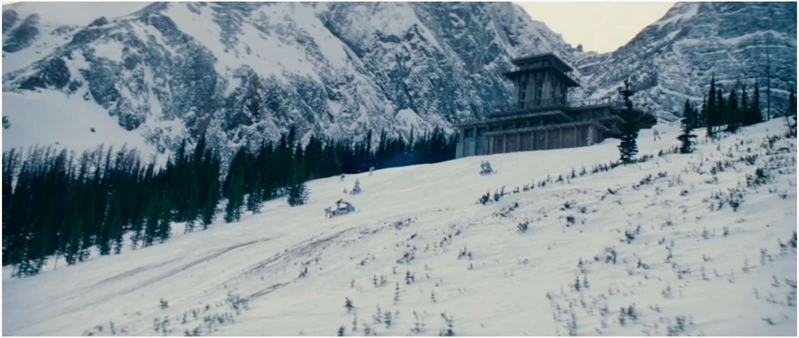
Parallel decompression of `gzip`-compressed files

and random access to DNA sequences

Rayan Chikhi, *joint work with Maël Kerbiriou*

Institut Pasteur, France
Inria, University of Lille, France

Motivation



Not so long ago, in a French research center..

Motivation

Please write a C++ FASTQ parser

Motivation

Please write a C++ FASTQ parser

On it

Motivation

Please write a C++ FASTQ parser

On it

It needs to be faster than KMC’s

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

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

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..

gzip format

. . . 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)

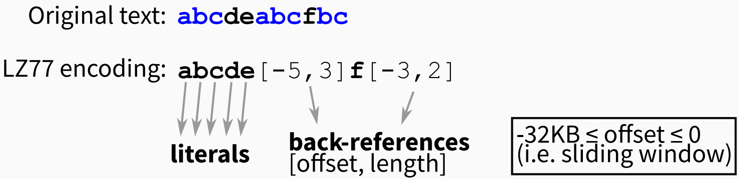
- Frequent symbols are encoded into fewer bits
- No code word is a prefix of another

No need to understand Huffman coding to follow this talk.

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: **abcde**[-5,3]**f**[-3,2]

literals

back-references
[offset, length]

-32KB ≤ offset ≤ 0
(i.e. sliding window)

`gzip` in bioinformatics

- 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

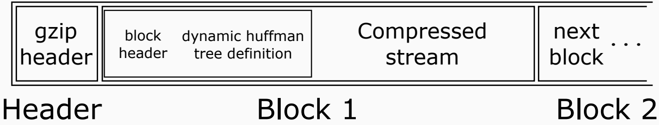
- `gunzip` decompresses at 30 MB/s (input read speed)
- Hard drives read at 100 – 200 MB/s
- SSDs read at 0.5 – 3 GB/s

→ 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:



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

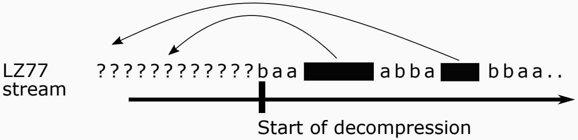
In **regular** `.fastq.gz` files:

- 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

Random access in gzip stream

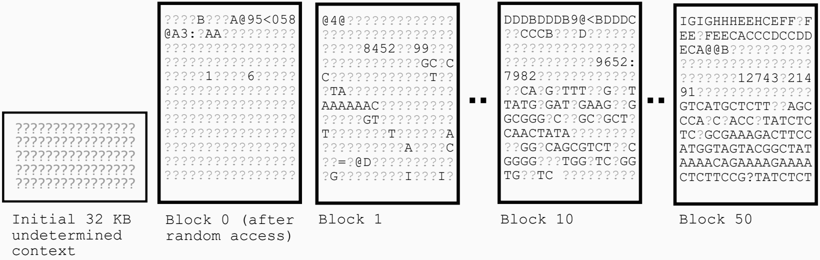
Suppose you start decompressing at the middle of a LZ77 stream.

- 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.



```
????????????????
????????????????
????????????????
????????????????
????????????????
```

Initial 32 KB undetermined context

```
????B???A@95<058
@A3:?AA?????????
????????????????
????????????????
?????1????6?????
????????????????
????????????????
????????????????
????????????????
????????????????
????????????????
????????????????
```

Block 0 (after random access)

```
@4@?????????????
????????????????
?????8452??99???
????????????GC?C
C????????????T??
?TA?????????????
AAAAAAC?????????
?????GT?????????
T???????T??????A
??????????A????C
??=?@D??????????
?G????????I???I?
```

Block 1

..

```
DDDBDDDB9@<BDDDC
??CCCB???D??????
????????????????
???????????9652:
7982????????????
??CA?G?TTT??G??T
TATG?GAT?GAAG??G
GCGGG?C??GC?GCT?
CAACTATA????????
??GG?CAGCGTCT??C
GGGG???TGG?TC?GG
TG??TC ?????????
```

Block 10

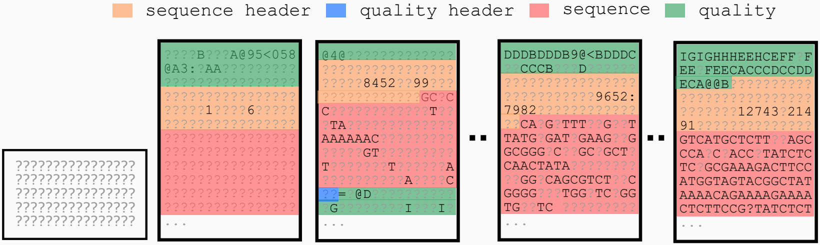
..

```
IGIGHHHEEHCEFF?F
EE?FEECACCCDCCDD
ECA@@B??????????
????????????????
???????12743?214
91??????????????
GTCATGCTCTT??AGC
CCA?C?ACC?TATCTC
TC?GCGAAAGACTTCC
ATGGTAGTACGGCTAT
AAAACAGAAAAGAAAA
CTCTTCCG?TATCTCT
```

Block 50

Random access in compressed FASTQs

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



Initial 32 KB undetermined context

Block 0 (after random access)

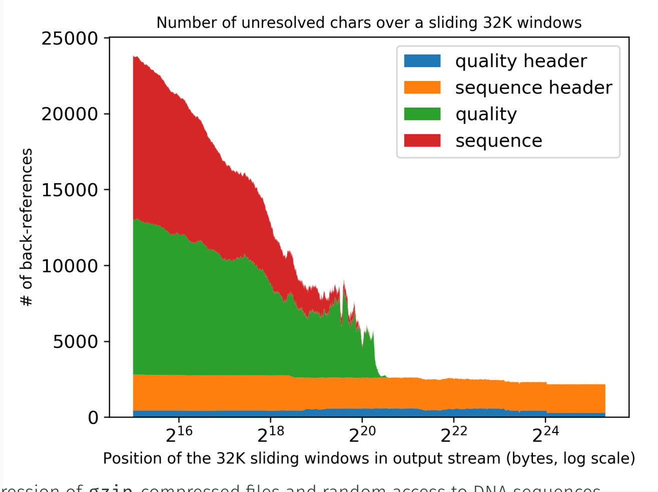
Block 1

Block 10

Block 50

Random access in compressed FASTQs

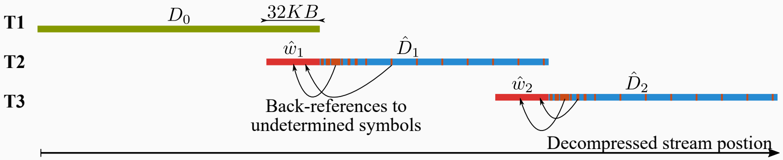
At normal compression level, after a random access, sequences can be fully determined after a while (2^{20} bytes), but not headers.



Parallel decompression

We designed 2-step parallel decompression algorithm.

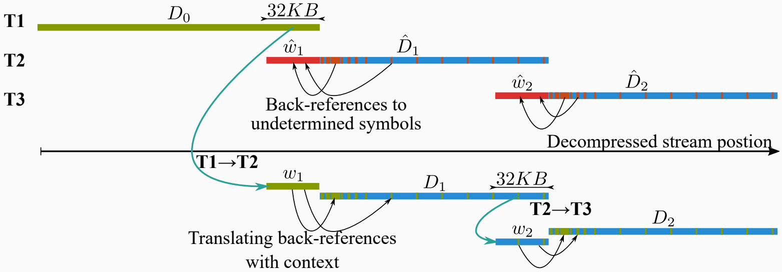
1. Partition file, decompress chunks independently, record unresolved back-references


T1
T2
T3
D_0
$32KB$
$\hat{w}_1$
$\hat{D}_1$
$\hat{w}_2$
$\hat{D}_2$
Back-references to undetermined symbols
Decompressed stream postion

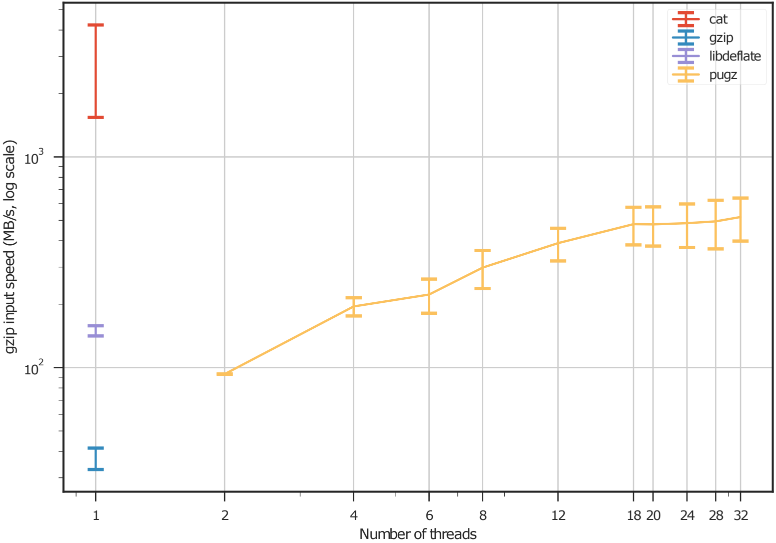
Parallel decompression

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


T1
D_0
$32KB$
T2
$\hat{w}_1$
$\hat{D}_1$
T3
$\hat{w}_2$
$\hat{D}_2$
Back-references to undetermined symbols
Decompressed stream postion
T1→T2
w_1
D_1
$32KB$
T2→T3
w_2
D_2
Translating back-references with context

Performance



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`