INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC 1/SC 29/WG 11 M39179
Chengdu, China – October 2016

Source: **Dapcom, DMAG-UPC, Made of Genes, BSC**
Status: **Proposal**
Title: **A proposal for a compression algorithm based in FAPEC**
Response to the Joint Call for Proposals for Genomic Information Compression and Storage
Authors: **Jordi Portell, Francesc Julbe (DAPCOM Data Services),**
Jaime Delgado, Silvia Llorente (Distributed Multimedia Applications Group – Universitat Politècnica de Catalunya),
Leonor Frías, Oscar Flores (Made of Genes),
Josep Lluís Gelpí, Dmitry Repchevsky (Barcelona Supercomputing Center)

Index

1 Introduction

This document presents a proposal led by DAPCOM Data Services for the efficient compression of genomics data using FAPEC.

FAPEC is a high-performance data compression tool commercialized by DAPCOM [1], offering good compression ratios, very high speeds (including multi-threaded operation), resilience in front of data corruption, and embedded encryption.

This is a joint proposal of the Spanish companies DAPCOM Data Services [1] and Made of Genes [2], DMAG-UPC [3] and BSC [4]. It is based in the FAPEC technology, initially developed by DAPCOM for handling satellite massive data.

2 FAPEC Data Compressor

2.1 Overview

The Fully Adaptive Prediction Error Coder (FAPEC, [5]) is a new data compression solution based on a variety of configurable pre-processing stages combined with a high-performance entropy coder. The latter provides better compression resiliency in front of statistical outliers, typically leading to higher ratios than the CCSDS 121.0 (adaptive Rice) entropy coder. This compressor was initially devised for satellite payloads, and therefore its computing requirements are really modest (thus being very fast in on-ground computers).

The latest release available, FAPEC Core 2016.0 [6], includes a pre-processing stage aimed at ASCII data files (such as FastQ or SAM files used in Genomics), although the compression ratios achieved are typically lower than those given by Zip-based solutions. However, the next release being prepared now (FAPEC Archiver 2017.0, to be issued Q1 2017) includes a specific lossless pre-processing stage for FastQ files which gives an excellent performance both on ratios and speed. Compared to well-known solutions such as Zip, it provides similar ratios but at a much higher speed. A specific stage for SAM files is also being developed. Lossy compression options for these two formats (just for the quality scores) are being investigated as well.

FAPEC is not only an entropy coder, but also a high-performance and reliable data compression framework including multi-platform executables and libraries. By construction, it runs on "chunks" of data with a configurable size (from a few KB up to 384MB each), which allows a quick block-based access to the compressed file – without requiring any external 'index' file (contrary to the BAM format, for example). Specially, these chunks also provide an embedded resiliency to data corruption, allowing to decompress the file minimizing the data loss (but not completely avoiding it yet, at least for now: in the near future we will embed error-correcting codes). This is true either for truncated files or for data corruption in the middle of a file. Such error handling is done transparently to the user, whenever any error is found. Data encryption is also embedded, currently using 256-bit AES with symmetric keys. Multi-threading is also available (up to 32 threads), although the single-threaded execution is often fast enough even for the Genomics case (the bottleneck uses to be the hard disk or the network). FAPEC files also support the inclusion of license-enforced privacy, only allowing to decompress a given file with the same license that was used for its compression.

Current software implementation, in ANSI C for better multi-platform compatibility, has been successfully tested on Linux, Windows and Mac OS systems, as well as x86, PowerPC and ARM processors.

2.2 FAPEC compressed format

The FAPEC implementation is based on different layers, each of which provides a given compliance level to a specific FAPEC binary stream format definition. FAPEC supports compression on files or in streaming mode. It is also designed to allow easy integration in third-party software, for example to run on memory buffers. Figure 1 provides an overview of the main concepts involved in FAPEC, which are described below.

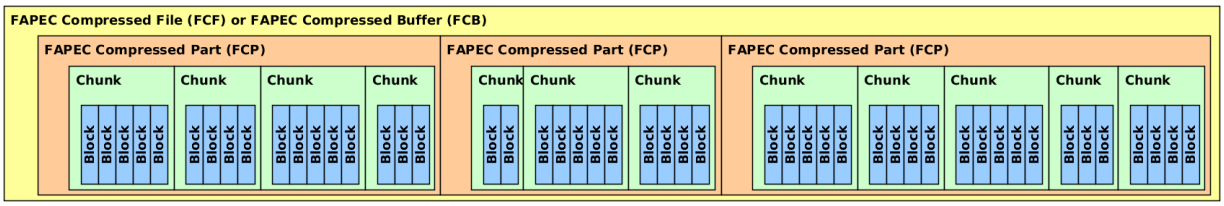

Figure 1 – Overview of the constituents of a FAPEC compressed stream

- One FAPEC compressed file or buffer, which in general we refer to as a "FAPEC compressed stream", can contain data from one or more files (or buffers).
- One "FAPEC compressed part" typically corresponds to one data file.
- Each of these parts is compressed in "chunks" (previously mentioned), each of these being from a few KB up to some hundred MB (configurable by the user).
- Chunks are the core elements of the FAPEC data compression concept. They contain the original data, pre-processed with the adequate algorithm (leading to "prediction errors") and afterwards compressed with the FAPEC entropy coder.
- The compressed chunk contents can then be encrypted with AES (using the OpenSSL EVP implementation).
- Each chunk can be decompressed independently from the others, thus isolating the effects of eventual data corruption.
- Finally, the "FAPEC blocks" just refer to the internal operation of FAPEC, which runs on small blocks of some hundred values (i.e. "prediction errors") to quickly adapt to statistical variations.

Each of these structures has optimized headers and footers (including checksums, separately for headers and data) in order to be self-described, including redundancy to provide error resiliency in case of data corruption.

The format of the chunks is very simple, just including a small header with a synchronization word (to recover from heavy data corruption), original and compressed data size, a few control flags and a checksum. The compressed contents are the FAPEC blocks as-is.

2.3 FAPEC implementation and usage

The FAPEC standalone executable software provides compression (and obviously decompression) of FastQ files, generating ".fapec" compressed files. In can be invoked either on a single FastQ file (thus generating a FAPEC compressed file with one single part), or on a set of FastQ files in one or more directories (leading to a FAPEC compressed file with several parts). Each part (i.e., each FastQ file) is automatically split in chunks.

The software provides several configuration options, including several pre-processing stages for images, text, binary data, etc. Obviously, here we are only interested in the FastQ case, for which we can provide options on the chunk size and on the activation of AES encryption, as well as the activation of multi-thread operation.

For the FastQ case, one can consider using the FAPEC executable as-is, which is a completely valid approach – and can be considered a very useful data compression standalone tool for FastQ files. However, the present proposal aims at a specific file format for genomic information, in which FAPEC will provide compression for its data elements. Therefore, the expected usage of FAPEC for this proposal is through its software API, which offers compression and decompression directly at a chunk level, thus avoiding the FAPEC-specific headers (files and parts) and relying on the MPEG-specific (genomics-specific) format definition. If necessary, the API also provides routines at a "FAPEC part" level (either on FastQ data loaded to a memory buffer, or invoking it to run on a disk file), in which case FAPEC will automatically do the necessary chunk-level operations.

2.4 Description of the FAPEC FastQ algorithm

As mentioned, the FAPEC approach is based on relatively small chunks of data to enforce error resiliency and minimize the computing and memory resources needed. Therefore, the FastQ algorithm cannot rely on file-wide correlations between nucleotide sequences (and of course inter-file correlations neither). Thus, the algorithm is relatively simple, just using some correlations between consecutive sequences, and specially relying on the kind of characters typically contained in a FastQ file.

2.4.1 Chunk preparation and format assessment

When invoked at a file level, FAPEC progressively loads the chunks of FastQ data taking care of not breaking any sequence between two consecutive chunks. This is done by simply checking the line feed and "@" (sequence header identifier) characters.

If invoked directly at a chunk level, the caller should also try to take care of this. However, FAPEC FastQ chunk compression can also handle broken sequences. If a chunk starts in the middle of the sequence, that is considered a "prelude" to the FastQ sequences, and compressed in a very simple manner – just coding the differences between consecutive characters. It will mean a small (typically negligible) decrease in the compression ratio, except in case of very long sequences combined with very small chunks: in that case, the decrease can be significant.

Once chunk compression starts, the very first step is to check the correctness of the data format:

- We search the "@" identifier after a line feed,
- then we determine the length of the first nucleotides sequence (by looking for the "+" character at the start of the next line),
- and finally we assess that the quality scores line for that first sequence has the same length than the nucleotides line.
- The newline format (DOS-like or UNIX-like) is also determined at this stage.

If any of these checks fails, FAPEC falls back to a simple text-like lossless compression for that chunk, leading to a moderate compression factor (barely reaching 50%) but ensuring no data loss.

If all checks are successful, FAPEC proceeds to compress that chunk using the FastQ algorithm. First of all, it outputs an additional small header indicating the newline format and the "prelude" length (i.e., the number of characters found prior to the start of the first complete sequence). Then, it indicates the length of the "sync stream" (see hereafter), which is needed to know the position in the compressed chunk where the FAPEC blocks in themselves start. Figure 2 illustrates this FAPEC compressed FastQ chunk format.

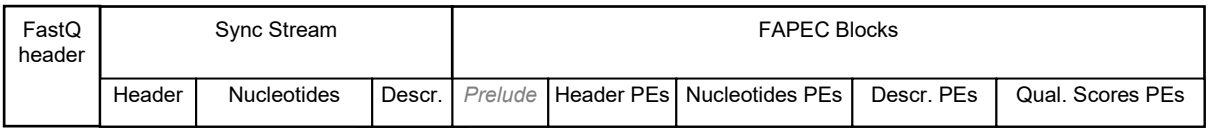

Figure 2 – Overview of the FAPEC FastQ compressed chunk format

In general, the approach for the several elements of a sequence is similar: We try to predict the next character to be compressed. If we succeed, we just generate a small code (typically 1 to 3 bits) and output that in the so-called "sync stream". Otherwise, we generate an "escape code" in the sync stream plus the prediction error (PE) – which is coded with FAPEC and thus output in the FAPEC blocks.

2.4.2 Sequence header

The header of the very first sequence in each chunk is compressed differentially. That is: if a given character is identical to the previous one, we just output one bit (set to 0) in the sync stream. Otherwise, we output the bit set to 1, and accumulate the prediction error for later coding.

The headers of the next sequences are also coded differentially, but with respect to the corresponding character position of the previous sequence header.

2.4.3 Nucleotides

Nucleotides are compressed line by line without using the previous sequences at all.

The first character is coded in 3 bits (into the sync stream) if that is A, C, G or T. Otherwise, its numerical (ASCII) difference with respect to 'N' is coded as a prediction error with FAPEC.

The next characters are coded as follows:

- If we find a consecutive repetition we just generate one '0' bit into the sync stream.
- Characters A, C, G and T are coded in 3 or 4 bits into the sync stream, depending on the value of the previous character as indicated in Table 1 (where "N/A" indicates a character repetition, which is coded in just 1 bit as previously indicated, and 'x' means "any character"). As can be seen, we just need 4 bits if the previous character was not one of these typical A, C, G or T values. Therefore, we can typically expect compression ratios of at least 2.66 (factors better than 37%) for nucleotide sequences – even better (up to 8, or 12.5%, in the most extreme case) depending on the number of repeated characters.

AA	N/A	CA	100	GA	100	TA	100	xA	100
AC	100	CC	N/A	GC	101	TC	101	xC	101
AG	101	CG	101	GG	N/A	TG	110	xG	110
AT	110	CT	110	GT	110	TT	N/A	xT	1110

Table 1 – Nucleotide coding depending on the previous character

- If we find the otherwise typical character ‘N’ after one of the typical characters (A, C, G or T), we code it in 4 bits (“1110”).
- The newline is coded in 5 bits (“11110”).
- Finally, any other character will be indicated with 5 bits in the sync stream (“11111”) and its numerical difference with respect to the previous character will be coded with FAPEC. Note, therefore, that in practice it will probably mean a very low compression ratio or even an expansion. Thus, FastQ files with many “rare” nucleotide characters will lead to poor compression ratios.

2.4.4 Sequence description

Sequence descriptions are coded in the same way as sequence headers.

2.4.5 Quality scores

Quality scores compression completely rely on the FAPEC entropy coding: we work on a sequence-by-sequence basis (independently from the others), simply determining the numerical differences between consecutive characters and coding them with FAPEC.

2.4.6 Final coding

Once we have finished generating all the necessary bits and codes in the “sync stream” of the chunk being compressed, we add the necessary padding bits (up to 7) to get byte-alignment, and determine the number of bytes written so far. Those are then stored in the FastQ header previously mentioned, which will tell the decoder how to handle this chunk – that is, from which byte onwards we have to invoke the FAPEC entropy decoder.

Note that this approach means that FAPEC must work in memory buffers to compress and decompress each chunk. That is, we cannot perform a strict “streaming compression” (on a word-by-word basis so to speak), but that can be done by simply using small buffers. The minimum recommended size for a chunk is 4 KB, but to better handle genomic information with very long sequences we recommend at least 64 KB. For a typical file-based operation we suggest chunks of 1 to 64 MB, which will provide a good compromise between compression efficiency, error resiliency and fast quasi-random access to some given chunk.

2.5 Compressed data access

The FAPEC format and its chunks do not really include an explicit index of its contents. Instead, we provide headers that indicate the original (and compressed) size of each chunk. From that, one can quickly jump through the several compressed chunks (without having to decompress them) until reaching the data of interest. In binary data this approach can be very useful, but in FastQ it could be improved. We are planning to include an additional header field in each FastQ chunk indicating the number of sequences contained there, as well as the first sequence identifier. Further improvements or alternative header contents can be included if necessary.

3 Use cases for fast compression/decompression

There are different situations where compression ratio is not as relevant as compression / decompression speed. To illustrate them, we propose the following use cases, which take into account both compression ratio and speed when dealing with genomic information.

The use cases are divided according to different combinations of the following characteristics:

- High compression ratio
- Fast compression
- Fast decompression: in particular, providing either/and
    - Stream Access
    - Indexed Access

3.1 High compression ratio use case

The High compression ratio is required for Long term storage (archive). In this case data is rarely used, so the main requirement is to save space. In this case, maximum compression ratio is desirable.

3.2 Fast compression + Fast decompression (streaming) use case

The fast compression / decompression or streaming is indicated for temporal files of a given pipeline. That is, a pipeline generates typically several files in the processing which, once the final files are generated, can be thrown away. However, due to several limitations, they need to be actual files:

- Some of the steps are computationally expensive and loss of the results on failure is costly.
- Pure streaming access might be beneficial for several applications, but still many applications need an actual file, in particular because the file needs to be read several times.

In this case, maximum compression speed is desirable.

3.3 Fast decompression with indexed access use case

This use case is foreseen for visualization tools working in real time, e.g., a genome browser, where there is a need of a fast index access and decompression to show the requested information.

3.4 Fast decompression with indexed access + fast compression with streaming access

This use case is foreseen for real-time generated data subsets to be transmitted to a client application. In this case, the source file(s) might be a long term storage and the destination file might be just a stream to be consumed a.s.a.p. or transmitted over a network.

4 Covered MPEG format requirements

FAPEC has the following general features and fulfils the following general requirements:

- Only FastQ files (raw reads) are supported for now.
- SAM files (aligned/mapped reads) are also supported, but currently with a worse performance (mainly regarding ratios) than Zip. That is being improved (a SAM-specific stage is being implemented in FAPEC).

- Only lossless compression is supported (lossy compression of quality scores will be implemented in the near future).
- Additional features:
    - Multiple files can be compressed and combined into a single FAPEC file.
    - Non-sequential access to compressed FastQ files is supported: direct access to a given raw file contained in a compressed FAPEC file, and quick access to a given data chunk from a compressed file.
    - Automatic recovery of corrupted compressed files: decoding errors are limited to the affected chunk (typ. 1MB), and after that the rest of the file is decoded without errors. Corrupted file headers and footers (up to a few tens of bytes) are typically recovered without any error.
    - Embedded data encryption using OpenSSL AES (256-bit symmetric key).

Following MPEG formalities, the following requirements are fulfilled:

Req ID	Requirement for compression of unmapped reads	Compliance (Yes/No)
1.1	The solution shall support lossless compression of reads headers.	Y
1.2	The solution shall support lossy compression of reads headers by preserving at least a unique read identifier and pairing information when available.	N
1.3	The solution shall support preservation of pairing information	Y
1.4	The solution shall support lossless compression of nucleotides sequences supporting a minimum of 5 symbols (A, C, G, T, N)	Y
1.5	The solution shall support lossless compression of quality scores.	Y
1.6	The solution shall support lossy compression of quality scores.	N
1.7.1	The solution shall structure compressed data so that parallel processing is enabled and compression efficiency is preserved	Y
1.7.2	The solution should structure compressed data so that efficient querying of data is enabled and compression efficiency is preserved	Y
1.8	The solution shall preserve the association among headers, nucleotides and quality scores	Y

5 Test results

The main test campaign has been executed at the Barcelona Supercomputing Centre (BSC), where the data inputs indicated by MPEG have been compressed by FAPEC, then decompressed, and finally their integrity (lossless operation) assessed.

Additionally, performance (speed) tests have been carried out in a local workstation of DAPCOM, based on an Intel Xeon E5-2630 at 2.4GHz with 16GB RAM and RAID5 SATA disks (running Linux CentOS 7 64-bit). Due to the slow writing operation of such RAID5 disk (much slower than the FAPEC speed), we have considered only the “User” (CPU) time of

compression and decompression, and thus I/O time is excluded. From that, combined with the size of the raw data, we have determined the equivalent throughput as MB/s (understood as: input data volume compressed/read per second, or output data volume decompressed/written per second).

FAPEC 2016.1 Beta r717 (64-bit) has been used in these tests, using a single-thread configuration and the default chunk size of 1MB. No parameter adjustment is needed to compress these files (only the "`-dtype fastq`" option).

Please note that both the FastQ algorithm and its implementation in FAPEC are still prototypes, and thus we can expect improvements both in ratios and speeds in coming versions (expected Q1 2017). Also, a specific algorithm for SAM is not available yet (also planned for Q1 2017).

Figure 3 shows the Compression Factor (CF) obtained by FAPEC on the MPEG dataset for the different input types. As can be seen, the best CF is 27.7% (for Id 20), whereas the worst CF is 42.6% (for Id 08). In general, the CF obtained by FAPEC is very similar to that achieved by Gzip.

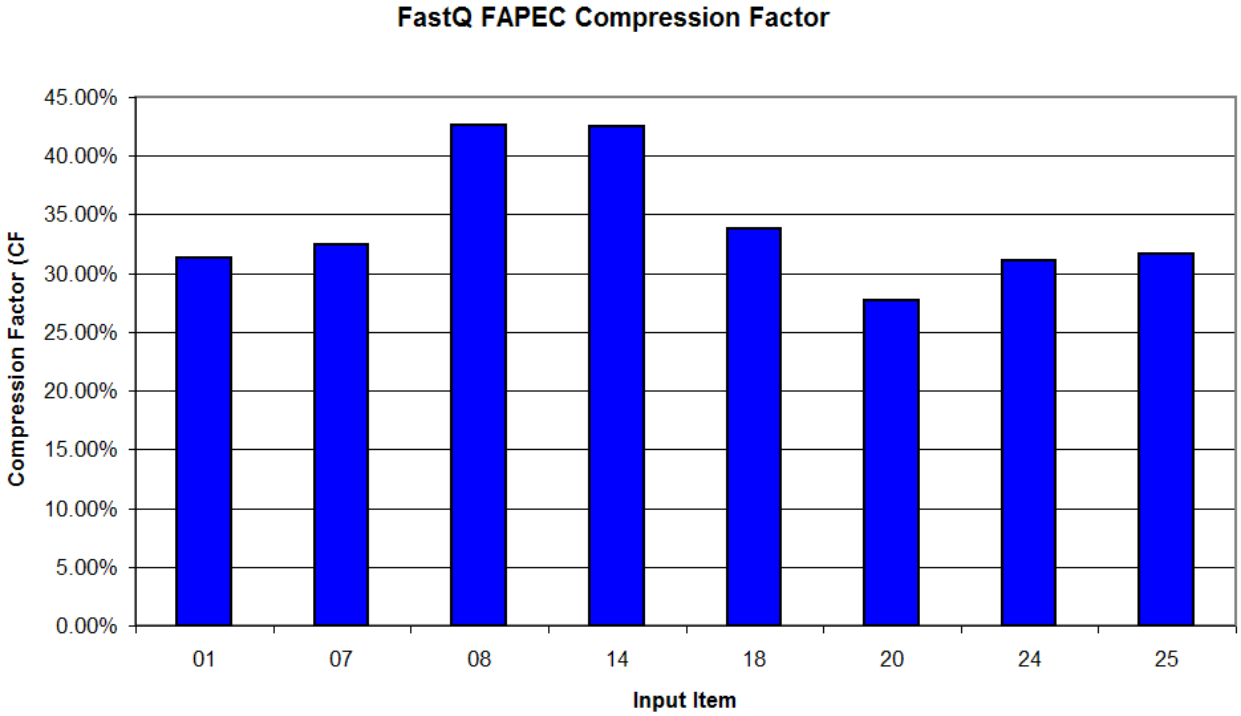

Figure 3 – FAPEC-FastQ compression factors

On the other hand, Figure 4 shows the speeds achieved by FAPEC on the mentioned environment (the DAPCOM workstation). Results for data inputs 07 and 24 are not available due to the lack of testing time. As we can see, even with a single thread FAPEC is able to process more than 60MB/s in an otherwise typical processor, with peaks reaching 100MB/s. When activating its multi-threading capability, FAPEC scales almost linearly – assuming that the disk I/O can cope with that.

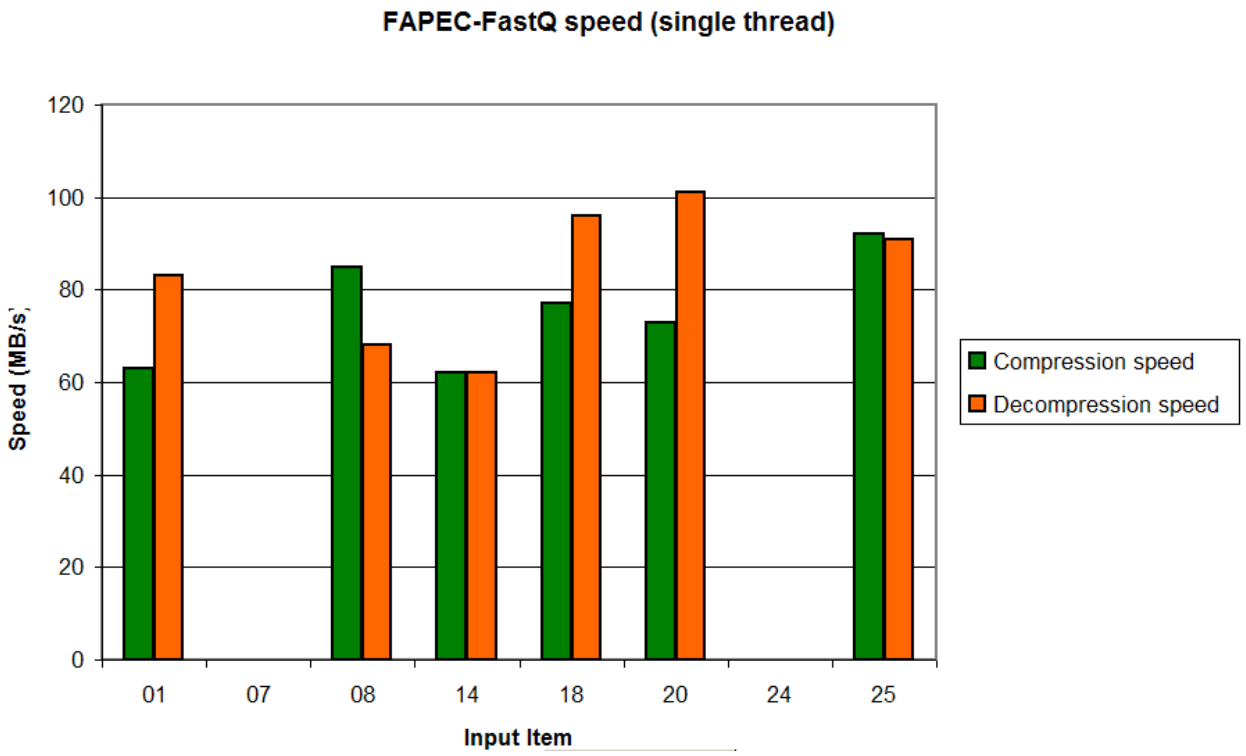

Figure 3 – FAPEC-FastQ compression factors

All these results are also available in the attached Excel file (m39179_FAPEC_ResultsSpreadsheet.xls).

Regarding test case 1.4 (to check that compressed data is structured, allowing to process data in parallel, and allowing to efficiently query the data): As previously described, FAPEC, by construction, generates compressed data in "chunks" of around 1MB (in raw data size). They are output following their original order in the raw file, and each of these chunks includes headers with the raw and compressed data sizes. Finally, each of these chunks can be decompressed independently from the others. Therefore, parallel handling of FAPEC files is indeed possible, and an efficient querying can also be designed based on this.

Finally, FAPEC also fulfils some Transport Requirements:

- Req. 3.1 is fulfilled thanks to the inclusion of checksums in the compressed files. One 32-bit checksum is available per chunk, and additionally FAPEC performs consistency checks during decompression. Any data corruption triggers a warning message to the user. Note that it still allows decompressing the rest of the file if just one or few chunks are corrupted. Also the global headers of FAPEC files include enough redundancy to detect corruption (and in some cases recover from it).
- Req. 3.2 is fulfilled thanks to the OpenSSL-based 256-bit AES encryption. Each chunk is encrypted independently.
- Req. 3.5 is fulfilled thanks to the moderate size of chunks, which can be as small as ~64KB without significantly affecting compression factors neither speeds. Such sizes allow real-time compression and decompression through a data stream such as a network. FAPEC can proceed with decompression from the very first chunk without having to wait for the others.

To summarize, FAPEC FastQ compression offers reasonable compression factors (very similar to those obtained with Gzip) but offers several appealing advantages: high speed, embedded error detection and recovery, encryption, and streaming.

6 Acknowledgements

The work done in this proposal has been partially supported by the Spanish Government under the project: Secure Genomic Information Compression (GenCom, TEC2015-67774-C2-1-R and TEC2015-67774-C2-2-R) and by the ESA Business Incubation Program through Barcelona Activa.

7 References

[1] DAPCOM Data services, http://www.dapcom.es/
[2] Made of Genes, http://madeofgenes.com
[3] DMAG-UPC, http://dmag.ac.upc.edu
[4] BSC, http://www.bsc.es/
[5] Portell, J., Villafranca, A.G., García-Berro, E. Quick outlier-resilient entropy coder for space missions, Journal of Applied Remote Sensing (2010) 4, 041784, SPIE
[6] DAPCOM Data Services, FAPEC latest published version
http://www.dapcom.es/fapec_core_2016_0.html