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
MPEG2017/m39948
January 2017, Geneva, Switzerland

Source: **Dapcom, DMAG-UPC**
Status: **Proposal**
Title: **FAPEC v.2 for Core Experiments on 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),

Contents

1 Introduction

This document provides updates on the proposal led by DAPCOM Data Services for the efficient compression of genomics data using FAPEC.

This document describes the changes done to the FAPEC data compression algorithm for FastQ files with respect to the initial description presented in Chengdu (China), m39179 [1]. This document describes FAPEC v.2 used for the Core Experiments identified in [2]. The specific FAPEC v.2 experiment is described in [3].

The mentioned document [1], with the overall description of FAPEC, is still applicable except Sections 2.4, 2.5 and obviously the results presented in Section 5. The overall benefits of FAPEC (speed, error resiliency, encryption, multi-thread operation), as well as the general requirements fulfillment indicated in [1], are still applicable.

These changes in the compression tool were motivated by the modest ratios achieved in the first version, and by the need of determining individual compression ratios for each of the genomic information constituents (identifiers, reads and quality values mainly).

2 Updates to the FAPEC Data Compressor FastQ algorithm and format

2.1 Compressed bitstream format

As indicated in [1], the overall structure of a FAPEC compressed file is based on some general headers followed by *chunks* of compressed data, each also with its corresponding header to allow a fast access to their contents.

Each of these chunks, for which we recommend a size of 1 MB to 16 MB for the FastQ case, is compressed independently from the others to enforce error resiliency.
Therefore, we should see a chunk as the basic container of compressed data (either for FastQ or other types of data).

It also allows for memory-based block compression; that is, invoking FAPEC as a dynamic library from third party software, instead of an executable.

The following is the format of a compressed FastQ chunk:

- Header:
    - 1 bit that should be set to ‘1’, indicating that the chunk was indeed compressed with the FastQ algorithm.
      If this bit is ‘0’ it means that the raw chunk doesn’t follow the FastQ syntax, and thus it was compressed as plain ASCII. In such case, only 7 additional padding bits (set to 0) will be found in the header, followed by the ASCII-based compressed data, and without any footer.
    - 1 bit with newline coding: ‘0’ for UNIX-like, ‘1’ for DOS-like.
    - 2 bits with Sequence Reads (nucleotides) coding: “00” for “best case” (only A/C/G/T and linefeed), “01” for “good case” (only A/C/G/T/N and linefeed), or “10” for “compatibility case” (any char is possible). “11” is forbidden for now.
    - 23 bits with the number of sequence reads (i.e. sets of 4 text lines) in this chunk.

    - 1 bit indicating if there's "prelude" ('1') or not ('0'). Reminder: we consider (and support) the possibility of having non-FastQ data at the beginning of a chunk, in case e.g. the file reading and chunking was not correctly aligned with reads.
    - If there's no prelude, 4 spare bits set to 0. Otherwise, 20 bits with the raw prelude size in bytes (thus supporting up to 1MB; if a larger prelude is found before reaching FastQ-formatted data, we fallback to simple ASCII compression for the whole chunk).
    - Overall size of header: 1 byte (not FastQ), or 4 bytes (no prelude), or 6 bytes (prelude).
- Compressed data sections, each with the necessary spare bits at the end (to start the next section at the next byte alignment):
    - Prelude (if present).
    - Headers (read identifiers).
    - Nucleotides (reads).
    - Quality scores (or quality values).
    - Descriptions.
- Footer:
    - 3 bits indicating the coding option ('0' differential, '1' dictionary-like) used in the Prelude, Headers and Descriptions.
    - 5 bits reserved for future extensions.
    - If there's prelude, 20 bits indicating its compressed size in bytes. Otherwise, 4 spare bits.
    - 28 bits with the Headers compressed size in bytes.
    - 28 bits, ditto for Nucleotides.
    - 28 bits, ditto for Quality scores.

Note that, thanks to this structure, FAPEC allows to directly access any specific section. It can be achieved by using the compressed sizes information available in the footer.

It also allows decompressing only a given specific section – either from just a chunk or from the complete file. For example, we can just decompress all the reads and nothing else, or just the headers and reads.

Such partial extraction can also be done in compressed form. That is, another compressed file can be generated containing only identifiers, or only quality scores, etc. This operation should be extremely quick, as no actual decompression and re-compression would take place.

The bitstream format also allows for partial compression; that is, storing (compressing) only some elements of a FastQ file.

Note that some of these selective access operations may be slightly slower in case preludes are present, but that's not the case in none of the tests done so far. That is, FAPEC perfectly aligns chunks with sequences.

These headers and footers also allow retrieving very quickly the total number of sequences available in a compressed file, as well as their compressed sizes.

2.2 FAPEC FastQ algorithm

For each of the chunks to be compressed, a first format assessment is done as indicated in Sect. 2.4.1 of [1].

Afterwards, we read all the sequences and accumulate them into 'partial' buffers (separate for the prelude, headers, nucleotides, descriptions and quality scores).

Then we do the FastQ compression in itself, as described hereafter.

The 'prelude', if present at all, is compressed with the dictionary-like pre-processing algorithm available in FAPEC, followed by the entropy coding algorithm of the FAPEC kernel.

2.2.1 Read identifiers

We code the identifiers "transversally", that is:

First we output the differences between the 1st character of consecutive lines; then, differences in the 2nd character of consecutive lines, and so on until we reach the longest line in the chunk.

For shorter lines we obviously stop outputting any value as soon as we reach the linefeed.

The resulting differences are entropy-coded with the FAPEC kernel.

Once done, we also run our dictionary-like algorithm on the read identifiers as-is. If the result is smaller than the "transversal" algorithm, we select it instead (and flag it adequately in the footer of the chunk).

The dictionary-like option appears to perform better in some cases with quite irregular identifiers.

2.2.2 Reads

Strictly speaking, here sequence reads are not actually "compressed" but just "recoded". That is, we use a reference-free algorithm, not aligning nor differentially coding the reads with respect to any reference genome. It leads to much lower ratios than other algorithms, but the execution speed is much faster. In terms of coding efficiency, ratios are 3.6 to 3.8 (either with 'N' characters or not), which is 90% to 95% of the theoretical limit.

Note that in this case we use a direct, specific coding algorithm (not using the FAPEC entropy coding kernel).

The algorithm is the following:

- When loading reads into their buffer we determine if we have a "best", "good" or "compatibility" case (see the Header description in Sect. 2.1).
- For optimization purposes, we use a Look-Up Table (LUT) to transform A/C/G/T/N and linefeed into values 0 to 5. It also transforms the rest of capital letters (until Z and also including blank space, "*" and "-") to allow coding them with just 5 bits in case of untypical nucleotide characters. The rest of chars are left as-is.
- We output, in 28 bits, the total number of nucleotide chars (including linefeeds).
- We then start processing in blocks of 7 to 12 chars (not passing through any entropy coder afterwards, as previously noted):

- If next 10 chars are all A/C/G/T, we output a '0' bit followed by 10×2 bits (2 bits per nucleotide), thus averaging 2.1 bits per char (ratio 3.81). Otherwise:
- If we are in the "best" case, it means it can only be some linefeed; then, we output one bit set to '1' followed by 28 bits with the A/C/G/T/linefeed combinations, creating a base-5 code from the individual chars. That averages 2.4 bits per char (ratio 3.31).
- On the other hand, if we are in the "good" case and next 12 chars are A/C/G/T/N, we output two bits set to "10" followed by a 28-bit base-5 code, as in the previous case (averaging 2.5 bits per char, ratio 3.2).
- If, still in the "good" case, we have some linefeed, we output two bits set to "11" followed by a 26-bit base-6 code with the next 10 chars, averaging 2.8 bits per char (ratio 2.85).
- Otherwise, if we are in the "compatibility" case (rare situations):
  - If next 10 chars are still A/C/G/T/N/linefeed, we output bits "10" and a 26-bit base-6 code (2.8 bits per char, ratio 2.85).
  - If next 11 chars are still "compressible" (caps, space, etc.), we output bits "110" followed by a 55-bit code (11×5 bits), meaning 5.27 bits per char (ratio 1.51).
  - Finally, if we have other exotic chars, we output bits "111" followed by the next 7 original 8-bit chars (56 bits) as-is, meaning 8.43 bits per char (ratio 0.95).

2.2.3 Quality Values

This algorithm is the simplest one, using a simple differential coding followed by the FAPEC entropy coding kernel.

For the first line we code the differences between consecutive characters.

For subsequent lines, the first 14 chars are coded vs. the same char of previous line (that seems to provide a significant improvement in Illumina data, whereas other data is barely affected by this). For the next chars of the line we simply code the consecutive differences.

2.2.4 Sequence description

Here we use exactly the same algorithm as in the read identifiers.

3 Acknowledgements

The presented work 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.

4 References

[1] A proposal for a compression algorithm based in FAPEC, ISO/IEC JTC 1/SC 29/WG 11 M39179 Chengdu, October 2016.
[2] ISO/IEC JTC1 SC 29/WG 11 N16526 - ISO/TC 276/WG 5 /N120, Core Experiments on Genomic Information Representation, Chengdu, October 2016.
[3] Description of FAPEC v.2 Core Experiments execution and results on Genomic Information Compression and Storage, ISO/IEC JTC 1/SC 29/WG 11 M39950, Geneva, January 2017.