DOCA Bench Sample Invocations

Table of contents

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

```
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_comp
\
                                    --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,
```

```
doption.companion_app.path:/opt/mellanox/doca/tools/doca_bench_comp
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
TELEMETRY: 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
doption.companion_app.path=/opt/mellanox/doca/tools/doca_bench_comp
\
                                        --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_comp
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
```

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
        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
        90th %ile:      2243 ns
        95th %ile:      2285 ns
        99th %ile:      2465 ns
        99.9th %ile:    3193 ns
        99.99th %ile:   4487 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 \
            --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
```

```
[35000ns -> 35999ns]: 0
[36000ns -> 36999ns]: 0
[37000ns -> 37999ns]: 0
[38000ns -> 38999ns]: 128
[39000ns -> 39999ns]: 0
[40000ns -> 40999ns]: 0
[41000ns -> 41999ns]: 0
[42000ns -> 42999ns]: 0
[43000ns -> 43999ns]: 128
[44000ns -> 44999ns]: 128
[45000ns -> 45999ns]: 0
.. OUTPUT RETRACTED (SHORTENED) ..
[>110000ns]: 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

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
        Egress rate:   000.295 Gib/s
Stats for thread[1](core: 1)
        Duration:      3000056 micro seconds
        Enqueued jobs: 3757335
        Dequeued jobs: 3757335
        Throughput:    001.252 MOperations/s
        Ingress rate:  019.110 Gib/s
        Egress rate:   000.299 Gib/s
Aggregate stats
        Duration:      3000064 micro seconds
        Enqueued jobs: 7471270
        Dequeued jobs: 7471270
        Throughput:    002.490 MOperations/s
        Ingress rate:  038.000 Gib/s
        Egress rate:   000.594 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:

```
cfg.companion.connection_string,cfg.pipeline.steps,cfg.pipeline.use
un_limit.jobs,cfg.run_limit.bytes,cfg.data_provider.type,cfg.data_p
ute.doca_compress.algorithm,cfg.attribute.doca_ec.matrix_type,cfg.a
fg.attribute.doca_eth.l4_chksum_offload,cfg.attribute.doca_sha.algo
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,s
ts.output.throughput.bytes,stats.input.throughput.rate,stats.output
,[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,,,,,,,7471
tions/s
```

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 \
                --csv-output-file /tmp/sha_512_test.csv \
                --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:   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
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,s
,[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,,,,,,,7246
```

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
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,s
,[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,,,,,,,7246
,[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,,,,,,,7261
```

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
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
        Ingress rate:  037.867 Gib/s
        Egress rate:   000.370 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)
        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_coun
22999128,22999128,47102214144,47102214144,070.185 Gib/s,070.185 Gib/s,4.599650 MO
28814502,28814502,59012100096,59012100096,087.932 Gib/s,087.932 Gib/s,5.762683 MO
28903009,28903009,59193362432,59193362432,088.203 Gib/s,088.203 Gib/s,5.780495 MO
```

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 \
                --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.

```
cfg.companion.connection_string,cfg.pipeline.steps,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,s
,[doca_dma],0,0,10000,1000,5,,,random-data,2048,d8:00.0,,,100,"[0, 1, 2, 3, 4,
5, 6,
7]",throughput,0,,,,,,,,,1024,1,1,1024,128,1 fragments,,,,,,,23645128,2364
internal-24.04-
0.4.8,N/A,x63,N/A,N/A,N/A,0000:af:00.0,Infiniband,N/A,Ethernet,2,true,N/
<none>,N/A,Gen15,x63,Gen15,N/A,true,x63,N/A,x86_64,104857600000,N/A,000
174-generic,N/A,x63,true,true,20.04.1 LTS (Focal
Fossa),Ubuntu,N/A,2,true,N/A
,[doca_dma],0,0,10000,1000,5,,,random-data,2048,d8:00.0,,,100,"[0, 1, 2, 3, 4,
5, 6,
7]",throughput,0,,,,,,,,,2048,1,1,1024,128,1 fragments,,,,,,,22963128,2296
internal-24.04-
0.4.8,N/A,x63,N/A,N/A,N/A,0000:af:00.0,Infiniband,N/A,Ethernet,2,true,N/
<none>,N/A,Gen15,x63,Gen15,N/A,true,x63,N/A,x86_64,104857600000,N/A,000
174-generic,N/A,x63,true,true,20.04.1 LTS (Focal
Fossa),Ubuntu,N/A,2,true,N/A
```

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

```
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 `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

```
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 \
            --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.

order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
