to generate functionally equivalent output which DPACC would have generated directly.

This example shows how to build various outputs of DPACC as source using this option and how to compile the generated source.

DPA-program Source

Generate DPA-program source by passing the following option to DPACC:

```
dpacc input.c -hostcc=gcc -o libfoo.c -src-output
```

Compile the generated source using host compiler to generate an object and build an archive with this object. A macro `__DPACC_SRC_TARGET__` must be defined when building this object to remove code which is unnecessary when building from source:

```
$ gcc libfoo.c -c -I /opt/mellanox/flexio/include -Wno-attributes -Wno-pedantic -Wno-unused-parameter -Wno-return-type -Wno-implicit-function-declaration -D__DPACC_SRC_TARGET__
$ ar cr libfoo.a libfoo.o
```

DPA-library Source

Generate DPA-library source by passing the following option to DPACC:

```
dpacc input.c -hostcc=gcc -o libfoo -gen-libs -src-output
```
This generates the device archive `libfoo_device.a` and host code files `libfoo.lib.c` and `input.dpa.c`.

The host archive of DPA-library is generated by compiling these sources and building an archive. The `__DPACC_SRC_TARGET__` macro must be defined in this instance to remove unnecessary code:

```
$ gcc libfoo.lib.c input.dpa.c -c -I /opt/mellanox/flexio/include -Wno-attributes -Wno-pedantic -Wno-unused-parameter -Wno-return-type -Wno-implicit-function-declaration -D__DPACC_SRC_TARGET__
$ ar cr libfoo_host.a libfoo.lib.o input.dpa.o
```

**DPA-object Source**

Generate DPA-object source by passing the following option to DPACC:

```
dpacc input.c -hostcc=gcc -c -src-output
```

This generates a single file, `input.dpa.c`.

Compile the host file to generate an object:

```
gcc input.dpa.c -c -I /opt/mellanox/flexio/include -Wno-attributes -Wno-pedantic -Wno-unused-parameter -Wno-return-type -Wno-implicit-function-declaration
```

**DPA Compiler Usage**

dpa-clang is a compiler driver for accessing the Clang/LLVM compiler, assembler, and linker which accepts C code files or object files and generates an output according to different usage modes.

**Note**
Invoking the compiler, assembler, or linker directly may lead to unexpected errors.

Refer to the following resources for more detailed information on Clang:

- *Clang Compiler User's Manual*
- *Clang command line argument reference*
- *Target-dependent compilation options*

**Compiler Driver Command-line Options**

```
dpa-clang <list-of-input-files> [other-options]
```

**Linker Command Line Options**

LLD is the default linker provided in the DPA toolchain. Linker-related options are passed to through the compiler driver.

```
dpa-clang -Wl,<linker-option>
```

For more information, please refer to the LLD command line reference.

**dpacc-extract Command Line Options**

dpacc-extract is a tool for extracting a device executable out of a DPA program or a host executable containing DPA program(s).

To execute dpacc-extract:
Usage: dpacc-extract <input-file> -o=<output-file> [other options]

Helper Flags:

-\( o, --output-file \) Specify name of the output file
-\( -app-name, --app-name <name> \) Specify name of the DPA application to extract
-\( -h, --help \) Print help information about dpacc-extract
-\( -V, --version \) Print dpacc-extract version information
-\( -optf, --options-file <file>,... \) Include command line options from the specified file

Mandatory arguments:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input file</td>
<td>DPA program or host executable containing DPA program. Specifying one input file is mandatory.</td>
</tr>
<tr>
<td>-( o, --output-file &lt;file&gt; )</td>
<td>Specify name and location of the output device executable.</td>
</tr>
<tr>
<td>-( -app-name, --app-name &lt;name&gt; )</td>
<td>Specify name of the DPA application to extract. Mandatory if input file has multiple DPA apps.</td>
</tr>
</tbody>
</table>

**Objdump Command Line Options**

The dpa-objdump utility prints the contents of object files and final linked images named on the command line.

For more information, please refer to the [Objdump command line reference](#).

Commonly used dpa-objdump options:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--mcpu=nv-dpa-bf3</td>
<td>Option to choose micro-architecture for DPA processor. ( nv-dpa-bf3 ) is the default CPU for dpa-objdump.</td>
</tr>
</tbody>
</table>

**Archiver Command Line Options**
dpa-ar is a Unix ar-compatible archiver.

For more information, please refer to the Archiver command line reference.

**NM Tool Command Line Options**

The dpa-nm utility lists the names of symbols from object files and archives.

For more information, please refer to the NM tool command line reference.

**Common Compiler Options**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--mcpu=nv-dpa-bf3</td>
<td>Option to choose micro-architecture and ABI for DPA processor. nv-dpa-bf3 is the default CPU for the compiler.</td>
</tr>
<tr>
<td>-mrelax/-mno-relax</td>
<td>Option to enable/disable linker relaxations.</td>
</tr>
<tr>
<td>-I &lt;dir&gt;</td>
<td>Option to include header files present in &lt;dir&gt;.</td>
</tr>
</tbody>
</table>

**Common Linker Options**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Wl,-L &lt;path-to-library&gt; -Wl,-l&lt;library-name&gt;</td>
<td>Option to link against libraries</td>
</tr>
</tbody>
</table>

**Note**

Linker options are provided through the compiler driver dpa-clang.
### Note

The LLD linker script is honored in addition to the default configuration rather than replacing the whole configuration like in GNU ID. Hence, additional options may be required to override some default behaviors.

### Debugging Options

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fdebug-macro</td>
<td>Option to emit macro debugging information. This option enables macro-debugging similar to GCC option -g3.</td>
</tr>
</tbody>
</table>

### Miscellaneous Notes

- Objects produced by LLD are not compatible with those generated by any other linker.

- The default debugging standard of the DPA compiler is DWARFv5. GDB versions <10.1 have issues processing some DWARFv5 features. Use the option -devicecc-options="-gdwarf-4" with DPACC to debug with GDB versions <10.1.

### NVIDIA DOCA DPA Execution Unit Management Tool

This document describes the DPA Execution Unit (EU) management tool, dpaeumgmt.
Introduction

This table introduces important terms for understanding this document:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPA</td>
<td>Data-path accelerator; an auxiliary processor designed to accelerate data-path operations.</td>
</tr>
<tr>
<td>DPA partition</td>
<td>PCIe device function capable of controlling the entire system’s EUs. On NVIDIA® BlueField®-3 it is the ECPF. The DPA partition manager is by default associated with the default partition.</td>
</tr>
<tr>
<td>manager</td>
<td></td>
</tr>
<tr>
<td>EU</td>
<td>Hardware execution unit; a logical DPA processing unit.</td>
</tr>
<tr>
<td>EU group</td>
<td>Collection/subset of EUs which could be created using dpaeumgmt. EU groups are created under an EU partition and could only be formed from the pool of EUs under that partition.</td>
</tr>
<tr>
<td>EU object</td>
<td>EU partition or EU group.</td>
</tr>
<tr>
<td>EU partition</td>
<td>An isolated pool of EUs which may be created using dpaeumgmt. Only when a partition is created and associated with other vHCAs are they able to use hardware resources and execute a DPA software thread.</td>
</tr>
<tr>
<td>EU affinity</td>
<td>The method by which a DPA thread is paired with a DPA EU. DPA supports three types of affinity:</td>
</tr>
<tr>
<td></td>
<td>- none – selects an EU from a pool of all available EUs</td>
</tr>
<tr>
<td></td>
<td>- strict – select only the specified EU (by ID)</td>
</tr>
<tr>
<td></td>
<td>- group – select an EU from all the EUs in the specified group</td>
</tr>
</tbody>
</table>

The DPA EU management tool can run either on the host machine or on the target DPU and allows users to manage the DPA’s EUs which are the basic resource of the DPA. The tool enables the resource control of EUs to optimize computation resources usage of the DPA before using DOCA FlexIO SDK API.

Note

Execution unit partitions will be supported in future releases.
Without EU allocation, a DPA software thread would lack access to the hardware pipeline/CPU time resource, and consequently not be able to execute.

dpaeumgmt serves the following main usages:

- Running a DPA software thread with strict affinity on a DPA EU (i.e., running a DPA thread using only the specific preselected EU). For this purpose, dpaeumgmt provides an option to query the maximum EU ID allowed to use.

- Allowing a DPA software thread to run over a DPA EU from a group of EUs:
  - Once an EU group is created, it is allocated a subset of EUs.
  - dpaeumgmt provides an ID to the created group which can be used to run DPA applications with group affinity where the affinity ID would be the same as that group's ID.

- EU partition management - the ability to manage EU partitions.

When the software stack wishes to run a DPA thread with group affinity type, one of the available EUs from the group's collection is used for the execution.

Note

A DPA thread may execute if and only if there is an available EU for it.

Execution Unit Objects

Upon boot, a default EU partition is automatically created. The default EU partition possesses all the system's EUs. The DPA partition manager function is the only function that belongs to it and can therefore control the entire resources of the system.

When running a DPA thread with none affinity, the EU chosen for the DPA thread to run with comes from the partition's pool of EUs. Namely, from the EUs belonging only to the DPA device's current partition which were not assigned to any EU groups (on the current partition). If the aforementioned group of EUs (i.e., the partition's default EU group) is empty, the DPA thread would fail to run with none affinity.
**dpaemgmt Commands**

dpaemgmt enables users to create, destroy, and query EU objects.

**Note**

dpaemgmt tool must run with root privileges and users must execute `sudo mst start` before using it.

**Top-level dpaemgmt command syntax:**

Usage: `dpaemgmt {help | version | eu_group | partition}`

Type `./dpaemgmt help` for detailed help

**General Commands**

- Print basic usage information for the tool:
  
  `dpaemgmt -h`

- Print a detailed help menu of the tool's commands:
  
  `dpaemgmt help`

- Print version information:
  
  `dpaemgmt version`
Execution Unit Group Commands

The commands listed in the following subsections are used to configure EU groups.

EU Group Command Flags and Arguments

The following table lists the flags relevant to `eu_group` commands. Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Print out basic tool usage information.</td>
</tr>
<tr>
<td>-d</td>
<td>dpa_device</td>
<td>The device interface name (MST/PCI/RDMA/NET).</td>
</tr>
<tr>
<td>-r</td>
<td>range_eus</td>
<td>The range of EUs to allocate an EU group or a partition. The argument must be provided within quotes.</td>
</tr>
<tr>
<td>-g</td>
<td>id_group</td>
<td>Group ID number. This number must be positive and less than or equal to the <code>max_num_dpa_eu_group</code> parameter which may be retrieved using the command <code>eu_group info -d &lt;device&gt;</code>.</td>
</tr>
<tr>
<td>-n</td>
<td>name_group</td>
<td>Group name; 15-character string. The argument must be provided within quotes.</td>
</tr>
<tr>
<td>-f</td>
<td>file_groups</td>
<td>Full path or only the filename if it is located in the same directory as the executable directory (where <code>dpaeumgmt</code> is).</td>
</tr>
</tbody>
</table>

Info EU Group

Print information on the relevant DPA resources for the EU groups:
Example:

```
$ sudo ./dpaeumgmt eu_group info -d mlx5_0
Max number of DPA EU groups: 15
Max number of DPA EUs in one DPA EU group: 190
Max DPA EU number available to use: 190
Max EU group name length is 15 chars
```

Create EU Group

Create an EU group with the specified name on the provided device's partition. The EUs indicated by the range are taken from the DPA device's EU partition.

```
dpaeumgmt eu_group create --dpa_device <device> --name_group <name> --range_eus <range>
```

Example:

```
$ sudo ./dpaeumgmt eu_group create -d mlx5_0 -n "HG hello world1" -r "6-8,16,55,70"
Group created successfully-
EU group ID: 1
EU group name: HG hello world
Member EUs are: 6,7,8,16,55,70
```

Note

After successfully creating an EU group, users can run a DPA thread using `group` affinity with the affinity type set to the group's ID.
**Destroy EU Group**

Destroy an EU group that exists on the device's partition with either the provided group name or ID.

```
dpaeumgmt eu_group destroy --dpa_device <device> [--name_group <name> | --id_group <id>]
```

Example:

```
$ sudo ./dpaeumgmt eu_group destroy -d mlx5_0 -g 1
Group with group id: 1, was destroyed successfully
```

**Query EU Group**

Query EU groups residing on the provided device's partition. If one of the optional parameters is used, the command only queries the specific group and prints it if it exists:

```
dpaeumgmt eu_group query --dpa_device <device> [--name_group <name> | --id_group <id>]
```

Example:

```
$ sudo ./dpaeumgmt eu_group query -d mlx5_0
1) EU group ID: 1
   EU group name: HG hello world
   Member EUs are: 6,7,8,16,55,70

   In total there are 1 EU groups configured.
```

More options:
Apply EU Group

Apply the EU groups provided in the file on the device's partition:

```
dpaemgmt eu_group apply --dpa_device <device> --file_groups <file>
```

File format example:

```json
{
   "eu_groups": [
       { "name": "hg1", "range": "178-180"},
       { "name": "hg2", "range": "2-10"}
   ]
}
```

⚠️ **Note**

The command removes all the previous EU groups defined on the EU partition that the DPA device belongs to and applies the ones from the file.

Example:

```
$ sudo ./dpaeumgmt eu_group apply -d mlx5_0 --file_groups example.json
1) EU group ID: 1
   EU group name: hg1
   Member EUs are: 178,179,180
```
1) EU group ID: 2
EU group name: hg2
Member EUs are: 2,3,4,5,6,7,8,9,10

In total there are 2 EU groups configured.

**EU Partition Commands**

The commands listed in the following subsections are used to configure EU partitions.

**EU Partition Command Flags and Arguments**

The following table lists the flags relevant to EU `partition` commands. Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Print out basic tool usage information.</td>
</tr>
<tr>
<td>-d</td>
<td>--dpa_device</td>
<td>The device interface name (MST/PCI/RDMA/NET).</td>
</tr>
<tr>
<td>-r</td>
<td>--range_eus</td>
<td>The range of EUs to allocate an EU group or a partition. The argument must be provided within quotes.</td>
</tr>
<tr>
<td>-p</td>
<td>--id_partition</td>
<td>Partition ID number. This number must be positive and less than or equal to the value of <code>max_num_dpa_eu_partition</code> which may be retrieved using the command <code>partition info -d &lt;device&gt;</code>.</td>
</tr>
<tr>
<td>-v</td>
<td>--vhca_list</td>
<td>The vHCA IDs to be associated with the partition. The argument must be provided within quotes.</td>
</tr>
<tr>
<td>-m</td>
<td>--max_num_eu_group</td>
<td>The number of EU groups to reserve for the partition upon its creation.</td>
</tr>
</tbody>
</table>
Info EU Partition

Print the relevant DPA resources of the EU partitions:

```
dpaeumgmt partition info --dpa_device <device>
```

Example:

```
$ sudo ./dpaeumgmt partition info -d mlx5_0
Max number of DPA EU partitions: 15
Max number of VHCAs associated with a single partition: 32
Max number of DPA EU groups: 15
Note- an allocation of a partition consumes from the number of DPA EU *groups* available to create
Max DPA EU number available to use: 190
```

Create EU Partition

Create an EU partition on the DPA device:

```
dpaeumgmt partition create --dpa_device <device> --vhca_list <id_list> --range_eus <range> --max_num_eu_group <max_num>
```

Example:

```
$ sudo ./dpaeumgmt partition create -d mlx5_0 -v 1 -r 10-20 -m 2
Partition created successfully-
EU Partition ID: 1
Maximal number of groups: 2
The partition has a total of 1 associated VHCA IDs, namely: 1
Partition's member EUs are: 10,11,12,13,14,15,16,17,18,19,20
```

Destroy EU Partition
Destroy an EU partition that exists on the device’s partition:

```bash
dpaemgmt partition destroy --dpa_device <device> --id_partition <id>
```

Example:

```bash
$ sudo ./dpaeumgmt partition destroy -d mlx5_0 -p 1
Partition with partition id: 1, was destroyed successfully
```

**Query EU Partition**

Query EU partitions that reside on the provided device’s partition and print out the partition if it exists:

```bash
dpaemgmt partition query --dpa_device <device> [--id_partition <id>]
```

Example:

```bash
$ sudo ./dpaeumgmt partition query -d mlx5_0 -p 1
EU Partition ID: 1
Maximal number of groups: 2
The partition has a total of 1 associated VHCA IDs, namely: 1
Partition's member EUs are: 10,11,12,13,14,15,16,17,18,19,20
```

More options:

```bash
$ sudo ./dpaeumgmt partition query -d mlx5_0
```
vHCAs and Partitions

The following diagram illustrates the ownership and control of a partition by a vHCA and also which vHCAs have claim to (i.e., can use) a partition.

Known Limitations

- Currently, dpaeumgmt is only supported on the DPU not the host
- dpaeumgmt should run before creating a DPA process so all resources are configured ahead of time
  - Running the tool over a device with an existing DPA process results in failure
- The EU group name assigned by the user must be unique for every EU group on a specific partition or the EU group create command fails
- The creation of an EU partition consumes from the number of EU groups allowed on the vHCA's partition it is created on:
  - 1 group for the partition itself due to a default group created for each partition
<max_num> of groups which is the user’s input provided upon partition creation

- Creating groups or running DPA threads in general (with any affinity) on interfaces other than ECPF, requires a configuration of a valid partition for the specific vhCA

- Only the default partition is exposed to the real EU numbers, all other partitions the user creates use virtual EUs
  
  - For example, if a user creates a partition with the range of EUs 20-40, querying the partition info from one of its virtual HCAs (vHCAs) would display EUs from 0-20. Therefore, the EU whose real number is 39 in this example would correspond to the virtual EU number 19.

- Group IDs on a non-default partition are virtual.
  
  - Different partitions can have completely distinct groups, even if they have the same ID.
  
  - The affinity ID parameter, specified on the FlexIO API, can distinguish between the groups according to the vhCA an application is running on.

- vhCA ID overlap is not allowed on EU partitions

- It is not possible to query vhCA IDs with dpaeumgmt, these are assumed to be known by the user beforehand

- Partition destruction fails if there are EU objects that exist on that partition

- It is not possible to know which EU has been chosen to run on

- Every vhCA sees the partition it belongs to, and its resources, as the entire world. It only sees:
  
  - Groups and partitions it created
  
  - The number of EUs it was given
  
  - The max_num_eu_group of the partition it belongs to

- No guarantee regarding EU group ID that will be given on group creation

- The default groups (of every partition) cannot be managed by the user
• The EU numbers available are between 0 and the max DPA EU number available to use minus 1 (the upper limit can be queried using the info command specified above)

• dpaeumgmt does not support virtual functions (VFs)

• It is not possible to create partitions on other vHCAs other than the DPA partition manager function

• There are at most 16 hardware EU group entities

**NVIDIA DOCA DPA GDB Server Tool**

This document describes the DPA GDB Server tool.

![Info]

The DPA GDB Server Tool is currently supported at beta level.

**Introduction**

The DPA GDB Server tool (dpa-gdbserver) enables debugging FlexIO DEV programs.

DEV programs for debugging are selected using a token (8-byte value) provided by the FlexIO process owner.

![Info]

Any GDB, familiar with RISC-V architecture, can be used for the debug. Refer to [this](#) page for information how to work with GDB.
**Glossary**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUD</td>
<td>Process under debug. DEV-side processes intended for debug.</td>
</tr>
<tr>
<td>EU</td>
<td>Execution unit (similar to hardware CPU core)</td>
</tr>
<tr>
<td>DPA</td>
<td>Data path accelerator</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote process communication. Mechanism used in FlexIO to run DEV-side code instantly. Runtime is limited to 6 seconds.</td>
</tr>
<tr>
<td>HOST</td>
<td>x86 or aarch64 Linux OS which manages dev-side code (i.e., DEV)</td>
</tr>
<tr>
<td>DEV</td>
<td>RISC-V code, loaded by HOST into the DPA's device. Triggered to run by different types of interrupts. DEV side is directly connected to ConnectX adapter card.</td>
</tr>
<tr>
<td>GDB</td>
<td>GNU Project debugger. Allows users to monitor another program while it executes.</td>
</tr>
<tr>
<td>GDBSERVER</td>
<td>Tool for remote debug programs</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-time operation system running on RISC-V core. Manages handling of interrupts and calls to DEV user processes routines.</td>
</tr>
<tr>
<td>RSP</td>
<td>Remote serial protocol. Used for interaction between GDB and GDBSERVER.</td>
</tr>
</tbody>
</table>

**Known Limitations**

- DPA GDB technology does not catch fatal errors. Therefore, if a fatal error occurs, core dump (created by `flexio_coredump_create()`) should be used.

- DPA GDB technology does not support Outbox access. GDB users cannot write to Doorbell or to Window configuration areas.

- DPA GDB technology does not support Window access. Read/write to Window memory does not work properly.
DPA-specific Notes

Token

The process under debug (PUD) can expose a debugging token. Every external process, using this token, get full access to the process with given token. To not show it constantly (e.g., for security reasons), users can modify their host application temporary. See flexio_process_udbg_token_get().

Connection on Application Launch

If the code which needs debugging begins to run immediately after launch, the user should modify the host application to stop upon start to give the user time to run dpa-gdbserver. One possible way of doing this is to place function getchar() immediately after process creation.

Dummy Thread Concept

Something to consider with DPA debugging is that a PUD does not have a running thread all time (e.g., the process's thread may exist but be waiting for incoming packets). In a regular Linux application, this scenario is not possible and GDB does not support such cases.

Therefore, when no thread is running, dpa-gdbserver reports a dummy thread:

```
(gdb) info thread
Id  Target Id            Frame
   * 1   Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ?? ()
(gdb)
```

In this case user can inspect memory, create breakpoints, and give the continue command.

Commands like step, next, and stei cannot be executed for the Dummy thread.
Watchdog Issues

The RTOS has a watchdog timer that limits DEV code interrupt processes to 120 seconds. This timer is stopped when the user connects to DEV with GDB. Therefore users will have no time limitation for debugging.

Tool TCP Port and Execution Unit (EU)

By default, dpa-gdbserver uses TCP port 1981 and runs on EU 29. If this conflicts with another application (or if other instances of dpa-gdbserver are running), users should change the defaults as follows:

```
$> dpa-gdbserver mlx5_0 -T <token> -s <port> -E <eu_id>
```

Debugging

Preparation for Debug

Modify your FlexIO application if needed. Make sure the HOST code prints udbg_token and waits for GDB connection if needed:

```c
+  uint64_t udbg_token;
    flexio_process_create(..., &flexio_process);
+  udbg_token = flexio_process_udbg_token_get(flexio_process);
+  if (udbg_token)
+      printf("Process created. Use token >>> %#lx <<< for debug\n", udbg_token);
+  printf("Stop point for waiting of GDB connection. Press Enter to continue..."); /* Usually you don't need this stop point */
+  fflush(stdout);
```
Extract the DPA application from the FlexIO application. For example:

```bash
$> dpacc-extract cc-host/app/host/flexio_app_name -o flexio_app_name.rv5
```

## Start Debugging

1. Run your FlexIO application. It should expose the debug token:

```bash
$> flexio_app_name mlx5_0
Process created. Use token >>> 0xd6278388ce4e682c <<< for debug
```

2. Run `dpa-gdbserver` with the debug token received:

```bash
$> dpa-gdbserver mlx5_0 -T 0xd6278388ce4e682c
Registered on device mlx5_0
Listening for GDB connection on port 1981
```

3. Run any GDB with RISC-V support. For example, `gdb-multiarch`:

```bash
$> gdb-multiarch -q flexio_app_name.rv5
Reading symbols from flexio_app_name.rv5...
(gdb)
```

4. Connect to the gdbserver using proper TCP port and hostname, if needed:

```bash
(gdb) target remote :1981
Remote debugging using :1981
```
DPA-specific Debugging Techniques

Easy Example of Transitioning from Dummy to Real Thread

Transitioning between the dummy thread and a real thread is not standard practice for debugging under GDB. In an ideal situation, the user would know exactly the entry points for all their routines and can set breakpoints for all of them. Then the user may run the `continue` command:

```
(gdb) target remote :1981
Remote debugging using :1981
0x0800000000000000 in ?? ()
(gdb) info threads
   Id  Target Id                                Frame
   * 1  Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ?? ()
(gdb) b foo
Breakpoint 1 at 0x400000b2: file ../tests/path/hello.c, line 58.
(gdb) b bar
Breakpoint 2 at 0x40000518: file ../tests/path/hallo.c, line 113.
(gdb) continue
Continuing.
```

Initiate interrupts for your DEV program (depends your task), and GDB should catch a breakpoint and now the real thread of the PUD appear instead of the dummy:

```
(gdb) continue
Continuing.
(gdb) [New Thread 1.2]
[New Thread 1.130]
[New Thread 1.258]
[New Thread 1.386]
[Switching to Thread 1.2]

Thread 2 hit Breakpoint 1, foo(thread_arg=9008)
```
From this point, you may examine memory and trace your code as usual.

**Complicated Example of Transitioning from Dummy to Real Thread**

In a more complicated situation, the interrupt happens after GDB connection. In this case, the real thread should start running but cannot because the PUD is in HALT state. The user can type the command `info threads`, see new thread instead of the old dummy, and then switch to the new thread manually:

```
(gdb) target remote :1981
Remote debugging using :1981
0x0800000000000000 in ?? ()
(gdb) info threads
   Id  Target Id                                       Frame
   * 1  Thread 1.805378433 (Dummy Flexio thread) 0x0800000000000000 in ?? ()
(gdb) info threads
[New Thread 1.32769]
   Id  Target Id                                       Frame
   2  Thread 1.32769 (Process 0 thread 0x8000 GVMI 0) bar (arg=0xc0, len=0)
       at /path/lib/src/stub.c:167

The current thread <Thread ID 1> has terminated. See `help thread'.
(gdb) thread 2
[Switching to thread 2 (Thread 1.32769)]
#0  bar (arg=0xc0, len=0)
    at /path/lib/src/stub.c:167
167    {
(gdb) bt
#0  bar (arg=0xc0, len=0)
```
The user must switch to the new thread manually (see line 14). After this, they can trace/debug the flow as usual (i.e., using the commands step, next, stept).

### Finishing Real Thread without Finishing PUD

Every interrupt handler at some point finishes its way and returns the CPU resources to RTOS. The most common way to do this is to call function `flexio_dev_thread_reschedule()`. The command `next` on this function will have the same effect as the command `continue`.

```c
__dpa_thread_fence(__DPA_MEMORY, __DPA_W, __DPA_W);
(gdb) next
flexio_dev_cq_arm(dtctx, app_ctx.rq_cq_ctx.cq_idx, app_ctx.rq_cq_ctx.cq_number);
(gdb) next
if ((dev_errno = flexio_dev_get_and_rst_errno(dtctx))) {
  print_sim_str("Nothing to do. Wait for next duar\n", 0);
  flexio_dev_thread_reschedule();
  (gdb) next
```
Info

GDB waits until the user types `^C` or a breakpoint is reached after the next interrupt occurred.

Error Reporting

Info

The DPA GDB server tool has been validated with `gdb-multiarch` (version 9.2) and with GDB version 12.1 from RISC-V tool chain.

Note

The GDB server should support all commands described in GDB RSP (remote serial protocol) for GDB stubs. But only the most common GDB commands are supported.

Should a dpa-gdbserver bug occur, please provide the following data:

- Used GDB (name and version)
- Commands sequence to reproduce the issue
- DPA GDB server tool console output
- DPA GDB server tool log directory content (see next part for details)
• Optional – output data printed when dpa-gdbserver is run in verbose mode

**Tool Log Directory**

For every run, a temporary directory is created with the template `/tmp/flexio_gdbs.XXXXXX`.

To locate the latest one, run the following command:

```
$> ls -ldtr /tmp/flexio_gdbs.* | tail
```

**Verbosity Level of gdbserver**

By default, dpa-gdbserver does not print any log information to screen. Adding `-v` option to command line increases verbosity level, printing additional info to dpa-gdbserver terminal display. Verbosity level is incremented according to number of 'v' in command line switch (i.e. -w, -ww etc.).

One `-v` shows the RSP exchange. This is a textual protocol, so users can read and understand requests from GDB and answers from the GDB server:

```
<<<< "qTStatus"
>>>> ""
<<<< "?"
>>>> "S05"
<<<< "qfThreadInfo"
>>>> "mp01.30011981"
<<<< "qsThreadInfo"
>>>> "l"
<<<< "qAttached:1"
>>>> "1"
<<<< "Hc-1"
>>>> "OK"
<<<< "qC"
>>>> "QCp01.30011981"
```
Info

In the examples, `<<<<<<` and `>>>>>>` are used to indicate data received from GDB and transmitted to GDB, respectively.

When running with a higher verbosity level (e.g., run `dpa-gdbserver` with option `-v` or higher), the exchange with the RTOS module is shown:

```
<<<<<< "qfThreadInfo"
/ 2/dgdb_module - cmd 0x5
/ 2/dgdb_module - retval 0x4
>>>>>> "mp01.30011981"
<<<<<< "qsThreadInfo"
/ 2/dgdb_module - cmd 0x5
/ 2/dgdb_module - retval 0x5
>>>>>> "l"
<<<<<< "m800000000000000,4"
/ 2/dgdb_module - cmd 0xc
/ 2/dgdb_module - retval 0x9
>>>>>> "E0a"
<<<<<< "m7fffffffffffffff,4"
/ 2/dgdb_module - cmd 0xc
/ 2/dgdb_module - retval 0x9
>>>>>> "E0a"
<<<<<< "qSymbol::"
>>>>>> "OK"
```

Info

Lines beginning with `/ #/` provide the number of internal RTOS threads printed from the DEV side.
Useful Info Regarding Work with GDB

This section provides useful information about commands and methods which can help users when performing DPA debug. This is not related to the dpa-gdbserver itself. But this is about remote debugging and FlexIO sources.

Command "directory"

GDB can run on a different host from the one where compilation was done. For example, users may have compiled and run their application on host1 and run their instance of GDB on host2. In this case, users will see the error message ../xxx/yyy/zzz/your_file.c: No such file or directory. To solve this problem, copy sources to the host running GDB (host2 in the example). Make sure to save the original code hierarchy. Use GDB command directory to inform where the sources are to GDB:

```
host2~$> gdb-multiarch -q /tmp/my_riscv.elf
Reading symbols from /tmp/my_riscv.elf...
(gdb) b foo
Breakpoint 1 at 0x4000016c: file ../xxx/yyy/zzz/my_file.c, line 182.
(gdb) target remote host1:1981
Remote debugging using host1:1981
0x0800000000000000 in ?? ()
(gdb) c
Continuing.
[New Thread 1.32769]
[Switching to Thread 1.32769]
Thread 2 hit Breakpoint 1, foo (thread_arg=5728) at ../xxx/yyy/zzz/my_file.c:182
182  ../xxx/yyy/zzz/my_file.c: No such file or directory.
(gdb) directory /tmp/apps/
Source directories searched: /tmp/apps:$cdir:$cwd
(gdb) list
179  struct flexio_dev_thread_ctx *dtctx;
180  uint64_t dev_errno;
181
182  print_sim_str("=====> NET event handler started\n", 0);
183
184  flexio_dev_print("Hello GDB user\n");
185
```
Note

Pay attention to the exact path reported by GDB. The argument for the command directory should point to the start point for this path. For example, if GDB looks for ..///xxx/yyy/zzz and you placed the sources in local directory /tmp/copy_of_worktree, then the command should be (gdb) directory /tmp/copy_of_worktree/// and not (gdb) directory /tmp/copy_of_worktree/.

Sometimes, the *.elf file provides a global path from the root. In this case, use the command set substitute-path <from> <to>. For example, if the file /foo/bar/baz.c was moved to /mnt/cross/baz.c, then the command (gdb) set substitute-path /foo/bar /mnt/cross instructs GDB to replace /foo/bar with /mnt/cross, which allows GDB to find the file baz.c even though it was moved.

See this page of GDB documentation for more examples of specifying source directories.

Core Dump Usage

If the code runs into a fatal error even though the host side of your project is implemented correctly, a core dump is saved which allows analyzing the core. It should point exactly to where the fatal error occurred. The command backtrace can be used to examine the memory and its registers. Change the frame to see local variables of every function on the backtrace list:

```bash
$> gdb-multiarch -q -c crash_demo.558184.core /tmp/my_riscv.elf
Reading symbols from /tmp/my_riscv.elf...

[New LWP 1]
#0 0x0000000004000126e in read_test (line=153, ptr=0x30) at /xxx/yyy/zzz/my_file.c:109
109   val = *(volatile uint64_t *)ptr;
(gdb) bt
#0 0x0000000004000126e in read_test (line=153, ptr=0x30) at /xxx/yyy/zzz/my_file.c:109
#1 0x0000000004000031a in tlb_miss_test (op_code=1) at /xxx/yyy/zzz/my_file.c:153
```
Debug of Optimized Code

Usually highly optimized code is compiled and run.

Two types of mistakes in code can be considered:

- Logical errors
- Optimization-related errors

Logical errors (e.g., using & instead of &&) are reproduced on the non-optimized version of the code. Optimization related errors (e.g., forgetting volatile classification, non-usage of memory barriers) only impact optimization. Non-optimized code is much easier for tracing with GDB, because every C instruction is translated directly to assembly code.

It is good practice to check if an issue can be reproduced on non-optimized code. That helps observing the application flow:

```sh
$> build.sh -O 0
```
For tracing this code, using GDB commands `next` and `step` should be sufficient.

But if an issue can only be reproduced on optimized code, you should start debugging it. This would require reading disassembly code and using the GDB command `stepi` because it becomes a challenge to understand exactly which C-code line executed.

**Disassembly of Advanced RISC-V Commands**

DPA core runs on a RISC-V CPU with an extended instruction set. The GDB may not be familiar with some of those instructions. Therefore, `asm` view mode shows numbers instead of disassembly. In this case it is recommended to disassemble your RISC-V binary code manually. Use the `dpa-objdump` utility with the additional option `--mcpu=nv-dpa-bf3`.

```bash
$> dpa-objdump -sSdxl --mcpu=nv-dpa-bf3 my_riscv.elf > my_riscv.asm
```

The following screenshot shows the difference:

```
4000057a: 03 35 84 fe     ld  a0,  -24(s0)  
4000057e: 08 65           ld  a0,  8(a0)     
40000580: 1355856b         ld  ra,  24(sp)    
40000584: e2 60           ld  s0,  16(sp)    
40000586: 05 61           addi  sp, sp, 32   
40000588: 02 80           ret                    
```

**NVIDIA DOCA DPA PS Tool**

**Introduction**

DOCA `dpa-ps` is a CLI tool which allows users to monitor running DPA processes and threads. The tool presents sorted lists of the currently running DPA processes and threads.
The process ID output of the `dpa-ps` tool may be used as the input parameter for the `dpa-statistics` tool.

Info

This tool is supported for NVIDIA® BlueField®-3 only.

Command Flags and Arguments

The following table lists the flags for the `dpa-ps` tool:

<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Help information</td>
</tr>
<tr>
<td>-d</td>
<td>--device</td>
<td>Device interface name (MST/RDMA)</td>
</tr>
<tr>
<td>-p</td>
<td>--process-id</td>
<td>Hexadecimal process ID for filtering</td>
</tr>
<tr>
<td>-t</td>
<td>--threads</td>
<td>Show threads info for each process</td>
</tr>
<tr>
<td>-i</td>
<td>--suppress-header-info</td>
<td>Suppress print header info</td>
</tr>
</tbody>
</table>

Info

Arguments for the flags must be used within quotes (if more than one) and without extra spaces.

Example

```
$ sudo ./dpa-ps -d mlx5_0 -t
```
Known Limitations

- The dpa-ps and dpa-statistics tools cannot be run at the same time on the same device

NVIDIA DOCA DPA Statistics Tool

Introduction

DOCA dpa-statistics is a CLI tool which allows users to monitor and obtain statistics on thread execution per running DPA process and thread. The tool is used to expose information about the running DPA processes and threads and to collect statistics on DPA thread performance.

The tool presents performance information for running DPA threads, including the number of cycles and instructions executed in a time period. The tool enables initiating and stopping collection of statistics and displaying the data collected per thread.
Collecting Performance Statistics Data

The command `collect` works on four mutually exclusive modes:

- Enable mode – start collecting performance data
- Disable mode – stop collecting performance data
- Timeout mode – start collecting, wait with a timeout, stop collect and print info. User could break the wait with Ctrl-C command and then the timeout will be canceled and tool will disable statistics collection and prints the info with the actual time of the collect operation.
- Infinite mode – no special flags. Same as timeout mode but with infinite timeout. The tool awaits the Ctrl-C command to stop.

The following table lists the `collect` command's flags and arguments:

<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Help information</td>
</tr>
<tr>
<td>-d</td>
<td>--device</td>
<td>Device interface name (MST/RDMA)</td>
</tr>
<tr>
<td>-p</td>
<td>--process-id</td>
<td>Hexadecimal process ID for filtering</td>
</tr>
</tbody>
</table>

The process ID output of the `dpa-ps` tool may be used as the input parameter for the `dpa-statistics` tool.

Info

This tool is supported for NVIDIA® BlueField®-3 only.
<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-i</td>
<td>--suppress-header-info</td>
<td>Suppress print header info</td>
</tr>
<tr>
<td>-n</td>
<td>--enable</td>
<td>Enable collect info</td>
</tr>
<tr>
<td>-o</td>
<td>--disable</td>
<td>Disable collect info</td>
</tr>
</tbody>
</table>

| -t           | --timeout   | Enable collect, wait with timeout, disable collect and print info |

**Info**

Timeout value is in milliseconds.

Examples for inputting timeout value:

- 45 – 45 milliseconds
- 45.55 – 45 milliseconds and 550,000 nanoseconds
- .0005 – 500 nanoseconds
- 45m55n – 45 milliseconds and 55 nanoseconds
- 66n – 66 nanoseconds

| -r           | --reset     | Reset counters before operation starting collect operation |

**Presenting Statistics List**

Presenting performance statistics is applicable after initiating data collection.
The following table lists the `show` command's flags and arguments:

<table>
<thead>
<tr>
<th>Short Option</th>
<th>Long Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
<td>Help information</td>
</tr>
<tr>
<td>-d</td>
<td>--device</td>
<td>Device interface name (MST/RDMA)</td>
</tr>
<tr>
<td>-p</td>
<td>--process-id</td>
<td>Hexadecimal process ID for filtering</td>
</tr>
<tr>
<td>-i</td>
<td>--suppress-header-info</td>
<td>Suppress print header info</td>
</tr>
</tbody>
</table>

Output example:

```
$ sudo ./dpa-statistics show -d mlx5_0 -p 1
ProcessID
ThreadId Cycles Instruction Time Executions
1         3       266268     18193    164    41
4         4       411571     32727    252    47
```

Where:

- **ProcessID** – The `dpa_process_object_id` to which the threads belongs
- **ThreadId** – DPA thread object ID
- **Cycles** – Total EU cycles the thread used
- **Instruction** – Total number of instructions the thread executed
- **Time** – Total time in ticks the thread was active
- **Executions** – Total number of thread invocations

**Examples**

- Example of `collect` in infinite mode for process 0 with suppress header info:

```
$ sudo ./dpa-statistics collect -d mlx5_0 -p 0 -i
```
• Example of `collect` in timeout mode with a timeout of 1 second and half a millisecond.

```bash
$ sudo ./dpa-statistics collect -d mlx5_0 -t 1000.500
```

Data collected for 1000 milliseconds 500000 nanoseconds

<table>
<thead>
<tr>
<th>ProcessID</th>
<th>ThreadID</th>
<th>Cycles</th>
<th>Instruction</th>
<th>Time</th>
<th>Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>223964</td>
<td>13754</td>
<td>140</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>190130</td>
<td>13754</td>
<td>114</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>266268</td>
<td>18193</td>
<td>164</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>411571</td>
<td>32727</td>
<td>252</td>
<td>47</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>223205</td>
<td>13754</td>
<td>137</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>189896</td>
<td>13754</td>
<td>113</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>191796</td>
<td>13754</td>
<td>117</td>
<td>31</td>
</tr>
</tbody>
</table>

• Example of enabling statistics collection with reset of counters.

```bash
$ sudo ./dpa-statistics collect -d mlx5_0 -n -r
```

• Example of disabling statistics collection.

```bash
$ sudo ./dpa-statistics collect -d mlx5_0 -o
```
Known Limitations

- Reading large statistics counter blocks takes a long time
- The dpa-ps and dpa-statistics tools cannot be run at the same time on the same device
NVIDIA DOCA PCC Counter Tool

This document provides instruction on the usage of the PCC Counter tool.

Introduction

The PCC Counter tool is used to print PCC-related hardware counters. The output counters help debug the PCC user algorithm embedded in the DOCA PCC application.

Prerequisites

DOCA 2.2.0 and higher.

Description

If NVIDIA® BlueField®-3 is operating in DPU mode, the script must be executed on the Arm side. If BlueField-3 is operating in NIC mode, the script must be executed on the host side.

Info

Refer to NVIDIA BlueField Modes of Operation for more information on the DPU’s modes of operation.

The following performance counters are supported for PCC:

- MAD_RTT_PERF_CONT_REQ – the number of RTT requests received in total
- MAD_RTT_PERF_CONT_RES – the number of RTT responses received in total
• SX_EVENT_WRED_DROP – the number of TX events dropped due to the CC event queue being full

• SX_RTT_EVENT_WRED_DROP – the number of "TX event with RTT request sent indication" dropped due to the CC event queue being full

• ACK_EVENT_WRED_DROP – the number of Ack events dropped due to the CC event queue being full

• NACK_EVENT_WRED_DROP – the number of Nack events dropped due to the CC event queue being full

• CNP_EVENT_WRED_DROP – the number of CNP events dropped due to the CC event queue being full

• RTT_EVENT_WRED_DROP – the number of RTT events dropped due to the CC event queue being full

• HANDLED_SXW_EVENTS – the number of handled CC events related to SXW

• HANDLED_RXT_EVENTS – the number of handled CC events related to RXT

• DROP_RTT_PORT0_REQ – the number of RTT requests dropped in total from port 0

• DROP_RTT_PORT1_REQ – the number of RTT requests dropped in total from port 1

• DROP_RTT_PORT0_RES – the number of RTT responses dropped in total from port 0

• DROP_RTT_PORT1_RES – the number of RTT responses dropped in total from port 1

• RTT_GEN_PORT0_REQ – the number of RTT requests sent in total from port 0

• RTT_GEN_PORT1_REQ – the number of RTT requests sent in total from port 1

• RTT_GEN_PORT0_RES – the number of RTT responses sent in total from port 0

• RTT_GEN_PORT1_RES – the number of RTT responses sent in total from port 1

• PCC_CNP_COUNT – the number of CNP received in total, regardless of whether it is handled or ignored

Execution
To use the PCC Counter:

1. Initialize all supported hardware counters. Run:

   ```
   sudo ./pcc_counters.sh set /dev/mst/mt41692_pciconf0
   ```

   **Info**
   
   Counters are zeroed after each `set` command.

2. Query all supported hardware counters. Run:

   ```
   sudo ./pcc_counters.sh query /dev/mst/mt41692_pciconf0
   ```

   **Info**
   
   The output counters are counted from the time the `set` command is executed to the time when the `query` command is issued.

Example output:

   ```
   sudo ./pcc_counters.sh query /dev/mst/mt41692_pciconf0
   -----------------PCC Counters-----------------
   Counter: MAD_RTT_PERF_CONT_REQ  Value: 000000000028b85b
   Counter: MAD_RTT_PERF_CONT_RES  Value: 000000000028b85a
   Counter: SX_EVENT_WRED_DROP     Value: 0000000000000000
   Counter: SX_RTT_EVENT_WRED_DROP Value: 0000000000000000
   Counter: ACK_EVENT_WRED_DROP    Value: 0000000000ccdf4f
   ```
<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACK_EVENT_WRED_DROP</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>CNP_EVENT_WRED_DROP</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>RTT_EVENT_WRED_DROP</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>HANDLED_SXW_EVENTS</td>
<td>0000000000932543a</td>
</tr>
<tr>
<td>HANDLED_RXT_EVENTS</td>
<td>000000000028b85c</td>
</tr>
<tr>
<td>DROP_RTT_PORT0_REQ</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>DROP_RTT_PORT1_REQ</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>DROP_RTT_PORT0_RES</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>DROP_RTT_PORT1_RES</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>RTT_GEN_PORT0_REQ</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>RTT_GEN_PORT1_REQ</td>
<td>00000000028b85c</td>
</tr>
<tr>
<td>RTT_GEN_PORT0_RES</td>
<td>0000000000000000</td>
</tr>
<tr>
<td>RTT_GEN_PORT1_RES</td>
<td>00000000028b85d</td>
</tr>
<tr>
<td>PCC_CNP_COUNT</td>
<td>0000000000000000</td>
</tr>
</tbody>
</table>
NVIDIA DOCA Socket Relay

This document describes DOCA Socket Relay architecture, usage, etc.

Introduction

DOCA Socket Relay allows Unix Domain Socket (AF_UNIX family) server applications to be offloaded to the DPU while communication between the two sides is proxied by DOCA Comch.

Socket relay only supports SOCK_STREAM communication with a limit of 512 AF_UNIX application clients.

The tool is coupled to the client AF_UNIX server application. That is, a socket relay instance should be initiated per AF_UNIX server application.
Socket relay is transparent to the application except for the following TCP flows:

- Connection termination must be done by the host side application only

- Once a FIN packet (shutdown system call has been made) is sent by the host side application, data cannot be transferred between the DPU and the host, and the connection must be closed.

The following details the communication flow between the client and server:

- The AF_UNIX client application connects to the socket relay AF_UNIX server in the same way as in the original flow
• The AF_UNIX client application sends SOCK_STREAM packets

• The socket relay (host) AF_UNIX server receives the client application packets, and the Comm Channel client sends them on the channel

• The socket relay (DPU) Comm Channel server receives the client application packets and the AF_UNIX client sends them to the user's AF_UNIX server application

**Prerequisites**

Windows 10 build 17063 is the minimal Windows version to run DOCA Socket Relay on a Windows host.

**Dependencies**

NVIDIA® BlueField®-2 firmware version 24.35.1012 or higher.

**Execution**

To execute DOCA Socket Relay:

```plaintext
Usage: doca_socket_relay [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:
- s, --socket               Unix domain socket path, host side will bind to and DPU connect to
- n, --cc-name               Comm Channel service name
- p, --pci-addr              DOCA Comm Channel device PCI address
- r, --rep-pci               DOCA Comm Channel device representor PCI address (needed only on DPU)
```

For example (DPU side):
To run `doca_socket_relay` using a JSON file:

```bash
doca_socket_relay --json [json_file]
```

For example:

```bash
doca_socket_relay --json /tmp/doca_socket_relay.json
```

### Arg Parser DOCA Flags

Refer to the [DOCA Arg Parser](#) for more information.

<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>General flags</td>
<td>h</td>
<td>help</td>
<td>Prints a help synopsis</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>version</td>
<td>Prints program version information</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>log-level</td>
<td>Set the log level for the application:</td>
<td>&quot;log-level&quot;: 60</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>
<tr>
<td>Flag Type</td>
<td>Short Flag</td>
<td>Long Flag/ JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>---------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>N/A</td>
<td>sdk-log-level</td>
<td></td>
<td>SDK log events are currently unsupported for this tool</td>
<td>N/A</td>
</tr>
<tr>
<td>j</td>
<td>json</td>
<td></td>
<td>Parse all command flags from an input JSON file</td>
<td>N/A</td>
</tr>
<tr>
<td>Prog flags</td>
<td>s</td>
<td>socket</td>
<td>AF_UNIX (SOCK_STREAM) path. On the host, this is the path of the socket relay AF_UNIX server for the client's application to connect to. On the DPU, this is the path of the client AF_UNIX server application. Note: This flag is mandatory.</td>
<td>&quot;socket&quot;: &quot;/tmp/uds-server.socket&quot;</td>
</tr>
<tr>
<td></td>
<td>n</td>
<td>cc-name</td>
<td>Comm Channel service name Note: This flag is mandatory.</td>
<td>&quot;cc-name&quot;: sr_channel</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>pci-addr</td>
<td>DOCA Comm Channel device PCIe address Note: This flag is mandatory.</td>
<td>&quot;pci-addr&quot;: b1:00.1</td>
</tr>
<tr>
<td></td>
<td>r</td>
<td>rep-pci</td>
<td>DOCA Comm Channel device representor PCIe address</td>
<td>&quot;rep-pci&quot;:</td>
</tr>
<tr>
<td>Flag Type</td>
<td>Short Flag</td>
<td>Long Flag/JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>--------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note</strong></td>
<td><strong>b1:02.2</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This flag is available and mandatory only on the DPU.</td>
<td></td>
</tr>
</tbody>
</table>