NVIDIA DOCA RegEx

Programming Guide

MLNX-15-060489 _v1.3 | May 2022

Table of Contents

Chapter 1. Introduction

DOCA RegEx is a library that provides RegEx pattern matching to DOCA applications. It provides access to the regular expression processor (RXP) , a high-performance, hardware-accelerated RegEx engine available on the NVIDIA® BlueField® DPUs, and can utilize software-based engines when required.

Using DOCA RegEx, developers can easily execute complex regular expression operations in an optimized, hardware-accelerated way.

This document is intended for software developers wishing to accelerate their regular expressions operations.

Chapter 2. Prerequisites

DOCA DMA-based applications can run either on the host machine or on the DPU target.

The RegEx engine is enabled by default on the DPU. However, to enable RegEx offloading on the host, run:

```
host> sudo /etc/init.d/openibd stop
host> sudo echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
dpu> echo 1 > /sys/bus/pci/devices/0000\:03\:00.0/regex/pf/regex_en
dpu> cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages 400
# Make sure to allocate 200 additional hugepages
dpu> echo 600 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
dpu> systemctl restart mlx-regex
# Verify the service is properly running
dpu> systemctl status mlx-regex
host> sudo /etc/init.d/openibd start
```

Chapter 3. Architecture

DOCA DMA-based applications can run either on the host machine or on the DPU target.

The RegEx engine is enabled by default on the DPU. However, to enable RegEx offloading on the host, run:

```
host> sudo /etc/init.d/openibd stop
host> sudo echo 1024 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
dpu> echo 1 > /sys/bus/pci/devices/0000\:03\:00.0/regex/pf/regex_en
dpu> cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages 400
# Make sure to allocate 200 additional hugepages
dpu> echo 600 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
dpu> systemctl restart mlx-regex
# Verify the service is properly running
dpu> systemctl status mlx-regex
host> sudo /etc/init.d/openibd start
```

DOCA RegEx provides a flexible API for programming regular expression databases, enqueuing jobs and dequeuing results. The API operates asynchronously allowing many pattern matching operations to be executed in parallel.

The library provides both hardware- and software-based pattern matching. This allows the library to fall back to software support if, for example, hardware acceleration is not available or for certain operations.

3.1. Rule Compilation

Regular expressions are provided as "compiled" rule files to the library, and must therefore be externally compiled by a "compiler" prior to loading by the library. For hardware acceleration, the external compiler is termed " rxpc" (RXP compiler) and generates RXP object format (ROF) binary files that represent the compiled regular expressions.

3.2. RegEx Implementations

The library itself is designed to support multiple RegEx engine implementations. These can be either hardware devices or software libraries.

At run-time, as part of the initialization process, you must effectively create the required SW and HW devices before passing them to DOCA RegEx as the hardware (hw) or software (sw) implementations.

DOCA RegEx provides some helper functions to assist with the creation of DOCA devices (e.g., `doca_regex_create_pre_configured_regex_impl`). Care should be taken to initialize the DOCA RegEx device with any specific options it requires prior to registering it with `doca_regex_hw_device_set` or `doca_regex_sw_device_set`, and to ensure that destruction of the device instance is correct.

3.3. Sliding Window

The library includes a facility to accept job lengths that are greater than the maximum size supported by an engine. The library fragments incoming jobs into smaller fragments and processes them sequentially looking for potential matches. The sliding window mechanism takes data from the end of the previous fragment and appends it to the start of the next fragment (the "size" of the window) to find additional matches. See the `doca_regex_overlap_size_set` API call for more information.

3.4. Small Job Threshold

When operating on certain data sets it may be more effective to avoid offloading the jobs to hardware if certain conditions are met. Each use case is different but, in general, if the job is smaller than a specific size, the overhead of offloading the job to hardware may be greater than executing it in software. Note that to enable this feature, you must have both a HW and SW device available to DOCA RegEx. For more information, see the `doca_regex_small_job_sw_offload_threshold_set` API call.

3.5. Software Fallback

If, during any operation, the hardware is unable to process incoming incoming jobs, DOCA RegEx can divert those jobs to the software RegEx engine . Note that to enable this feature, you must have both a hardware and software device available to DOCA RegEx. For more information, see the `doca_regex_sw_fallback_enabled_set` API call.

Chapter 4. API

This section details the specific structures and API operations related to the DOCA RegEx library.

Note: The pkg-config (`*.pc` file) for the RegEx library is named `doca-regex`.

4.1. doca_regex_job_request

This structure contains information on the job to be submitted to DOCA RegEx.

```
struct doca_regex_job_request {
 uint64_t id;
 uint16_t rule_group_ids[4];
 struct doca_regex_buffer const *b uffer;
};
```

Where:

- `id` – a user-defined field used to correlate the matches with the enqueued job
- `rule_group_ids` – an array of IDs which can be used to select which group of rules are used to process this job. Set each value to a non-zero value to enable group selection, or to 0 to ignore it.
- `buffer` – a pointer to a buffer containing the data to be scanned

4.2. doca_regex_buffer

This structure contains information related to the required DOCA RegEx operation.

```
struct doca_regex_buffer {
 void const *address;
 uint32_t length;
 uint32_t has_mkey;
 uint32_t mkey;
};
```

Where:

- `address` – a pointer to job data. This must remain valid until the response is returned for the enqueued job.
- `length` – the number of bytes in this job

- `has_mkey` – any non-zero value indicates that the `mkey` field is valid
- `mkey` – if `has_mkey` is > 0, this field contains the Mkey value

4.3. doca_regex_job_response

When a job response is dequeued, this structure is populated with any match information.

```
struct doca_regex_job_response {
 uint64_t id;
 uint64_t status_flags;
 uint32_t detected_matches;
 uint32_t num_matches;
 struct doca_regex_match *matches;
};
```

Where:

- `id` – the `id` value as supplied by the user during enqueue. See doca_regex_job_request for more information.
- `status_flags` – a bit-masked field for zero or more status flags. See `doca_regex_status_flag` for more information.
- `detected_matches` – the total number of detected matches
- `num_matches` – the total number of matches returned in this response (may be fewer than `detected_matches`)
- `matches` – a linked list of match structures (`num_matches` in length)

4.4. doca_regex_job_match

When a job response is dequeued, this structure is populated with any match information.

```
struct doca_regex_match {
 struct doca_regex_match *next;
 uint32_t match_start;
 uint32_t rule_id;
 uint32_t length;
};
```

Where:

- `next` – as matches are linked together using a linked list, this is the pointer to the next match in the linked list
- `match_start` – the index relative to the start of the job of this match
- `rule_id` – the ID of the rule that generated this match
- `length` – the length of the matched value

4.5. Instance Construction/Destruction API

This section details API calls related to the creation and destruction of DOCA RegEx instances.

4.5.1. doca_regex_create

Creates a DOCA RegEx instance.

```
struct doca_regex *doca_regex_create(void);
```

This function returns `doca_regex` object on success. NULL otherwise.

4.5.2. doca_regex_destroy

Destroys a previously created DOCA RegEx instance.

```
void doca_regex_destroy(struct doca_regex *regex);
```

Where:

- `regex [in]` – a pointer to a previously created DOCA RegEx instance

4.6. RegEx Device Creation and Destruction

DOCA RegEx devices are the hardware devices and/or software implementations that perform pattern matching. DOCA RegEx supports the creation of one hardware device and/or one software device, allowing it to utilize both hardware and software for maximum benefit.

RegEx devices are separate user-managed objects that must be created and registered with DOCA RegEx as either hardware or software devices.

These devices include their own specific, per implementation, initialization and destruction API calls. The standard practice is to `init_fn` the devices after creation. Then, after use, to call the `cleanup_fn` and `destroy_fn` to remove the device.

4.6.1. doca_regex_create_pre_configured_regex_impl

DOCA RegEx provides some RegEx implementations that are ready to use. This saves on the effort of developing them yourself. This function returns an existing implementation using a supplied name.

```
struct doca_regex_device *doca_regex_create_pre_configured_regex_impl(char const
 *name);
```

Where:

- name [in] – the name of the required implementation. Can be `bf2` or `hyperscan`.

This function returns a non-null pointer to the created device upon success. Null can be returned upon failure to create the device or if the given name was not recognized.

4.6.2. init_fn

This function exists on the `doca_regex_device` you previously created and provides a method of initializing it.

```
int init_fn(struct doca_regex_device *inst, const char *dev_addr);
```

Where:

- `inst [in]` – instance pointer of the device
- `dev_addr [in]` – PCIe address of RegEx device (usually NULL if a software device)

The function returns 0 on success or a negative POSIX status code.

4.6.3. cleanup_fn

This function exists on the `doca_regex_device` and is used to allow device clean-up prior to destruction.

```
int cleanup_fn(struct doca_regex_device *inst);
```

Where:

- `inst [in]` – instance pointer of the device

The function returns 0 on success or a negative POSIX status code.

4.6.4. destroy_fn

This function exists on the `doca_regex_device` and destroys the created instance. After this call, the instance pointer can be set to NULL.

```
int destroy_fn(struct doca_regex_device *inst);
```

Where:

- `inst [in]` – instance pointer of the device

The function returns 0 on success or a negative POSIX status code.

4.7. RegEx Device Registration

After the selected RegEx devices have been created, they must be register ed with DOCA RegEx. This section details the API calls required to register the devices.

4.7.1. doca_regex_sw_device_set

This function registers a previously created software-based RegEx device with DOCA RegEx. The application ensures the lifetime of the device is maintained until the device is either

deregistered or the doca_regex instance is destroyed. Pass a NULL value to this function to deregister a registered device.

```
int doca_regex_sw_device_set(struct doca_regex *regex, struct doca_regex_device
 *device);
```

Where:

- regex [in] – the DOCA RegEx instance
- device [in] – the DOCA RegEx device instance to be used as the software device

The function returns 0 on success, or a negative POSIX status code.

4.7.2. doca_regex_hw_device_set

This function registers a previously created hardware-based RegEx device with DOCA RegEx. The application ensures the lifetime of the device is maintained until the device is either deregistered or the doca_regex instance is destroyed. Pass a NULL value to this function to deregister a registered device.

```
int doca_regex_hw_device_set(struct doca_regex *regex, struct doca_regex_device
 *device);
```

Where:

- regex [in] – the DOCA RegEx instance
- device [in] – the DOCA RegEx device instance to be used as the hardware device

The function returns 0 on success, or a negative POSIX status code on failure.

4.8. DOCA RegEx Setup

This section details the API calls required to setup DOCA RegEx with memory to store received matches, adjust the number of queue pairs, etc.

4.8.1. doca_regex_mempool_create

This function creates a single producer and single consumer memory pool that can store RegEx matches.

```
struct doca_regex_mempool *doca_regex_mempool_create(size_t elem_size, size_t
 nb_elems);
```

Where:

- elem_size [in] – the required size of each element. For RegEx matches, the value sizeof(struct doca_regex_match) should be used.
- nb_elems [in] – the number of elements the memory pool should hold

The function returns a pointer to the memory pool on success, or NULL on failure.

4.8.2. doca_regex_num_qps_set

Specifies the number of queue pairs to use for this DOCA RegEx instance. This function should only be called when the instance is not running. By default, it should be set to 1.

```
int doca_regex_num_qps_set(struct doca_regex *regex, uint16_t num_qps);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `num_qps [in]` – the number of queue pairs to assign to the instance. The default is 0.

The function returns 0 on success and a negative status code on failure.

4.8.3. doca_regex_qp_mempool_set

Register a memory pool to the DOCA RegEx instance so it can acquire doca_regex_match objects without requiring memory allocations. If an application does not wish to get match details from a search, then this can be left out. As the mempool receives all matches, for all jobs, it should be sufficiently sized for the maximum matches expected (i.e., maximum matches per job multiplied by the number of jobs you wish to process at a time).

Note: After you have completed your processing of any RegEx matches, you must return each one of them to the mempool. See `doca_regex_mempool_obj_put` in the NVIDIA DOCA Libraries API Reference Manual for more information.

This call should only be executed when the DOCA RegEx device is not running after calling `doca_regex_num_qps_set`.

```
int doca_regex_qp_mempool_set(struct doca_regex *regex, struct doca_regex_mempool
 *mp, uint16_t qid);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `mp [in]` – the memory pool that contains any found matches
- `qid [in]` – the ID of the queue to associate with this mempool

The function returns 0 on success and a negative status code on failure.

4.9. Configuration Options

DOCA RegEx has several options that alter its mode of operation and control certain features. This section details those API calls and their related impact.

4.9.1. doca_regex_overlap_size_set

This API call enables the sliding window functionality of the DOCA RegEx instance, allowing it to find matches in data that exceeds the maximum job length of a particular RegEx device. For example, the BlueField RXP hardware device has a maximum job size of 16KB.

This function is provided with a size parameter that indicates the size of overlap to use in the sliding window algorithm. This algorithm breaks up the incoming job data into fragments. Therefore, the overlap size causes data from the previous fragment to be prepended to the start of the next fragment.

As this overlap impacts performance (job data may get searched multiple times) the overlap size should be kept to a minimum value that still guarantees that matches are found.

```
int doca_regex_overlap_size_set(struct doca_regex *regex, uint16_t
 nb_overlap_bytes);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `nb_overlap_bytes [in]` – the number of bytes for the overlap functionality to use

The function returns 0 on success, and a negative status code on failure.

4.9.2. doca_regex_small_job_sw_offload_threshold_set

This function defines the threshold for the "small jobs" feature. This feature automatically executes any RegEx jobs on the software device driver (if one is registered) when the size of the job is below a certain threshold.

```
int doca_regex_small_job_sw_offload_threshold_set(struct doca_regex *regex, uint16_t
 threshold);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `threshold [in]` – number of bytes. When a job has fewer bytes than the threshold, the engine prefers sending the job to the software.

The function returns 0 on success, and a negative status code on failure.

4.9.3. doca_regex_sw_fallback_enabled_set

This function enables or disables software fallback. When a job is unable to be executed on a hardware RegEx device, the engine can automatically re-execute the job on the software device.

```
int doca_regex_sw_fallback_enabled_set(struct doca_regex *regex, bool enabled);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `enabled [in]` – determines whether to enable this functionality or not

The function returns 0 on success, and a negative status code on failure.

4.10. Programming RegEx

As part of initialization, the RegEx devices must be programmed with compiled regular expressions. This compilation process takes place offline and generates a compiled file that can be given to a selected device.

4.10.1. doca_regex_program_compiled_rules

This function programs the registered hardware and software RegEx devices using rules that are already loaded into memory as pointers to arrays of bytes.

```
int doca_regex_program_compiled_rules(struct doca_regex *regex,
                                      char const *hw_compiled_rules_bin,
                                      size_t hw_compiled_rules_size,
                                      char const *sw_compiled_rules_bin,
                                      size_t sw_compiled_rules_size);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `hw_compiled_rules_bin [in]` – a pointer to a buffer of pre-compiled binary rules data suitable for use by the selected hardware device
- `hw_compiled_rules_size [in]` – the size, in bytes, of the hardware specific pre-compiled binary rules data
- `sw_compiled_rules_bin [in]` – a pointer to a buffer of pre-compiled binary rules data suitable for use by the selected software device
- `sw_compiled_rules_size [in]` – the size, in bytes, of the software specific pre-compiled binary rules data

The function returns 0 on success of writing at least one rule to either a hardware or software device, and a negative status code on failure.

4.11. Executing Jobs and Receiving Matches

The DOCA RegEx API provides an asynchronous method for enqueuing job data and dequeuing detected matches.

4.11.1. doca_regex_enqueue

This function enqueues a job to the DOCA RegEx instance.

```
int doca_regex_enqueue(struct doca_regex *regex, uint16_t qid,
                       struct doca_regex_job_request const *job,
                       bool allow_aggregation);
```

Where:

- `regex [in]` – the DOCA RegEx instance

- `qid [in]` – the ID of the queue in which to enqueue the job
- `job [in]` – a DOCA RegEx job to be enqueued. The caller retains ownership of the data.
- `allow_aggregation [in]` – when set, the RegEx device may choose to not begin processing this job immediately to maximise overall efficiency and throughput. When not set, the RegEx engine must begin processing immediately, potentially reducing latency.

  This allows an application to favor either throughput or latency. If in doubt, it is recommended to favor throughput.

The function returns:

- 0 – device busy, wait until one or more results are dequeued before enqueuing more jobs
- 1 – job enqueued successfully
- Negative POSIX status code upon failure

4.11.2. doca_regex_dequeue

This function dequeues any matches from a previously enqueued job.

```
int doca_regex_dequeue(struct doca_regex *regex, uint16_t qid,
                       struct doca_regex_job_response *responses,
                       uint8_t max_results);
```

Where:

- `regex [in]` – the DOCA RegEx instance
- `qid [in]` – the ID of the queue in which to dequeue the results data
- `responses [out]` –
- `max_results [in]` – maximum number of results to return. The responses array must have capacity for at least this many elements.

The function returns 0 or a positive integer representing the number of results dequeued, or a negative status code on failure.

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