Efficient Encoding and Compression Techniques for the SLOW5 File Format

Sasha Jenner
Genomic Technologies Group

ABSTRACT

Contemporary data storage of raw nanopore signals in the FAST5 file format doesn't benefit from parallel file access. A more computationally resourceful and space efficient file format could result in significant improvements in the runtime and storage size of nanopore sequencing pipelines.

To address this issue, binary and compressed binary equivalents to the existing SLOW5 format were designed. Parallel access was implemented using SLOW5 index files and multithreading.

Benchmarking experiments to determine the access time and file size of each SLOW5 format and their corresponding FAST5 files were performed using a sequenced human genome dataset on a rack-mounted server. The binary SLOW5 format was found to have the fastest access time with on average roughly 5000 reads accessed per second using 32 threads. Whilst the compressed binary SLOW5 file format was the most space efficient, using 250kB per read on average. In comparison, using the FAST5 format on average 100 reads could be accessed per second, with each read using roughly 300kB on average.

By exploiting modern CPU architectures with multithreading whilst employing space efficient compression techniques, the runtime and space requirements of nanopore sequencing pipelines can be significantly improved.

Contents

List of Figures

List of Tables

1 BACKGROUND

Genome sequencers read DNA strands in segments and convert each segment into a packet of data known as a *read*. DNA sequencing devices from Oxford Nanopore Technologies (ONT) are no different and record disturbances in ionic current as DNA molecules are passed through a biological nanopore [1]. These measurements can be translated to determine the sequence of each DNA molecule analysed.

The time series current signal data is written in a format called FAST5, which stores the raw data for each nanopore sequencing read. FAST5 is a Hierarchical Data Format 5 (HDF5) file [2] with a specific schema defined by ONT. HDF5 is a complex file format for storing and managing high volume data. It works well with the time series current signal of a nanopore read, since it uses B-trees to index table objects.

However, the official library used to access the HDF5 file format is not scalable with threads [3]. This means that tasks performed with the FAST5 file format suffer from an inefficient utilisation of parallel resources. Therefore, the process of basecalling FAST5 data into DNA sequence reads and other common analyses that utilise signal-level data (such as DNA methylation calling with Nanopolish [4] or f5c [5]) slow down the typical sequencing pipeline.

2 INTRODUCTION

A new file format is needed to improve the efficiency of these bioinformatics tasks. SLOW5 is a simple file format designed to address this [6]. It is a tab-separated values (TSV) file with a header marked by lines beginning with a '#', followed by the metadata and time series current signal data for one nanopore read per line. An example is shown in table 4.

In order to read a SLOW5 file using multiple threads of execution, a corresponding SLOW5 index file is used [6]. It is also a TSV file and begins with a single header line prepended by a '#' to define the column names and their order. Each following line represents the location of a particular read in the corresponding SLOW5 file by storing its file offset and length in bytes. See table 1 for an example.

Table 1: Example of a SLOW5 index file.

#read_id	file_offset	length
id-0	120	8073
id-1	8193	72705
⋮	⋮	⋮
id-N	397774	56481

In the current readable format, a SLOW5 file containing multiple reads takes up much more memory than its corresponding FAST5 files, each containing one read. Thus, the *aim* is to efficiently compress the SLOW5 file format to a size smaller than or equal to the total size of its corresponding FAST5 files. Whilst still maintaining multithreaded access to the SLOW5 file format and achieving the best possible level of performance within these constraints.

3 METHODS

3.1 Encodings

Two encodings of the SLOW5 file format were designed and implemented. The first is a binary encoding where each numerical field is stored in binary rather than in the ASCII format. This is analogous to how SAM [7, 8] and VCF [9], two gold standard genomics file formats, have their own binary counterparts BAM and BCF. This encoding is more natural for almost all modern computers since digital memory is designed to store data in binary. It is also significantly more efficient in theory, with one byte of data capable of storing the 10 digits in ASCII versus 256 (= 2^8) numbers in binary.

The binary encoding also removes the commas, tab characters and the newline characters separating signal data, fields and reads respectively since the size of each numerical field is now static. For all remaining variable length fields such as the read identifier (ID), the encoding prepends them by a static sized numerical field storing their length in bytes. This prepended field is also stored in binary. An example is displayed in figure 3.

The second encoding is a compressed binary encoding in the gzip file format [10]. This is achieved by separately compressing the header and each read of the binary encoded SLOW5 file format using the default level of compression, internal memory allocation and compression algorithm of the zlib C library [11]. An example is shown in figure 4.

3.2 Multithreading

Reading *reads* from the SLOW5 file format can be performed by multiple threads in parallel as was mentioned in section 2. The method is the same for all encodings (ASCII, binary and compressed binary) and works as follows.

The algorithm takes as input a SLOW5 file and a list of read IDs. It then outputs the list of reads from the SLOW5 file corresponding to the given read IDs. First of all, it generates a SLOW5 index file (section 2) for the SLOW5 file of interest, if it does not exist already. This index file is then stored in memory as a hash table where the key is the read ID and the values are the location and size in bytes of each read. Then, the list of read IDs are processed one batch at a time to avoid an out of memory state, with the maximum number of read IDs in a batch set arbitrarily to 2^{12}. Each thread is then allocated an equal amount of read IDs from the current batch. The threads then begin iterating in parallel through their allocated read IDs.

For each read ID, the offset and length of the corresponding read are found from the hash table. The offset is then used to locate the beginning of the read in the SLOW5 file and the length dictates how many bytes to read from that position into memory for later output.

Work stealing between threads occurs in the case that a thread completes the above task for all their allocated reads earlier than another. Once all read IDs in the current batch have been processed, the stored reads are written as output and freed from the program's memory. The next batch then begins if there are remaining read IDs to process from the initial list.

This algorithm displays the superiority of the simple SLOW5 file format over FAST5 since it very basically overcomes the thread scalability issue outlined in section 1. In fact, it implements parallel access without any synchronisation mechanisms and is inherently thread-safe since the SLOW5 file is read-only throughout the entire algorithm and each stored read is written to a pre-allocated position in memory.

4 EXPERIMENTS

4.1 Design

Experiments to evaluate the performance of each file type and encoding were designed and executed. The dataset of FAST5 files used for these experiments was obtained by sequencing a genome sample from an adult male human on the ONT GridION. It's details are summarised in table 2.

Table 2: The dataset used for the experiments.

Description	Adult male human genome
Sequencer	ONT GridION
Sequencing time (days)	3
No. of base pairs (Gb)	10.56
No. of reads	496 368
Avg. read length (bp)	21 278
Max read length (bp)	1 328 419

A list of read IDs representative of a typical analysis's order of read requests was generated by sorting the dataset's BAM file [7]

by chromosome number and then by the base location on each chromosome. The result is a list of 534 000 read IDs sorted by their genomic coordinates with 480 004 being unique. This covers 96.7% of the number of reads in the dataset, making it a robust list of read IDs for performance benchmarking.

The experiments were executed on a rack-mounted server capable of running up to 40 threads of execution in parallel. This amount is beneficial for observing the full relationship between performance and the number of threads used. It's specifications (table 3) are typical for a small high performance computing (HPC) server which is usually available for genomics research scientists. The server's disk cache was also cleared before executing each experiment in order to prevent inaccurate I/O results.

Table 3: Specifications of the server used for the experiments.

Description	Dell PowerEdge C4140 Server Rack
CPUs	2 × Intel Xeon Silver 4114
CPU cores	2 × 10
CPU threads	2 × 20
RAM	376GB
Disk System	6.4TB NVMe drive
File System	ext4
OS	Ubuntu 18.04.5 LTS

Two experiments were designed to evaluate performance. The first aimed to compare the size of each file type and encoding. Using the dataset's FAST5 files, equivalent SLOW5 files encoded in ASCII, binary and compressed binary were created. Each file's size was then measured in bytes using the Unix utility *wc* [12].

The second experiment aimed to compare the read access time of each file type and encoding using a varying number of threads. The motivation for varying the number of threads was to visualise the thread scalability issue from section 1 and how it was overcome in section 3.2. For each file type and encoding, the total time taken to retrieve and store each read corresponding to the list of read IDs described in section 4.1 was recorded. However, in order to reasonably compare the different encodings, each read was additionally transformed into the ASCII format which would be necessary in most user applications. This transformation time was also recorded, and was combined with the total read access time.

However, the file sizes and total read access times recorded by both experiments are specific for this dataset and list of read IDs. Hence, more normalised metrics were devised based on the number of base pairs in the dataset and corresponding reads of the list of read IDs. In particular, rather than comparing the raw file sizes, the average file size per base pair was used. And instead of comparing the total read access times, the average number of base pairs accessed per second was used.

4.2 Results

To benchmark the access time and file size of each format and encoding technique storing nanopore sequencing reads, experiments were performed on a sample of human DNA. A realistic list of read IDs was recorded from a sequencing pipeline and used to obtain the total access time for each format and encoding. Dividing the total number of reads by the total access time, we can obtain a normalised dataset-independent metric for comparing performance. Figure 1 plots this metric for each file type and encoding using the described dataset.

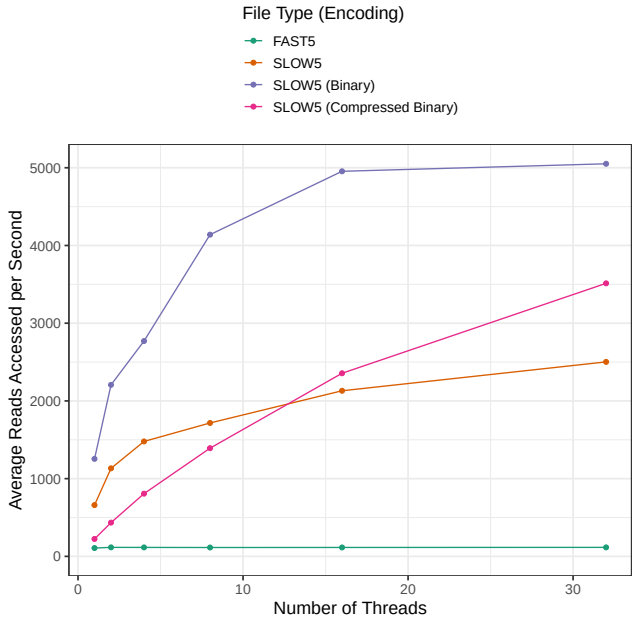

Figure 1: Number of reads accessed and transformed per second on average for each file format and encoding technique against the number of threads used.

A similar metric for comparing the space efficiency of each file type and encoding independent of datasets was devised by finding the size in bytes per read. Figure 2 was obtained on the aforementioned data and shows the size differences between each file type and encoding using this metric.

5 DISCUSSION

When integrated with popular third part tools for nanopore sequencing analysis (e.g. Bonito, Nanopolish and F5C), it is anticipated that the SLOW5 format will deliver considerable performance advantages that will scale with the number of allocated threads. Given these advantages, the SLOW5 format should be readily adopted by the Nanopore sequencing community.

Different levels of compression and algorithms used by the zlib library for the compressed binary encoding are for future work.

More datasets to generate more robust measurements. A server with RAID Other read lists

6 CONCLUSION

REFERENCES

[1] Jain et al. 2016. Erratum to: The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology* 17 (2016), 256. https://doi.org/10.1186/s13059-016-1103-0

[2] The HDF Group. 2016. HDF5 File Format Specification Version 3.0. https://support.hdfgroup.org/HDF5/doc/H5.format.html. (April 2016). Accessed: 25-11-2020.

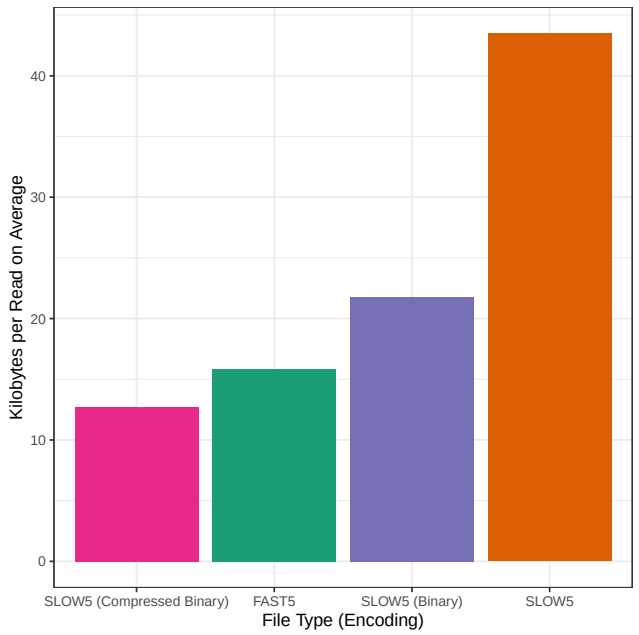

Figure 2: Kilobytes stored per read on average for each file type and encoding technique.

[3] The HDF Group. 2020. Questions about thread-safety and concurrent access. https://portal.hdfgroup.org/display/knowledge/Questions+about+thread-safety+and+concurrent+access. (2020). Accessed: 24-11-2020.

[4] Simpson et al. 2017. Detecting DNA cytosine methylation using nanopore sequencing. *Nature Methods* 14 (Feb. 2017), 407–410. https://doi.org/10.1038/nmeth.4184

[5] Gamaarachchi et al. 2020. GPU accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis. *BMC Bioinformatics* 21 (Aug. 2020), 343. https://doi.org/10.1186/s12859-020-03697-x

[6] Gamaarachchi. 2020. *Computer Architecture-Aware Optimisation of DNA Analysis Systems.* Ph.D. Dissertation. Faculty of Engineering, UNSW, High St, Kensington, NSW 2052, Australia. Chapter 7.4.1.

[7] Li et al. 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 25, 16 (Aug. 2009), 2078–2079. https://doi.org/10.1093/bioinformatics/btp352

[8] Li. 2011. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics* 27, 21 (Nov. 2011), 2987–2993. https://doi.org/10.1093/bioinformatics/btr509

[9] Danecek et al. 2011. The variant call format and VCFtools. *Bioinformatics* 27, 15 (Aug. 2011), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

[10] Deutsch and Randers-Pehrson. 1996. *GZIP file format specification version 4.3.* https://tools.ietf.org/html/rfc1952.

[11] Gailly and Adler. 2017. *zlib 1.2.11 Manual.* https://zlib.net/manual.html.

[12] The Open Group 2018. *The Open Group Base Specifications* (7 ed.). The Open Group. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/wc.html.

A APPENDIX

Table 4: Example of the SLOW5 file format in the ASCII encoding.

##file_format=slow5v0.1						
#read_id	n_samples	digitisation	offset	range	sample_rate	raw_signal
id-0	6028	8192.0	3.0	1467.6	4000.0	1373, 712, 738, 715, 716, . . .
id-1	59676	8192.0	23.0	1467.6	4000.0	1039, 588, 588, 593, 586, . . .
⋮	⋮	⋮	⋮	⋮	⋮	⋮
id-N	45690	8192.0	5.0	1467.6	4000.0	1255, 773, 617, 574, 568, . . .

```
##file_format=blow5v0.1
#read_id    n_samples   digitisation    offset  range   sample_rate raw
^D^@^@^@^@^@^@^@id-0^\é^@^@^@^@^@^@^@^@F^@^@¸A<85>s·D^@^@zE^Q^DL^BL^B
^D^@^@^@^@^@^@^@id-11=^@^@^@^@^@^@^@^@F^@^@<90>A<85>s·D^@^@zEÓ^Dd^Bc^
...
^D^@^@^@^@^@^@^@id-NÊ2^@^@^@^@^@^@^@^@F^@^@À@<85>s·D^@^@zEé^D<92>^B^O
```

Figure 3: Binary encoding of the example SLOW5 file shown in table 4. Note: newline characters between the header and each read have been added for readability and '…' indicates where more information is not displayed.

```
1 ^_<8b>^H^@^@^@^@^@^@^C%Ê1
2 <80>0^L^@À9ýFwÑÁÑ·<84>HÓ^RHSi<82>~_Áõ¸<9c>«(c^]³S^\§<8e>g¿×eKy2^U<94>^B
  <8c>Z<9d>^C&Ycø^GN
3 þäA<97>fɐé^E´<99>ro\^@^@^@
4 ^_<8b>^H^@^@^@^@^@^@^C<<9d>^E¸<94>ÅÛÆ<99>^Aé<94>n$Eº»^[é<92>^RP@1^P;0±^
5 ^_<8b>^H^@^@^@^@^@^@^C=<9d>^E¸TÕÛöU<8b><94>NAB^Zi¥»^O^RJ<83> ^BJK© <9d>
6 ...
7 ^_<8b>^H^@^@^@^@^@^@^C=<9d>^GÜ<8f>Õÿÿ<9d>c¯
8 ¡"<9b><94><91>=JF)Ù;{eo¢<8c>^PJ¨o{<90>^URQDi 2";3dï-;Ùÿçëuß¿¿ëñ^Y÷õ¹®s<
```

Figure 4: Compressed binary encoding of the example SLOW5 file shown in table 4. The header found in figure 3 is compressed in lines 1, 2 and 3. Whilst the compressed reads corresponding to read IDs '*id-0*', '*id-1*' and '*id-N*' are found in lines 4, 5 and 7/8 respectively. Note: newline characters between the header and each read have been added for readability and '…' indicates where more information is not displayed.

NOMENCLATURE

ASCII	American Standard Code for Information Interchange: An encoding for representing text on computers, page 2
Avg.	Average, page 2
bp	Base pairs, page 2
CPU	Central processing unit, page 1
DNA	Deoxyribonucleic acid, page 1
Gb	Giga base pairs, page 2
HDF5	Hierarchical Data Format 5, page 1
HPC	High performance computing, page 3
I/O	Input/output, page 3
ID	Identifier, page 2
No.	Number, page 2
ONT	Oxford Nanopore Technologies, page 1
OS	Operating system, page 3
RAM	Random-access memory, page 3
Read	Data generated from sequencing a segment of DNA, page 1
TSV	Tab-separated values, page 2