CQA Project Report: Milestone–1

"Verification Challenges in Compression and Cryptographic Stacks in Quick Assist Technology"

Department of Computer Science and Engineering
IIT Kharagpur

May 2017

1 Introduction

1.1 RESEARCH OBJECTIVE, TIMELINE AND MILESTONE FOR FIRST YEAR

The broad research goal in this project is to develop a deep understanding of the nature of input data, configurations, or operating modes, or a combination of these three, that might cause functional failures or performance bottlenecks in the compression/decompression and encryption/decryption functionality provided by Intel *QuickAssist* accelerator chipset.

The proposed deliverable timeline is:

[0-6 Months]

1. Advertisement of research position, hiring of research staff, procuring equipment.

2. Familiarization with the theory, international standards (e.g. RFC 1951), open-source software implementations (if any) of common data compression algorithms (e.g. DEFLATE), and the syntax, semantics and usage scenarios of the corresponding QuickAssist API calls of common data encryption/decryption and compression/decompression algorithms.

[7-12 Months]

1. Creation of in-house analysis testbed software for the compression/decompression algorithms as made available by QuickAssist APIs, with easily tunable parameters.

2. Creation of in-house analysis software testbed for the encryption/decryption algorithms as made available by QuickAssist APIs, with easily tunable parameters.

Creation of in-house analysis testbed software for the compression/decompression involves creating test cases on top of the QAT driver, with possibility of varying the following parameters:

1. Different buffer sizes: e.g., 32 bytes, 128 bytes, 1K, 1K +1, 2K, etc.

2. Different buffer content: random data, data with specified patterns.

3. Concurrency: sending multiple jobs in-parallel to different QAT instances.

1.2 Milestone-1: Focus and Deliverables

- **Focus**

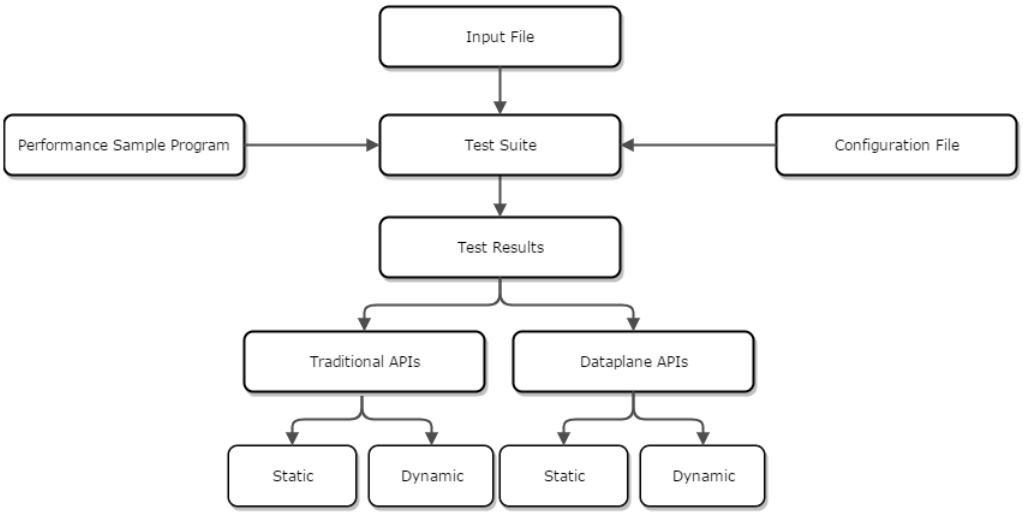

Figure 1: Test Suite Architecture.

1. Understaing the semantics and syntax of QAT APIs and analysis of tests on QAT driver

- **Deliverables**

1. A report on the above.
2. Analysis results of tests performed.

- **Duration:** 8 months.

1.3 Organization of the Report

The rest of the report is organized as follows. Section 2 presents the design and test results with quantitative comparison graphs. Section 3 presents some observations about the test results. Future work are listed in the Section 4, which concludes the report.

2 Design of Test Suite and Test Results

2.1 Design

Note that we have concentrated solely on compression/decompression functionalities in this phase of the work. Fig. 1 shows the architecture of the proposed test suite developed. We describe the working of the test suite developed with a running example. The test suite is designed to work in command line mode. The test suite script file gets the input variable

definitions from a configuration (ASCII text) file provided by the user. The test suite is also interactive because it requires the user to input a few parameters during execution. An example configuration file content is shown below:

```
[root@localhost script]# cat QatHwFnTst.cfg

## This file is used as input config file for testing
## of QAT Hardware functionality.
## The Nomenclature of variables defines their meaning.

## Buffer Size in K
QAT_HW_COMP_TST_BUF_SIZE=32
QAT_HW_CRYPTO_TST_BUF_SIZE=32

# Buffer Content
# 0 for Calgary Corpus Set, 1 for random, 2 for data with specific pattern
QAT_HW_COMP_TST_BUF_CONTENT=1
#QAT_HW_COMP_TST_BUF_CONTENT_SIZE=3251493
QAT_HW_CRYPTO_TST_BUF_CONTENT=0

## Does it support PARALLEL (0: disable, 1 enable)
QAT_HW_TST_PAR=1

## Number of instances should be used
QAT_HW_TST_NUM_INST=2
```

The test suite is executed by running a top-level shell script `test_suite.sh`:

```
[root@localhost script]# ./test_suite.sh
-----------------------------------------------
     TEST SUITE FOR QAT HW FUNCTIONALITY
-----------------------------------------------
1. Test with Specific Buffer Size and Buffer Content
2. Test with Specific Number of Instances
3. Analyze with varying buffer size for Specifc Input Data Type (Calgary Corpus)
4. Analyze with varying buffer size for Specifc Input Data Type (Random Corpus)
5. Analyze with all fixed size buffer with varying the size of Input Data
6. Exit
---------------------------------
Enter your choice [1-6] :
```

The sample test output for the Calgary Corpus (option 3) is available here: `https://git.io/v9Ary`. The log is used as intermediate input to plot the graphs. The following parameters were varied for the experiments:

1. Different buffer sizes: e.g., 64 bytes, 128 bytes, 1K, 1K+1, 2K, etc.
2. Different buffer content: random data, data with specified patterns.
3. Concurrency: multiple jobs are issued in-parallel to different QAT instances. **Note that our available chipset limits the number of parallel QAT instances to 2.**

2.2 Results for Buffer Size Variations under Different Modes/APIs

The aim is to enumerate the variation of compression ratio w.r.t. buffer size. The buffer size starts from 64 bytes and goes upto 65536 bytes (increasing by 2's exponent). Table 1 shows the abbreviations for the different degrees of freedom for which the experiments were performed. In reading the plot titles, the first letter indicates API type; the second letter indicates compression or decompression operation; the third letter indicates static or dynamic coding scheme, and finally the fourth letter is number which indicates the compression level (1 or 2). For example, a plot with a title **TCS1** indicates the experiment was carried out with traditional APIs, compression was performed, static coding was used at compression level 1.

T- Traditional APIs	D- Dataplane APIs
C- Compression	D- Decompression
S- Static	D- Dyanamic

Table 1: Abbreviations for different degrees of freedom.

2.2.1 Results for Calgary Corpus

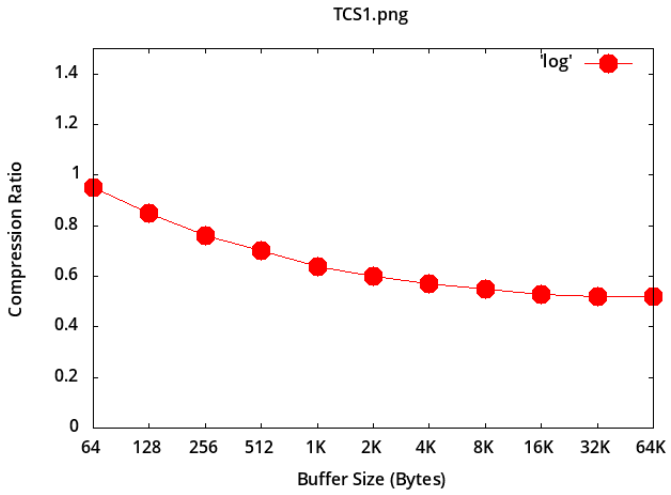

Figure 2: Compression ratio plot for TCS1 (Calgary Corpus).

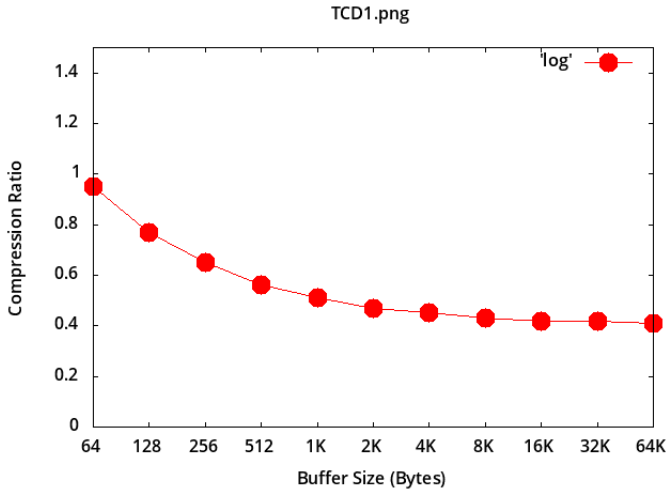

Figure 3: Compression ratio plot for TCD1 (Calgary Corpus).

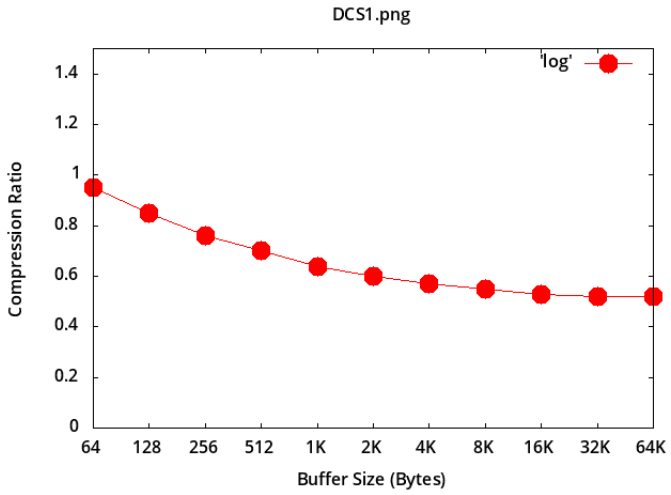

Figure 4: Compression ratio plot for DCS1 (Calgary Corpus).

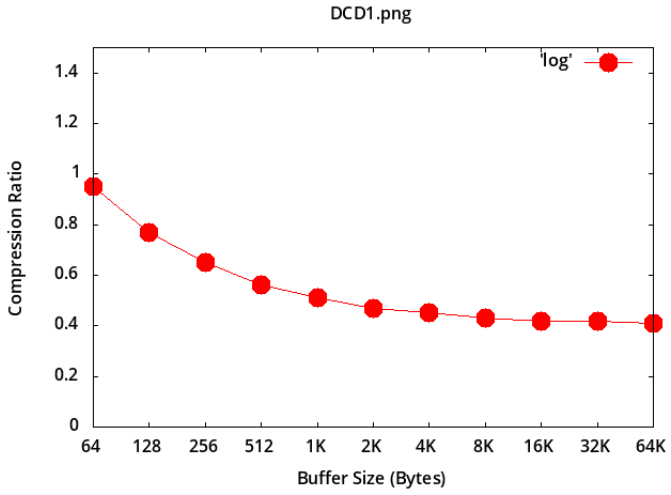

Figure 5: Compression ratio plot for DCD1 (Calgary Corpus).

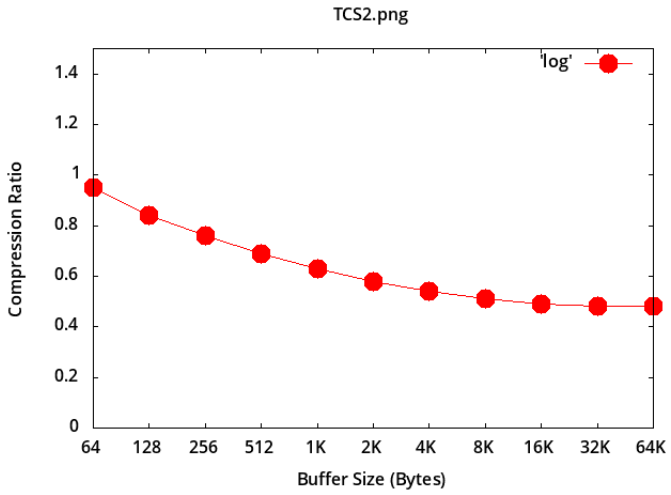

Figure 6: Compression ratio plot for TCS2 (Calgary Corpus).

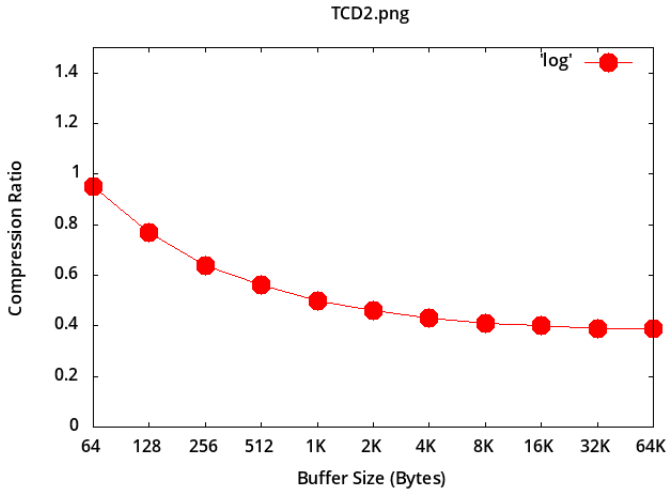

Figure 7: Compression ratio plot for TCD2 (Calgary Corpus).

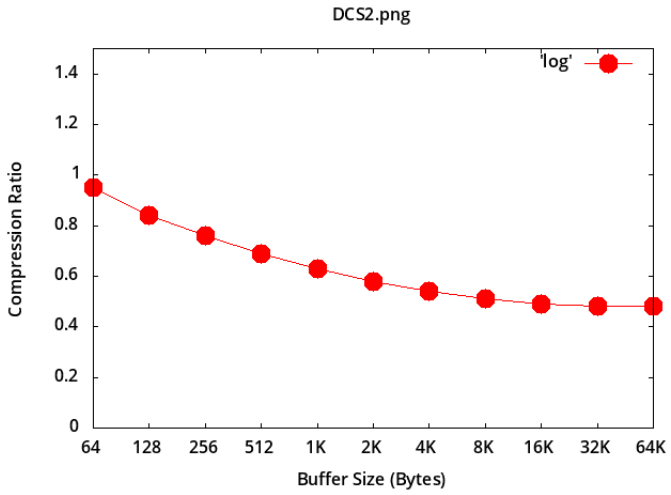

Figure 8: Compression ratio plot for DCS2 (Calgary Corpus).

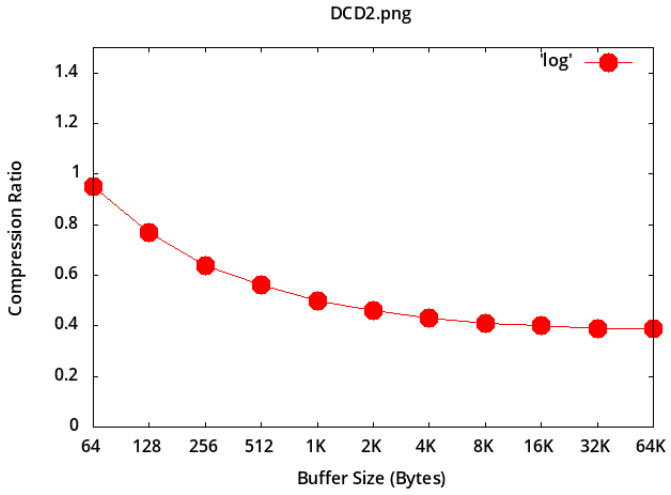

Figure 9: Compression ratio plot for DCD2 (Calgary Corpus).

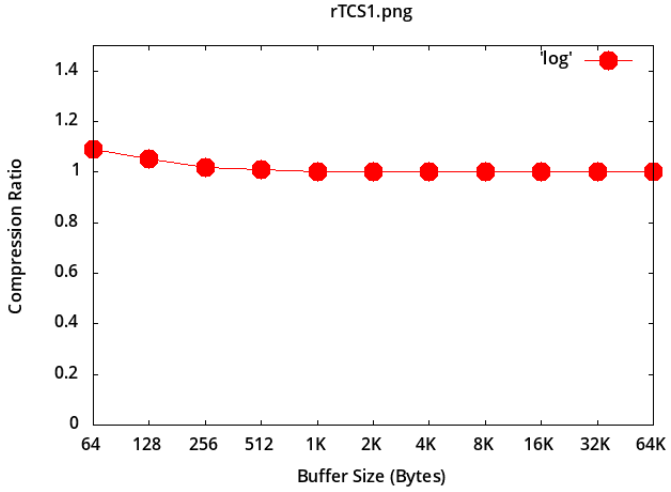

Figure 10: Compression ratio plot for TCS1 (Random Corpus).

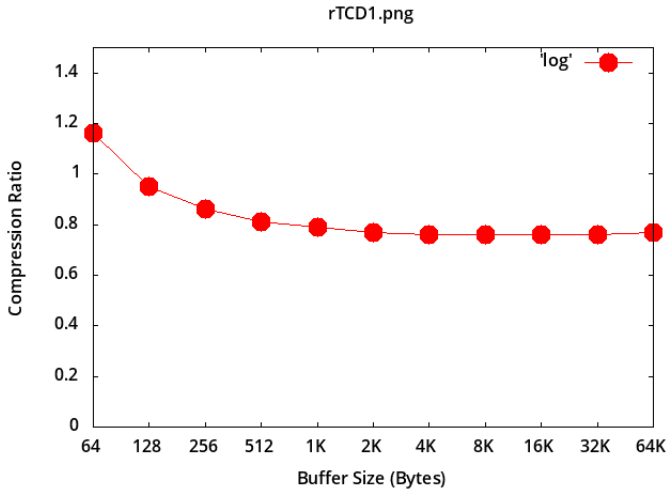

Figure 11: Compression ratio plot for TCD1 (Random Corpus).

2.2.2 Results for Random Corpus

The test input file considered here is a random ASCII file, created automatically. The size of file was kept the same as the Calgary Corpus. The file can be found here: `https://git.io/v9bVD`. Graphs from figure 10- 13 shows about this test.

The results for level 2 are similar to level 1.

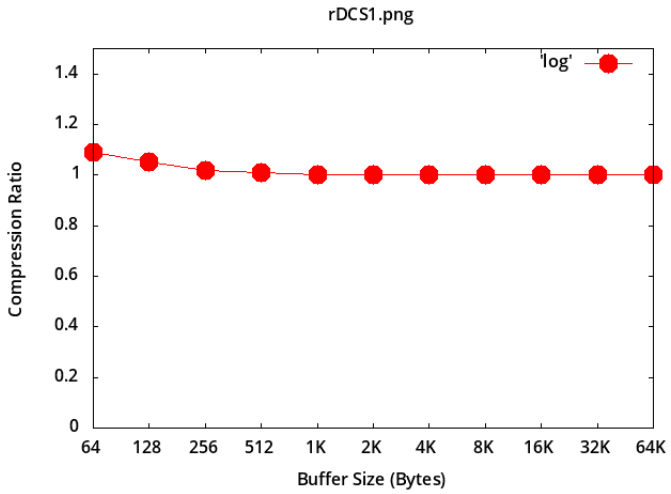

Figure 12: Compression ratio plot for DCS1 (Random Corpus).

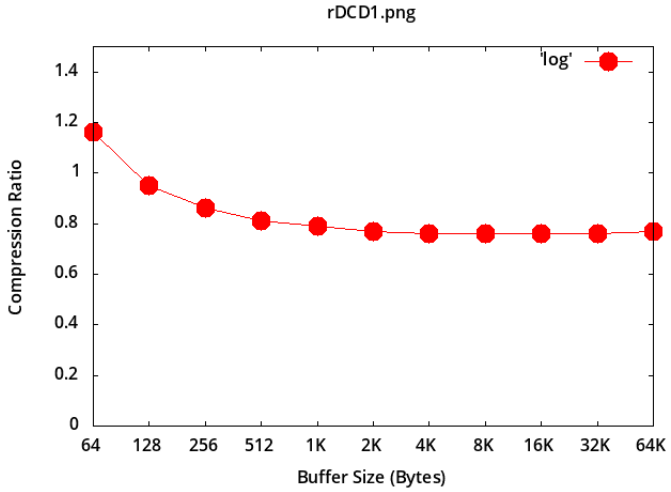

Figure 13: Compression ratio plot for DCD1 (Random Corpus).

3 Observations about Results

- **Calgary Corpus:** In the results, one can understand that as the buffer size increases, the compression ration decreases. The good amount of buffer size results in good compression which is quite intuitive. Also, there is not much difference in results when the compression level is changed. The main difference can be observed in static and dynamic types of coding.

- **Random Corpus:** The random corpus showed results which are counter-intuitive. Although for dynamic compression, as the buffer size increases, compression ratio decreases. But more interestingly it when the buffer size is less, the compression ratio goes beyond 1.0, i.e. effectively there is expansion of size! Hence, for this testcase, QuickAssist in not effective in practice.

4 Future Work

Next set of works that we will focus on at the next milestone are as follows:

- As we have observed, the input dataset changes the compression performance, trying with different source of files may hit different corners of compression. This also includes with different levels of compression.

- The work needs to carried out for cryptographic APIS. The combination of executing cryptographic and compression APIs would give more corner space to explore.

- More test cases has to be identified to put the hardware in stress mode to evaluate the performance, and to suggest necessary improvements.