NVIDIA DOCA RXPBench Performance Comparison Tool

User Guide

MLNX-15-060475 _v1.3 | June 2022

Table of Contents

List of Tables

Chapter 1. Introduction

RXPBench is a tool that allows for the performance comparison between the NVIDIA® RXP® hardware RegEx acceleration engine found in the NVIDIA® BlueField® DPU and the Intel® Hyperscan software library. It provides a comprehensive set of options and can facilitate ingress of data from live network ports or previously recorded PCAP files.

It is designed to provide a real-world comparison of these technologies, and present results customers could expect to receive after implementing either technology in their products.

1.1. Document Scope

This document provides the following information for RXPBench:

- Example use case
- Breakdown of analysis and runtime statistics
- Options and configuration settings available

1.2. Document Glossary

The terms listed in the following table are used in this document.

Table 1. Terms and Definitions

Term	Definition
Job	A unit of data for the RXP to scan. A job can be a packet, packet header, packet payload, packet header and payload, or a block of user-defined data.
Job Directory	A directory with custom files that contain data to test that matches returned are as expected (validation)
RegEx	A common abbreviation for regular expression.
Regular expression	A regular expression is a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of

Term	Definition
	characters. A common abbreviation for this is "Regex".
ROF file	The compiled Regex rules as object code, produced by the RXP compiler, and programmed into the RXP engine.
Ruleset	A list of regular expressions and strings that can be compiled into object code by the RXP Compiler and executed on the RXP.
RXP	High-speed, hardware-accelerated regular expression engine
RXPC	The external compiler application that translates regular expressions into compiled object code (ROF file)

The acronyms listed in the following table are used in this document.

Table 2. Acronyms

Acronym	Definition
HS	Intel® Hyperscan Software Library
PCRE	Perl Compatible Regular Expressions
RE	Regular Expression
ROF	RXP Object Format (currently at version 2)
RXP	Regular eXpression Processor
RXPC	Regular eXpression Processor Compiler

1.3. Icons

The following icons are used within this document:

- 

– the configuration option is related to a physical DPDK port

- 

– the configuration option is related to a PCAP format file
- 

– the configuration option is related to a standard text file
- 

– the configuration option is related to a job format directory
- 

– this setting is only available when using DOCA (see `--regex-dev` option)

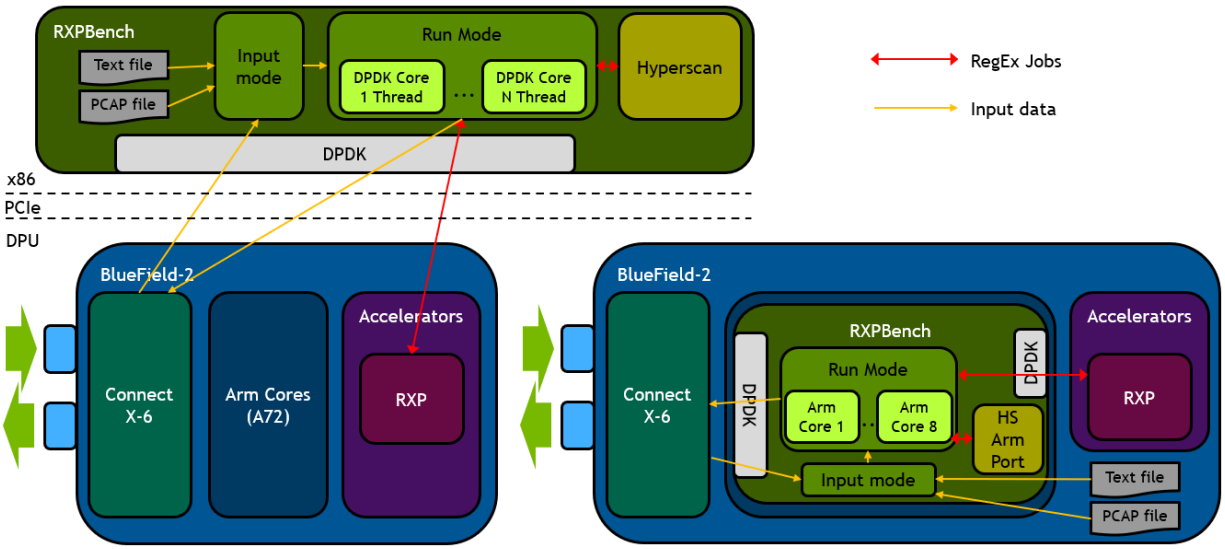
Chapter 2. RXPBench Overview and Installation

Whilst the primary focus of this tool is to provide accurate real-world performance comparisons between the Intel® Hyperscan software library (HS) and the BlueField-2 RXP hardware acceleration engine has additional functionality. This functionality includes:

- Execution on both Intel host and the BlueField-2 DPU Arm cores
- Multicore support
- Ingress of traffic from live DPDK network ports, or PCAP files
- Can act as a "bump in the wire"
- Ability to accept RXP, Hyperscan, and generic rules files
- Asynchronous operations, similar to end-user applications
- Comprehensive configuration through a configuration file or command line options
- A high-performance reference application for DPDK RegEx operations

RXPBench utilizes either the high-speed DOCA (`doca_regex`) framework, or DPDK (`dpdk_regex`) to provide hardware accelerated regular expression offloading using the RXP engine. Software-based RegEx evaluation is provided through the standard Hyperscan library.

The following is an overview of the RXPBench architecture:



This diagram shows the relationship between RXPBench and the underlying BlueField-2 hardware when the application is being run on the x86 host (left side) or on the BlueField-2 Arm cores (right).

At the core of the application is a packet processing engine designed to acquire packets from a network or file source. These packets are then processed through DPDK threads and offloaded to the RXP hardware accelerator using the high performance DOCA or DPDK libraries for pattern matching (or via the Hyperscan library).

2.1. Host Installation

RXPBench is installed as part of the standard DOCA installation process via the `doca-tools` package.

Follow the instructions in the NVIDIA DOCA Installation Guide for instructions on how to install DOCA if you have not done so yet.

2.1.1. Prerequisites

Prior to execution of the RXPBench, an installation of Hyperscan must be present on the host. Hyperscan can be obtained from your Linux distributions package manager (APT, dpkg, yum, etc.) or alternatively compiled from the source. Depending on your installation the following version of Hyperscan is required:

Linux Distribution	Hyperscan Version	Installation Command
Ubuntu 18.04	4	`apt install libhyperscan4`
Ubuntu 20.04	5	`apt install libhyperscan5`
CentOS 7.x	–	Hyperscan is provided through 3rd party vendors. The following command will install Hyperscan 5.3.0 on CentOS 7:

Linux Distribution	Hyperscan Version	Installation Command
		`yum install epel-release sudo yum install http://repo.openfusion.net/centos7-x86_64/hyperscan-5.3.0-1.of.el7.x86_64.rpm`
CentOS 8.x	–	Hyperscan is provided through 3rd party vendors. The following command will install Hyperscan 5.3.0 on CentOS 8: `yum install epel-release sudo yum install https://download-ib01.fedoraproject.org/pub/epel/8/Everything/x86_64/Packages/h/hyperscan-5.3.0-5.el8.x86_64.rpm`

2.2. DPU Installation

The RXPBench utility is provided as part of the DOCA framework and is therefore installed by default with the BFB.

2.3. DOCA and DPDK

RXPBench provides support for both the DOCA framework and the DPDK.

While most functionalities are supported in both frameworks, DOCA provides additional features to further enhance RegEx pattern matching. The icon DOCA is used to indicate a DOCA only feature.

For information on selecting between DOCA and DPDK, refer to the `--Regex-dev` option.

Chapter 3. Example Application Usage

This section details an example use case of the RXPBench application, providing in depth explanations of the processes and statistics produced.

This example will focus on the execution of RXPBench using a simple text file containing the works of William Shakespeare (shakespeare.txt), using rules in Hyperscan format (henry.hs) whilst executing on the BlueField-2 RXP engine.

3.1. Configuring RXPBench

RXPBench supports the configuration of options through a "configuration" file, or through the command line. In practice if a configure file is used, the command line options are still available and will override any options already present in the configuration file.

By default, RXPBench will always search for a "rxpbench.conf", this allows common set-up commands to be removed from the command line. Commonly the DPDK EAL (-D) options are placed in this file as they rarely change after being initially set.

For a full list of options, see section General Configuration Options.

In this example we are providing all the options on the command-line; there is no configuration file.

The command-line required to execute our example (simple text file containing the works of William Shakespeare (shakespeare.txt), using rules in Hyperscan format (henry.hs) whilst executing on the BlueField-2 RXP engine) is as follows:

```
./rxpbench -D "-l 0,1,2,3 -n 1 -a 5e:00.0,class=regex –file-prefix=rxpbench -a
 5e:00.01" --input-mode text_file -f ../Shakespeare.txt -d rxp -R ../henry.hs -l
 2048 -n 10000 -c 1
```

The "-D" option provides the DPDK EAL options, contained within a set of quotation marks ("). These options are passed directly to DPDK during the initialization of the application and are in general specific to your host. "

The first RXPBench option is the "--input-mode" which states that RXPBench will pull data from a "text_file", the "-f" option then specifies the location and name of the text file to be searched.

The "-d" option states the mode in which RXPBench will operate, available options are "rxp" or "hs" and in this instance we are requesting that the BlueField-2 hardware accelerator is used.

The "-R" option provides the tool with a set of uncompiled rules, in this case they are presented in Hyperscan format. RXPBench supports the use of rules formats that are different

from the selected device/algorithm. For example, the RXP can accept Hyperscan rules and the Hyperscan library can received RXP formatted rules. The conversion process within RXPBench is automatic.

The "-l" option supplies the size of the data block sent to the device/algorithm. In this instance a buffer of 2KB is received and pattern matched from the text file.

The number of iterations is controlled by the "-n" option; due to the high performance of the BlueField-2 RXP engine the input file must be iterated 10,000 times to provide enough input data to ensure are run-time of a few seconds.

The final option is the core count (-c), this defines how many CPU cores the tool can use. In this instance we are using a single core.

3.2. Regular Expressions

RXPBench accept regular expressions in two different formats:

- Uncompiled – The regular expressions are presented in a text file which follows with the RXP rules file format, or the Hyperscan file format.
- Compiled – In the case for the RXP, rules are externally compiled using the RXP Compiler (rxpc) and presented to RXPBench as a ROF file.

If uncompiled rule files are used, RXPBench can cross compile the rules regardless of the file format or device selected, i.e. A Hyperscan format rules file will be converted for use by the RXP engine, whilst an RXP format rules file will be converted for use by the Hyperscan engine.

RXPBench presents information during its start-up that indicates the progress of compilation of uncompiled rules, as well as the success of programming those compiled rules to the device:

```
**********************************************************************
WARNING:        Compiling rule file with default params.
                Better performance may be achieved by
                compiling separately with taylored inputs.
**********************************************************************


----------------------------------------------------------------------------
RXP Compiler - a utility to compile regular expressions for the RXP
Version 20.08.2 Build df58611
Copyright (C) 2020 Mellanox Technologies Ltd. All rights reserved.
----------------------------------------------------------------------------
Info: Processing started: Tue Mar  9 12:27:01 2021
Info: Setting target RXP hardware version to v5.7...done
Info: Setting virtual prefix mode to 0...done
Info: Setting RXP prefix capacity to 32K...done
Info: Setting compiler objective value to 5...done
Info: Setting number of threads for compilation to 1...done
Info: Reading ruleset...done
Info: Detected 1 rules
Info: Setting maximum TPE data width to 4...done
Info: Scanning rules...[==============================]...done
Info: Enabling specialized string mode...done
Info: Mapping prefixes, phase 1...[==============================]...done
```

If any errors or warnings are detected during the compilation process, RXPBench provides detailed information on the problematic regular expression. For example:

```
*****************************************************************
ALERT: Compiling rule file with default params.
Better performance may be achieved by compiling separately with tailored inputs.
*****************************************************************

Error:
     :subset_id:1
     :rule_id:  2
     :error_code:12
     :Rule exceeded maximum PTPB threshold of 0.0001. To include this rule, increase the threshold greater than 0.00434783 and try again.
/g.*e.*t/i

<< ERROR: Regex rules compilation error. >>
EAL: Error - exiting with code: 1
  Cause: Regex dev rule compilation error
```

3.3. Runtime Statistics

During the execution of RXPBench a series of run-time statistics are presented by the utility. This provides detailed information on the current process:

```
+-------------------------------------------------------------------------------+
|                            SPLIT PER QUEUE STATS                              |
+-------------------------------------------------------------------------------+

+-------------------------------------+
|              CORE 00                |
+-------------------------------------+
| Recv Bytes:              6345585739 |
| Regex Bytes:             6345585739 |
| Recv Bufs:                  3098837 |
| Regex Bufs:                 3098837 |
| Matches:                     708180 |
+-------------------------------------+

+-------------------------------------+
|               TOTAL                 |
+-------------------------------------+
| Recv Bytes:              6345585739 |
| Regex Bytes:             6345585739 |
| Recv Bufs:                  3098837 |
| Regex Bufs:                 3098837 |
| Matches:                     708180 |
|                                     |
| Duration:                    1.0000 |
| Regex Perf (total):         50.7645 |
| Regex Perf (split):         50.7645 |
+-------------------------------------+
```

For each core in use by the tool, the following statistic are presented:

- Received Bytes – These are bytes received from the input source
- Regex Bytes – These are the bytes transmitted to the Regex engine; this value can be less than the received byte count if certain confirmation options are used (such as payload thresholds or if "app layer" payloads are only being scanned)

- Recv Bufs – The total buffers received from the input source; in this case due to the "-l 2048" option each buffer contains 2048 bytes.
- Regex Bugfs – The number of buffers transmitted to the Regex device.
- Matches – The total number of Regex matches seen in the input data.

In addition to each core statistic, a running total output is provided, including aggregated values for the above fields and a duration field it also provides:

- Regex Perf (total) – The performance total in Gigabits per second (Gb/s) for the entire duration of the run.
- Regex Perf (split) – The performance total in Gigabits per second (Gb/s) for the past update period.

3.4. End-of-Run Statistics

When the execution is completed, or aborted using Ctrl + C, several statistics blocks are output to the console. This allows users to verify and understand the execution of the performance test.

3.4.1. Configuration Statistics Block

This section of statistics provides an overview of the RXPBench configuration, most of this information is simply the mirroring of configuration files.

```
+-------------------------------------------------------------------------------+
|                               CONFIGURATION                                   |
+-------------------------------------------------------------------------------+
|                                                                               |
| - RUN SETTINGS -                                                              |
|                                                                               |
| INPUT MODE:          DPDK Regex                                               |
| REGEX DEV:           PCAP File                                                |
| INPUT FILE:          ../defcon_100k.pcap                                      |
| RULES INPUT:         ../waf_rules_bf2.rof2                                    |
|                                                                               |
|                                                                               |
| - DPDK LIVE CONFIG -                                                          |
|                                                                               |
| DPDK PRIMARY PORT:   -                    DPDK SECOND  PORT:   -              |
| APP LAYER MODE:      False                                                    |
|                                                                               |
|                                                                               |
| - RUN/SEARCH PARAMS -                                                         |
|                                                                               |
| INPUT DURATION:      0                    BUFFER LENGTH:       0             |
| INPUT PACKETS:       0                    BUFFER THRESHOLD:    0             |
| INPUT BYTES:         0                    BUFFER OVERLAP:      0             |
| INPUT ITERATIONS:    10000                GROUP/BATCH SIZE:    64            |
|                                                                               |
|                                                                               |
| - PRELOADED DATA INFO -                                                       |
|                                                                               |
| DATA LENGTH:         58170495                                                 |
| APP LAYER MODE:      True                                                     |
| VALID PACKETS:       56710                VLAN/QNQ:            0             |
| INVALID LENGTH:      0                    IPV4:                56710         |
| UNSUPPORTED PROT:    5772                 IPV6:                0             |
| NO PAYLOAD:          37518                TCP:                 50114         |
| THRESHOLD DROP:      0                    UDP:                 6596          |
|                                                                               |
|                                                                               |
| - REGEX DEVICE CONFIG -                                                       |
|                                                                               |
| RXP MAX MATCHES:     254                  HS MODE:             -             |
| RXP MAX PREFIXES:    N/A                  HS SINGLE MATCH:     -             |
| RXP MAX LATENCY:     N/A                  HS LEFTMOST MATCH:   -             |
|                                                                               |
|                                                                               |
| - PERFORMANCE CONFIG -                                                        |
|                                                                               |
| NUMBER OF CORES:     1                                                        |
|                                                                               |
+-------------------------------------------------------------------------------+
```

The "Preloaded Data Info" section details any preloading of data, when using PCAP or text files, that has occurred during the initialization of the application:

- Data Length – When the input is file based (PCAP or Text) this is the total data that is preloaded/cached to reduce I/O operations

- App Layer Mode – Whether the application is effectively scanning the application layer (TCP/UDP frames) and ignore the headers (Ethernet, MAC, etc.) prior to the application layer.
- Valid Packets – If app layer mode is enabled, these are packets that contain a valid payload
- Invalid Length – This value is incremented if a PCAP packet is found to be unexpectedly truncated
- Unsupported Prot – If app layer mode is enabled, the packet did not contain one of the required protocols (VLAN/IPv4/IPv6/TCP or UDP)

3.4.2. Run Overview Block

This section provides an overview of the RXPBench execution; it provides the core statistics which allow you to gauge the performance of the algorithm using the supplied rules and input data.

```
+-------------------------------------------------------------------------------+
|                                RUN OVERVIEW                                   |
+-------------------------------------------------------------------------------+
|                                                                               |
| - RAW DATA PROCESSING -                                                       |
|                                                                               |
| TOTAL PKTS:     59411914              QNQ:           0                        |
| TOTAL BYTES     60942897892           VLAN:          0                        |
| VALID PKTS:     0                     IPV4:          0                        |
| UNSUPPORTED:    0                     IPV6:          0                        |
| NO PAYLOAD:     0                     TCP:           0                        |
| UNDER THRES:    0                     UDP:           0                        |
|                                                                               |
| PACKET PROCESSING RATE (Mpps):    1.9701                                      |
| PACKET PROCESSING PERF (Gb/s):    16.1667                                     |
|                                                                               |
|                                                                               |
| - REGEX PROCESSING -                                                          |
|                                                                               |
| TOTAL REGEX BUFFERS:              59411914                                    |
| TOTAL REGEX BYTES:                60942897892                                 |
| TOTAL REGEX BATCHES:              928213                                      |
| VALID REGEX RESPONSES:            59411914                                    |
| REGEX RESPONSES WITH MATCHES:     33626582                                    |
| TOTAL REGEX MATCHES:              182318553                                   |
|                                                                               |
| AVERAGE REGEX BUFFER LENGTH:      1025.77                                     |
| MATCH TO BYTE RATIO:              334.27                                      |
|                                                                               |
| REGEX BUFFER RATE (Mbps):         1.9701                                      |
| REGEX PERFORMANCE (Gb/s):         16.1667                                     |
|                                                                               |
| TOTAL DURATION (secs):            30.1572                                     |
|                                                                               |
+-------------------------------------------------------------------------------+
```

While most of these fields are self-explanatory some fields require further definition:

- Packet Processing Rate (Mpps) – This is the rate which, in million packets per second, RXPBench has been able to acquire packets from the input source (Physical port or precached PCAP/text file). For the physical ports this rate may be different that the RegEx PPR value as not all packets (depending on configuration) may be sent to the Regex device.
- Packet Processing Perf (Gb/s) – The actual data-rate of the input source in Gigabits per second
- Total Regex Buffers – This is the number of complete buffers that were sent to the RegEx device for processing
- Total Regex Bytes – The total bytes contained within all buffers transmitted to the RegEx device for processing
- Total Regex Batches – RegEx buffers are gathered together into batches (based on the "-g" flag) and submitted to the RegEx device in a single operation

3.4.3. DPDK RegEx Stats Block

If the selected RegEx device is "rxp" or "regex_dpdk" the following block of statistics is provided. It presents more internal statistics from the DPDK RegEx device (BlueField-2 RXP):

```
+-------------------------------------------------------------------------------+
|                              DPDK REGEX STATS                                 |
+-------------------------------------------------------------------------------+
|                                                                               |
| INVALID RESPONSES:                0                                           |
| - TIMEOUT:                        0                                           |
| - MAX MATCHES:                    0                                           |
| - MAX PREFIXES:                   0                                           |
| - RESOURCE LIMIT:                 0                                           |
|                                                                               |
| TX BUSY - AVE PER CORE (secs):    27.8311                                     |
|                                                                               |
| MAX LATENCY (usecs):              3339.7540                                   |
| MIN LATENCY (usecs):              11.7405                                     |
| AVERAGE LATENCY (usecs):          519.3994                                    |
|                                                                               |
+-------------------------------------------------------------------------------+
```

The following are the definitions of each of these counters:

- Invalid Responses – These are responses from operations that have not completed successfully
- Timeout – When processing a block of input data a hardware triggered timeout occurred and the search was aborted
- Max Matches – The maximum number of configured matches was exceeded, and the job was aborted
- Max Prefixes – The maximum prefixes per scan was exceeded, and the job was aborted
- Resource Limit – A generic/internal resourcing limit was reached; the job was aborted
- Latency Figures – These provides max/min and average latency of jobs from transmitted to the DPDK RegEx device

It is important to note that in normal mode (i.e., not in latency mode) RXPBench ensures that the hardware is supplied with data that is designed to maximize throughput. As stated previously, latency figures in this mode are not calculated accurately. To view the correct hardware latency, make sure the `--latency-mode` option is provided. The following screenshot shows the RegEx stats with latency mode enabled:

```
+-------------------------------------------------------------------------+
|                            DPDK REGEX STATS                             |
+-------------------------------------------------------------------------+
|                                                                         |
| INVALID RESPONSES:                0                                     |
| - TIMEOUT:                        0                                     |
| - MAX MATCHES:                    0                                     |
| - MAX PREFIXES:                   0                                     |
| - RESOURCE LIMIT:                 0                                     |
|                                                                         |
| TX BUSY - AVE PER CORE (secs):    0.0000                                |
|                                                                         |
| RX IDLE - AVE PER CORE (secs):    3.9344                                |
|                                                                         |
| PER PACKET LATENCY - BATCH SIZE:  1                                     |
| - MAX LATENCY (usecs):            115.7178                              |
| - MIN LATENCY (usecs):            5.3722                                |
| - AVERAGE LATENCY (usecs):        5.7887                                |
|                                                                         |
+-------------------------------------------------------------------------+
```

3.4.4. Hyperscan Stats Block

If the selected Regex device is "hs" (or "Hyperscan") then an additional block of statistics is provided detailing the latency of requests to and response from the Hyperscan Library:

```
+-------------------------------------------------------------------------+
|                            HYPERSCAN STATS                              |
+-------------------------------------------------------------------------+
|                                                                         |
| MAX LATENCY (usecs):              471.2939                              |
| MIN LATENCY (usecs):              0.0168                                |
| AVERAGE LATENCY (usecs):          17.1162                               |
|                                                                         |
+-------------------------------------------------------------------------+
```

Chapter 4. General Configuration Options

Configuration options to control the operation of RXPBench can be provided either through a pre-defined configuration file or through the command line. If both a configuration file is supplied and a set of command line options then the command line options will supersede, effectively overriding, the options present in the configuration file.

4.1. Configuration File (-C, --config-file)

The configuration file option allows you to supply a text file that contains one or more options that would normally be present on the command line.

```
-C configuration.file
--config-file configuration.file
```

The file should contain each configuration option stripped of the leading dashes on a new line. A colon (:) should be placed between the option and the value. You may use either the short (-) or long (--) option name. For example:

```
input-mode : dpdk_port
m : inputfile.pcap
run-time-secs : 10
```

If the "-C" or "--config-file" option is used without any supplied parameter, RXPBench will attempt to open the default file "rxpbench.conf".

Note: Providing any additional command line options after the `-C` or `--config-file` will override any present within the configuration file.

4.2. DPDK EAL (-D)

RXPBench utilizes the DPDK framework to provide core memory management, packet ingress and Regular expression offloading. As common with DPDK applications there are several EAL options that can be used to ensure DPDK is optimally configured for the host environment.

EAL options should be enclosed in quotations ("..") and are passed directly to DPDK without any processing by RXPBench.

Please ensure if you are created a custom set of EAL commands that the “class=regex” parameter is included to ensure the Regex devices is available for use. You should use the “class=eth:regex” if you wish to use packet acquisition from physical ports and Regex.

Note: The CPU cores selected for use through the EAL options will be the same cores used by the whole RXPBench application.

Note: Care should be taken when selecting EAL options. Misconfiguration may affect the utilities ability to obtain maximum performance on the target hardware. A full list of EAL options is provided by DPDK.

4.3. Verbose (-V)

This option provides additional verbose output on any matching patterns found by the Regex algorithm. The supplied integer value dictates the amount of information provided:

```
-V 1
-V 2
-V 3
```

All verbose levels will write out to a CSV files named "`rxpbench_matches_main_core_XX.csv`", where XX is the main logical core ID returned by DPDK, and "`rxpbench_matches_core_XX.csv`" for additional cores in a multicore environment.

Each entry in the CSV file provides match information including queue ID, rule ID, start offset and length. If the verbose level is set to `3` then the match string is also returned.

Note: `-V 2` and `-V 3` will cause the writing of large amounts of data if a substantial number of matches are reported and it may result in characters that break the CSV format (such as commas, new lines, etc.) being placed in the output file. In extreme cases this may result in a performance reduction.

4.4. Cores (-c)

The "Cores" option allows for the configuration of the total number of cores available to RXPBench.

```
-c 4
```

The use of the CPU cores is dependent on the application’s Regular Expression algorithm and whether packets are being received from an ingress port or PCAP capture file.

If the BlueField-2 RXP hardware accelerator is used each core will be given a unique DPDK Regex queue to operate on; if the accelerator is Hyperscan then each core will be used to execute the Hyperscan software library.

In addition, if packets are being received from a physical port, the value will be used to allocate X number of DPDK Tx and Rx queues on the port.

Note: The value supplied here must be ⩽ the number of cores provided in the -D (EAL) options. If an invalid value is supplied a warning will be produced and the EAL (-D) core count will be used.

Chapter 5. Algorithm, Ingress, and Rules Options

This group of options provides the ability to select the Algorithm (BF2 RXP or Hyperscan), where input data should be received from (physical ports, text files, or PCAP files) and Regular Expression rules information.

5.1. Algorithm/Device Select (--Regex-dev, -d)

This option allows you to select the underlying framework to use (DOCA/DPDK) and whether acceleration should be provided by the BlueField RXP hardware accelerator or the Hyperscan software library.

Each option is provided with a short (i.e. `doca`) or long (i.e. `doca_regex`) version:

```
--Regex-dev regex_dpdk
--Regex-dev rxp
--Regex-dev hyperscan
--Regex-dev hs
```

5.2. Input Mode (--input-mode, -m)

RXPBench can receive data from various input sources. This option allows you to provide which method you require:

```
--input-mode dpdk_port --dpdp-primary-port X --dpdk-secondary-port Y
--input-mode pcap_file
--input-mode text_file
--input-mode job_format
```

5.2.1. --input-mode dpdk_port, -m dpdk_port

The DPDK port option enables RXPBench to receive live traffic from a port, specified in the `--dpdk-primary-port`. If the secondary port option exists (`--dpdk-second-port`) then any packets received, after pattern matching has occurred, are transmitted onto the second port.

See section DPDK Port Operations for more information.

5.2.2. --input-mode pcap_file, -m pcap_file

This option allows you to supply an external PCAP file. This allows for reproducible results using a known input file. The entire payload recorded in each frame within the pcap file is made available to RXPBench.

5.2.3. --input-mode text_file, -m text_file

If processing of a standard text file is required, this option allows you to select any file. The entire text file contents are made available to the RXPBench application with no parsing or changes made.

5.2.4. --input-mode job_format, -m job_format

This option is used to provide a specific "job format" directory to RXPBench. This directory contains files, provided by NVIDIA or through your NVIDIA Networking Support representative, that include data and results to validate that matches returned by the algorithms are expected.

This allows RXPBench to validate that all aspects of the hardware, libraries, and software are operating correctly. In normal operation, this mode is not used, but information on this mode is provided for your reference.

5.3. Compiled Rules File (--rules, -r)

The RXP hardware accelerator can accept regular expressions that have been externally compiled using the RXPC (RXP Compiler) into a ROFF file. This option allows you to specify this ROF2 file.

5.4. Uncompiled Rules File (--raw_rules, -R)

RXPBench can accept an input file containing raw regular expressions. The uncompiled rules file can be in either of these formats:

- RXP rules file
- Hyperscan rules file

The tool can accept either format of rules file, regardless of which algorithm (BlueField-2 RXP, or Intel Hyperscan) is used. In the case where a rules file is not in the expected format for the algorithm, a conversion process is employed to ensure they operate correctly.

Note: In this configuration, the BlueField-2 RXP compiler is configured with its default optimizations; enhanced performance can be obtained through the adjustment of these parameters. For more information see the NVIDIA RXP Compiler, and provide any compiled rules through the `-rules`/`-r` option.

5.5. App-Layer Filtering (--run-app-layer, -A)

This option will cause RXPBench to extract the upper-layer data from the received packets and submit them for regular expression testing. Upper-layer data includes data found in TCP and UDP streams found in IPv4 and IPv6 packets (including any such data contained within VLAN tagged packets).

Chapter 6. DPDK Port Operations

When input data is received from a physical network port, the options in the subsequent sections can be used to configure the ports.

6.1. Primary Port (--dpdk-primary-port, -1)

This is the port where packets will be received from. The supplied ID is used directly to access the requested DPDK port.

6.2. Secondary Port (--dpdk-secondary-port, -2)

RXPBench can be used as a "bump" in the wire, where received packets are pattern matched before transmission through a secondary port. This option provides the port ID, as used by DPDK, for the onwards transmission of scanned packets.

Chapter 7. RegEx Compilation Operations

This section provides information on options that affect the compilation of regular expressions.

7.1. Force Compilation (--force-compile, -F)

RXPBench can accept both uncompiled and compiled rules. As part of the initialization process, any uncompiled regular expression rules must be compiled into object code that can be executed on the BlueField RXP hardware accelerator or Intel™ Hyperscan software library.

While the BlueField RXP supports a wide range of RegEx constructs, both itself and Hyperscan cannot provide for all constructs due to complexity and performance impacts.

When a supplied set of regular expressions is compiled, either algorithm may abort the compilation due to the inclusion of one or more unsupported rule constructs. This option prevents the compilers from aborting, and forces RXPBench to continue with the rules that successfully compiled.

7.2. Single-line Mode (--comp-single-line, -S)

This option activates single-line mode so any new line characters will not match.

7.3. Caseless Matching (--comp-caseless, -i)

This option activates caseless matching which causes all rules to be seen as case insensitive.

7.4. Anchoring Multi-line Mode (--comp-multi-line, -u)

This option controls how anchors are handled in regular expressions. If enabled, anchoring is applied per line.

7.5. Free Spacing Mode (--comp-free-space, -x)

This option activates free spacing mode. Effectively, the white spaces in rules are ignored.

Chapter 8. Runtime Options

This groups of options allows for the specific duration of any execution to be controlled through various metrics.

8.1. Runtime Seconds (--run-time-secs, -s)

This option sets the time in seconds that a test ought to be run for. If a file is used as input and no `--run-num-iterations/-n` are set then the file is looped over until the time period is met.

8.2. Iterations (--run-num-iterations, -n)

If input data is being received from either a PCAP File or text file, this option is used to limit the execute to a complete number of iterations of the input file.

For example, if an iteration count of 4 was given on a PCAP file contains 1,000 packets. The total number of packets processed would be 4,000. If the input file was a standard text file containing 5,000 bytes of information, an iteration count of 4 would mean 20,000 bytes would be read by RXPBench.

Note: If iterations are used along with a `runtime-seconds` option, the test will finish with whatever limit comes first.

8.3. Packet (--run-packets, -p)

This option sets the total number of packets that are read from the selected input mode. After this number of packets is read from a file or received from a network port, rxpbench will complete.

8.4. Total Bytes (--run-bytes, -b)

Regardless of the input mode this is the total number of bytes received that is required to mark the execution as complete.

For live traffic this is the total number of bytes received from the physical port. For both the PCAP input file and text file this is the total bytes to read from the input files.

Chapter 9. Search-specific Options

These options allow for finer control of data to be transmitted to the RegEx device for pattern matching.

9.1. Buffer Length (--buf-length, -l)

When the RXPBench is reading from input files (whether PCAP, or text files) it has all the information readily available (unlike live traffic which must be received). This allows the application to read a variable amount of input data per iteration.

This option controls the amount of data that is read from the input file and passed to the RegEx algorithm.

Note: With PCAP capture files this option may result in data be transmitted from part of packet, or alternatively multiple packets, if the buffer length supplied is less than or greater than the PCAP's frame length.

9.2. Buffer Threshold (--buf-thres, -t)

When live traffic is being received from a physical port, this option specifies the received packets minimum size before it will be processed.

For example, setting this value to 256 bytes means that if a packet arrives that is less than 256 bytes in length it will not be processed by RXPBench.

Packets that are dropped by this threshold are recorded in the statistics under "UNDER THRES" field.

9.3. Buffer Overlapping (--buf-overlap, -o)

When the input is being read from files (either PCAP, or text files) this option allows a certain number of bytes to be overlapped from the previous frame.

9.4. Batching (--buf-group, -g)

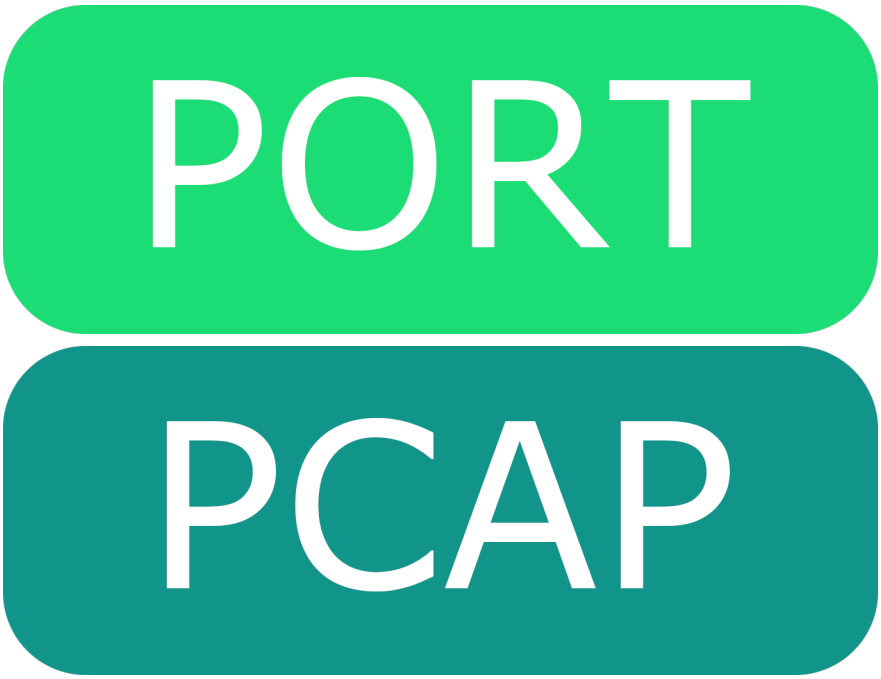
Most high-performance applications obtain additional performance by batching together multiple operations into a single process.

DPDK Regex provides the capability of enqueuing multiple buffers to the BlueField-2 RXP Hardware accelerator. This option allows you to specify how many payloads should be grouped together before enqueuing on the hardware.

If receiving packets from a physical port this also determines the batch size to read (and write) to the network ports.

If this option is not supplied, RXPBench defaults to grouping (batching) together 64 packets at a time.

9.5. Layer 5 to 7 Payloads Only (--run-app-layer, --A)



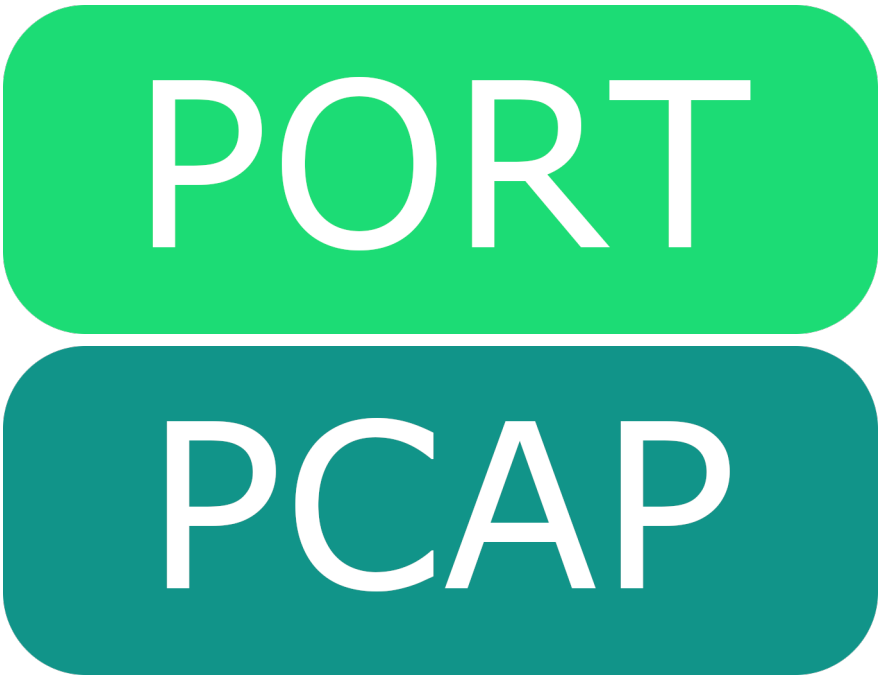
This option will process each received payload packet and identify any layer 5 to layer 7 information present in them. It will then send only this layer 5 to layer 7 data to the RegEx algorithm.

For example, if a 500-byte packet is received that contains 60 bytes of layer 1 to layer 4 data, then the first 60 bytes are ignored and the 440 bytes of layer 5 to layer 7 data is sent to the RegEx algorithm.

For PCAP-based input files, any `-l` (or `--buf-length`) option will be overwritten and lengths will be assigned on a per-packet basis. Similarly, for live traffic received from a physical port, each packet is processed independently with data from their layer 5 through 7 being sent to the RegEx algorithm. It may be appropriate to use the threshold option (`--buf-thres`, `-t`) to remove small payloads.

Note: Using this option in live mode may increase the average job size due to the skipping of certain "no payload" frames (such as TCP ACKs) that would otherwise be included.

9.6. Sliding Window (--sliding-window, -w)



DOCA

The BlueField RegEx engine accepts jobs up to a maximum length of 16KB. Afterwards it rejects the job as invalid. This DOCA-only feature provides user the ability to supply huge jobs (up-to 2GB) in length.

To enable this option, provide the `--sliding-window` or `-w` option with an optional positive integer argument that defines the size of the window to use. This window size can be in the range of 0 to 16383 (default is 32).

Internally, the DOCA framework fragments the job into smaller buffers that can be accepted by the hardware and then reassembles the results of the fragmented searches into a single result (the framework takes care of pruning any duplicate results).

The window is effectively the number of bytes appended to the start of a job, which belong to the end of the previous job fragment. This "window" effectively moves forward through data looking for matches within it.

A match that is up to sliding-window-size bytes long is guaranteed to be found. Any match longer than the window size may be missed if it happens to appear across the boundary of two fragments. Therefore, correct selection of the sliding window size is paramount.

Please note this mode has a performance impact as some job data may need to be scanned twice. Therefore, it is recommended that you use the smallest possible window size necessary for your case.

Chapter 10. BlueField RXP-specific Operations

These options provide control over features available in the DPU's RXP hardware accelerator.

10.1. Latency Mode (--latency-mode, -8)

RXPBench provides a latency figure as part of the "RXP Stats" section. This latency figure can be calculated in two different ways depending on your requirements.

By default, when RXPBench executes, it keeps the RXP hardware queue filled with as much data as possible. This provides the maximum performance but does not provide a true representation of the hardware latency in the calculated statistics.

To see the actual latency of the RXP hardware, you must enable latency mode. Once enabled, this mode batches together packets (either 64 packets, or a user-supplied value using the `--buf-group`/ `-g` batching options) and then waits on the results of that batch before calculating the latency.

Using this method, the latency returned and displayed shows a truer representation of the hardware latency offered by the RXP offload engine which can be compared to that of Hyperscan.

Chapter 11. Hyperscan-specific Operations

Several options that are specific to the Hyperscan software library exist. These allow you to modify the behavior of the Regular Expression pattern matching engine.

Note: Hyperscan does not support both flags being enabled at the same time.

11.1. HS Single Match (--hs-singlematch, -H)

The Hyperscan algorithm provides an option called "HS_FLAG_SINGLEMATCH". Please see the Hyperscan documentation for more information.

11.2. HS Left Most Match (--hs- leftmost, -L)

The Hyperscan algorithm provides an option called "HS_FLAG_SOM_LEFTMOST", please see the Hyperscan documentation for more information.

Chapter 12. Running RXPBench on BlueField

RXPBench utilizes the DPDK framework to provide packet operations and hardware-accelerated regular expression (RegEx) offloading (`dpdk_regex`).

RXPBench can run in the following input modes: Port, PCAP, or text file.

- In port mode, live traffic is received from a DPDK port to receive live traffic from a port specified in the `--dpdk-primary-port` configuration option. If the secondary port option exists (`--dpdk-second-port`), then any packet received, after pattern matching has occurred, is transmitted onto the second port.
- In PCAP mode, traffic is supplied via an external PCAP file. This allows for reproducible results using a known input file. The entire payload recorded in each frame within the PCAP file is made available to RXPBench.
- Text file mode allows the user to select any file when processing of a standard text file is required. The entire text file contents are made available to the RXPBench application with no parsing or changes made.

To run RXPBench on BlueField, follow these steps:

1. Refer to the NVIDIA DOCA Installation Guide for details on how to install BlueField related software.

   Note: The RXPBench tool is supplied in both binary and source package formats as described earlier in this document.

2. Before executing RXPBench, an installation of Hyperscan must be present on the host. Hyperscan can be obtained from the Linux distribution package manager (apt, dpkg, yum, etc.) or alternatively compiled from the source. Depending on the Linux distribution on the host, the following Hyperscan versions are required:

Host Linux Distribution	Hyperscan Version	Installation Command
Ubuntu 18.04	4	`apt install libhyperscan4`
Ubuntu 20.04	5	`apt install libhyperscan5`
CentOS 7.x	–	Hyperscan is provided through 3rd party vendors. The following command will install Hyperscan 5.3.0 on CentOS 7:

Host Linux Distribution	Hyperscan Version	Installation Command
		`yum install epel-release` `sudo yum install http://repo.openfusion.net/centos7-x86_64/hyperscan-5.3.0-1.of.el7.x86_64.rpm`

CentOS 8.x	–	Hyperscan is provided through 3rd party vendors. The following command installs Hyperscan 5.3.0 on CentOS 8:
		`yum install epel-release`
		`sudo yum install https://download-ib01.fedoraproject.org/pub/epel/8/Everything/x86_64/Packages/h/hyperscan-5.3.0-5.el8.x86_64.rpm`

3. Build the RXPBench tool from the source code. RXPBench source code packages are found in the following locations:

   - Ubuntu 18.04 – `/usr/share/doca-host-repo-ubuntu1804-1.3.0/repo/main`
   - Ubuntu 20.04 – `/usr/share/doca-host-repo-ubuntu2004-1.3.0/repo/main`
   - Debian 10.8 – `/usr/share/doca-host-repo-debian108-1.3.0/repo/main`

   The source code is unpacked using the following commands for example.

   - For Ubuntu:
     ```
     dpkg-source -x rxpbench_21.06.0.dsc
     ```
   - For CentOS:
     ```
     rpmbuild --recompile rxpbench-21.06-1.el7.src.rpm
     ```

4. To re-build the RXPBench tool. Run:
   ```
   cd <source extract directory>/rxpbench-21.06.0
   make
   ```

   The RXPBench executable will be located in the `build` subdirectory.

   The build process depends on the `PKG_CONFIG_PATH` environment variable to locate the DPDK libraries. If the variable was accidently corrupted, and the build fails, please run the following command.

   - For Ubuntu:
     ```
     export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/opt/mellanox/dpdk/lib64/pkgconfig
     ```
   - For CentOS:
     ```
     export PKG_CONFIG_PATH=/opt/mellanox/dpdk/lib64/pkgconfig:/usr/local/lib64/pkgconfig:/usr/lib64/pkgconfig/
     ```

5. RXPBench requires the following configurations to enable RegEx.

   a). On the host side, stop the driver. Run:
   ```
   host$ sudo /etc/init.d/openibd stop
   ```

   b). Log onto the BlueField-2 and run the following commands:
   ```
   dpu$ sudo /etc/init.d/openibd start
   dpu$ echo 1 > /sys/class/net/p0/smart_nic/pf/regex_en
   dpu$ current_huge='cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages'
   dpu$ echo $((200 + current_huge)) > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
   ```

```
dpu$ systemctl start mlx-regex
```

c). Verify that the service is running. Run:

```
dpu$ systemctl status mlx-regex
```

d). On the host, start the driver and add hugepages. Run:

```
host$ sudo /etc/init.d/openibd start
host$ echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
```

6. To run the application:

```
cd build
./rxpbench [dpdk_flags] -- [additional application flags]
```

For example:

```
./rxpbench -D "-l 0,1,2,3 -n 1 -a 5e:00.0,class=regex –file-prefix=rxpbench -a
 5e:00.01" --input-mode text_file -f ../Shakespeare.txt -d rxp -R ../rules.hs -l
 2048 -n 10000 -c 1
```

- This command runs in the text file input mode (`--input-mode`).
- The input file is Shakeseare.txt (`-f`).
- This command uses the RXP device for pattern matching (`-d`).
- The RXP device is programmed with the rules specified in `rules.hs` (`-R`).
- This command sends 2048 bytes of data to be searched in each job (`-l`).
- This command reads and processes the input text file 10,000 times (`-n`).
- This command uses 1 CPU core during the run (`-c`).

Information on the complete set of configuration settings and options may be found in other sections of this document.

As RXPBench executes, statistics will be updated on screen periodically. On exit, summary information will be displayed on screen.

Chapter 13. BlueField-2 Performance Overview

To demonstrate the expected performance of the NVIDIA® BlueField®-2 DPU using RXPBench, 3 different datasets are used:

- An open-source text file containing the works of William Shakespeare
- A PCAP of captured HTTP/port 80 traffic (closed source),
- A PCAP taken from the National CyberWatch Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC)

5 different RegEx rulesets are selected:

- The l7-filter application recognition rules
- The well-known open-source snort_pcres, snort_literals, and teakettle_2500
- A selection of Web Application Firewall RegEx rules taken from the OWASP core-ruleset

All rules are compiled using the RXP Compiler with default options and the addition of some HTTP keywords as a graylist option.

The table below shows the number of rules compiled for each ruleset. Uncompiled rules either contain unsupported PCRE syntax or are considered as "bad" or potential DoS rules by the compiler.

Ruleset	Rules Compiled/Total Rules
l7-filter	126/142
snort_pcres	769/847
snort_literals	2381/3116
teakettle_2500	2500/2500
owasp-waf	117/123

The rules are run against the different datasets in job lengths of 2KB on a single core of both an x86 host and Arm on the BlueField-2 DPU (model MBF2H516A-EEEOT).

The following command line is an example of the RXPBench parameters used in the tests:

```
rxpbench -D "-l5,6 -n 1 -a 03:00.0,class=regex" --input-mode text_file -f
 Shakespeare.txt -d rxp -r snort_pcre.rof2.binary -c 1 -s 10 -l 2048
```

The table below presents the performance results achieved for the different datasets on both Arm and the x86 host. All results are in Gb/s.

	Shakespeare		Port80		MACCDC	
x86/Arm	x86	Arm	x86	Arm	x86	Arm
l7-filter	50.95	50.97	50.95	50.85	50.93	50.76
snort_pcres	27.26	27.68	5.67	5.67	6.20	6.23
snort_literals	50.95	50.98	26.70	19.96	14.26	8.80
teakettle_2500	15.77	15.77	50.94	50.92	50.93	50.75
owasp-waf	3.96	3.97	28.08	28.51	50.59	50.63

The results show that 50Gb/s pattern matching throughput can be achieved when applying complex regular expression rulesets to various datasets. Some of the ruleset/dataset combinations show performance below the maximum RXP bandwidth. This is down to a combination of complex rules that require a lot of processing and data that contains a lot of matches or partial matches.

For example, the Owasp-waf rules are known to contain a lot of common English language words which are followed by a "dot star". This means that, when applied to English language data, a lot of extra processing is required to validate full matches. Our tests show that software algorithms are impacted by similar scenarios and, while the RXP throughput is well below line rate, it still offers a significant performance boost over software.

The throughput reported by RXPBench when run on both the x86 host and the Arm is approximately the same in almost all cases. This highlights the benefit of the offload engine in that the power of the CPU used for applications has a limited effect on the pattern matching capabilities.

It is only the snort_literals ruleset that has taken a performance hit. Here, the rules produce many matches. This means that more effort is required by the CPU to process the results. Adding a second BlueField-2 Arm core to RXPBench pushes the performance to the same levels achieved as the x86 host.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation nor any of its direct or indirect subsidiaries and affiliates (collectively: "NVIDIA") make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

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

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

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

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

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

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF 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, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com