NVIDIA DOCA Bench
Table of contents

DOCA Bench Sample Invocations
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 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:

<table>
<thead>
<tr>
<th>Aggregate stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration:</td>
</tr>
<tr>
<td>3000049 micro seconds</td>
</tr>
<tr>
<td>Enqueued jobs:</td>
</tr>
<tr>
<td>17135128</td>
</tr>
<tr>
<td>Dequeued jobs:</td>
</tr>
<tr>
<td>17135128</td>
</tr>
<tr>
<td>Throughput:</td>
</tr>
<tr>
<td>5712042 Operations/s</td>
</tr>
<tr>
<td>Ingress rate:</td>
</tr>
<tr>
<td>063.832 Gib/s</td>
</tr>
<tr>
<td>Egress rate:</td>
</tr>
<tr>
<td>063.832 Gib/s</td>
</tr>
</tbody>
</table>

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.

```
[25000ns -> 25999ns]: 0
[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
[35000ns -> 35999ns]: 0
```

**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

....
min:  1878 ns
max:  4956 ns
median:  2134 ns
mean:  2145 ns
90th %ile:  2243 ns
95th %ile:  2285 ns
99th %ile:  2465 ns
```
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.

Info
Currently, DOCA supports running only one pipeline at a time.

---

**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.

---

**Info**

When executed, DOCA Bench reports a full list of all parameters and configured values.

---

**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>doca_compress::compress</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::decompress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>doca_dma</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></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::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:

In this particular setup, the BlueField executes "DOCA Bench" while the host (x86) is 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 should be done over a secure channel when using the standard TCP/IP sockets.
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.

Info

Due to batching it is possible that more than the supplied jobs are executed.
Command-line Parameters

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.
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:

- 01234501234501234501
- 23450123450123450123
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:

```
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:

```
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 line 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 summery 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:
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:

CPU Usage stats
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.

--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.

---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**

---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**
--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

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

⚠️ **Warning**
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-outputBuffers. 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.

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

Requires the companion app to be configured.
--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`

Info
---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 supplied address (to launch the companion). Refer to your OS’s documentation for information on how to achieve this.
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:

```bash
--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:

```bash
--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:
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.

**Note**


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`
- `doption.mmo.log_qp_depth`
- `doption.mmo.log_num_qps`
- `doca_rdma.transport-type`

- `doca_rdma.gid-index`

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:

```
--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
```
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.

Info

To make the samples clearer, certain verbose output and repeated information has been removed or shortened, in particular to output of the configuration or defaults when DOCA Bench is first executed is removed.

Info

The command line options may need to be updated to suit your environment (e.g., TCP addresses, port numbers, interface names, usernames). See the "Command-line Parameters" section for more information.

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**

```bash
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  
   doption.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: [
```

NVIDIA DOCA Bench 46
Attributes: [doca_eth.l4-chksum-offload: false, doca_eth.max-burst-size: 512, doption.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
EAL: Detected shared linkage of DPDK
EAL: Multi-process socket /run/user/48679/dpdk/rte/mp_socket
EAL: Selected IOVA mode 'PA'
EAL: VFIO support initialized
TELEMERTY: No legacy callbacks, legacy socket not created
EAL: Probe PCI driver: mlx5_pci (15b3:a2d6) device: 0000:b1:00.1 (socket 2)
[08:19:32:110524][398304][DOCA][WRN][engine_model.c:90][adapt_queue_depth] adapting queue depth to 128.
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:   3000633 micro seconds
  Enqueued jobs:  611215
  Dequeued jobs:  611215
  Throughput: 000.204 MOperations/s
  Ingress rate: 002.276 Gib/s
  Egress rate: 002.276 Gib/s

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
dooption.companion_app.path=/opt/mellanox/doca/tools/doca_bench_companion \
  --companion-core-list 6 \
  --job-output-buffer-size 1500
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, doption.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
    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...
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: 3000049 micro seconds
    Enqueued jobs: 17135128
    Dequeued jobs: 17135128
    Throughput: 005.712 MOperations/s
    Ingress rate: 063.832 Gib/s
    Egress rate: 063.832 Gib/s
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
[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
  Throughput: 000.467 MOperations/s
  Ingress rate: 000.265 Gib/s
  Egress rate: 000.223 Gib/s
Worker thread[2](core: 14) stats:
  Duration: 10733 micro seconds
  Enqueued jobs: 5000
  Dequeued jobs: 5000
  Throughput: 000.466 MOperations/s
  Ingress rate: 000.264 Gib/s
  Egress rate: 000.222 Gib/s
Worker thread[3](core: 15) stats:
  Duration: 10788 micro seconds
  Enqueued jobs: 5000
  Dequeued jobs: 5000
  Throughput: 000.463 MOperations/s
  Ingress rate: 000.262 Gib/s
  Egress rate: 000.221 Gib/s
Aggregate stats
  Duration: 10788 micro seconds
  Enqueued jobs: 20000
  Dequeued jobs: 20000
  Throughput: 001.854 MOperations/s
  Ingress rate: 001.050 Gib/s
  Egress rate: 000.884 Gib/s
  min: 1878 ns
  max: 4956 ns
  median: 2134 ns
  mean: 2145 ns
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 \ 
```
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:
  :
  :
  :
  :
  :
  :
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

**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
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
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:
**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:

   ```bash
   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:

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

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: 001.146 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
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_count,[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_count,[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

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
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 
   --csv-stats "stats.*"

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
WT[1] 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
Data path thread [2] started...
WT[2] 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)
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,4.5
28814502,28814502,59012100096,59012100096,087.932 Gib/s,087.932 Gib/s,5.762683 MOperations/s,5.7
28903009,28903009,59193362432,59193362432,088.203 Gib/s,088.203 Gib/s,5.780495 MOperations/s,5.7
```

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`. 

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
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**

```bash
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 \
    --run-limit-seconds 5 \
    --csv-output-file /tmp/dma_sweep_job_size.csv \
    --enable-environment-information
```

**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
  Dequeued jobs: 23645128
  Throughput: 004.729 MOperations/s
  Ingress rate: 036.079 Gib/s
  Egress rate: 036.079 Gib/s
Preparing permutation 2 of 2...
Executing permutation 2 of 2...
Data path thread [0] started...
WT[0] Executing 100 warm-up tasks using 100 unique tasks
Cleanup permutation 2 of 2...
[main] Completed! tearing down...
Aggregate stats
  Duration: 5000027 micro seconds
  Enqueued jobs: 22963128
  Dequeued jobs: 22963128
  Throughput: 004.593 MOperations/s
  Ingress rate: 070.078 Gib/s
  Egress rate: 070.078 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 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
  Throughput: 006.440 MOperations/s
  Ingress rate: 098.272 Gib/s
  Egress rate: 098.272 Gib/s

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
doxa_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 `doca_ec` 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 `doca_ec` 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

```
./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
```

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**

```bash
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

```
  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 \n           --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

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.