Parallel decompression of gzip-compressed files
and random access to DNA sequences

Rayan Chikhi, joint work with Maël Kerbiriou

Institut Pasteur, France
Inria, University of Lille, France
Not so long ago, in a French research center.
Motivation

Please write a C++ FASTQ parser

Parallel decompression of gzip-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Motivation

Please write a C++ FASTQ parser

On it

Parallel decompression of \texttt{gzip}-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Motivation

Please write a C++ FASTQ parser

It needs to be faster than KMC’s
Motivation

Please write a C++ FASTQ parser

It needs to be faster than KMC’s

So please find a way to decompress gzip files in parallel

Parallel decompression of gzip-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Motivation

Please write a C++ FASTQ parser

It needs to be faster than KMC’s

So please find a way to decompress gzip files in parallel

That’s impossible
Motivation

Please write a C++ FASTQ parser

On it

It needs to be faster than KMC’s

So please find a way to decompress gzip files in parallel

That’s impossible

because..

Parallel decompression of gzip-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
... of the nature of `gzip` files.

But first:

- `gzip` is the software (v0.1 in 1992)
- `zlib` is the library
- **DEFLATE** is the algorithm (1989)

**DEFLATE** compression has essentially two components:

- LZ77
- Huffman coding
Huffman coding (1952)

- Frequent symbols are encoded into fewer bits
- No code word is a prefix of another

No need to understand Huffman coding to follow this talk.

Parallel decompression of \texttt{gzip}-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Lempel-Ziv 1977 (DEFLATE variant)

Encodes a string as a sequence of either:

- a literal (raw character), or
- a back-reference to a previous substring

Original text: `abcdeabcfbc`

LZ77 encoding:

```
literals    back-references
abcde [-5, 3] f [-3, 2]
```

-32KB ≤ offset ≤ 0 (i.e. sliding window)
• Text files everywhere
• `gzip` is a natural choice:
  • ASCII: > 12% space reduction (every 8th bit is always 0)
  • FASTQ: encodes ACTG’s in ≈2 bits/character
• Not best in class in neither speed nor ratio (see brotli, Zstd)
• However, ubiquitous and fast (note: `gzip -1` is 7x faster than default)
• Default compression format of `bcl2fastq` (Illumina)
gzip decompression speed

- `gunzip` decompresses at 30 MB/s (input read speed)
- Hard drives read at 100 – 200 MB/s
- SSDs read at 0.5 – 3 GB/s

→ reading compressed data can slow down programs, up to 30x
**Parallelism in gzip**

**pigz** only does parallel *compression* because parallel decompression is difficult.

Anatomy of a `.gz` file:

```
gzip header | block header | dynamic Huffman tree definition | Compressed stream | next block | ...  
```

Header  Block 1  Block 2

**Obstacles:**

- Positions of **start of blocks** are unknown
- Blocks contain **back-references** to previous blocks
Common solutions

zran.c / BGZF / BAM

- Keep an index of block start positions
- Avoid back-references across blocks

A fine solution, but:

- Not a widely deployed format
- < 50% of the SRA had indexed `.fastq.gz`'s in 2018
- Although new `bcl2fastq` indexes by default
- Slightly worse compression ratio
Our work

In regular `.fastq.gz` files:

- Quickly **guess** block positions
- See if we can perform random access
Our work

In regular .fastq.gz files:

• Quickly **guess** block positions
• See if we can perform random access
• We could, but only at low compression levels
• Failed to do it reliably at higher levels
• Found another way to do parallel decompression, without random access
Suppose you start decompressing at the middle of a LZ77 stream.

- Some characters can be decoded (those encoded as-is by LZ77).
- But back-references to positions before the start remain unknown.
Random access in compressed FASTQs

Decompression of FASTQ starting at the beginning of a block, without context.

Parallel decompression of gzip-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Random access in compressed FASTQs

Decompression of FASTQ starting at the beginning of a block, without context.

Parallel decompression of gzip-compressed files and random access to DNA sequences
M. Kerbiriou & R. Chikhi
Random access in compressed FASTQs

At normal compression level, after a random access, sequences can be fully determined after a while ($2^{20}$ bytes), but not headers.
We designed 2-step parallel decompression algorithm.

1. Partition file, decompress chunks independently, record unresolved back-references
We designed 2-step parallel decompression algorithm.

1. Partition file, decompress chunks independently, record unresolved back-references
2. Propagate context across chunks, resolve back-references
Parallel decompression of \texttt{gzip}-compressed files and random access to DNA sequences

M. Kerbiriou & R. Chikhi
Time for a demo?
Summary

In the paper (on Github):

- We study random access to .fastq.gz files
- Probabilistic model of compression
- Description of a general parallel decompression algorithm
- Implementation for ASCII files
- up to 700 MB/sec decompression (20x speedup over gzip)

Open questions:

- In FASTQs, one could possibly guess LZ-contexts (hard)
- Whether a .gz.index file would be useful in bioinfo (like .bai files in BAMs)
github.com/Piezoid/pugz