cljam: a library for handling DNA sequence alignment/map (SAM) with parallel processing
Toshiki Takeuchi*, Atsuo Yamada, Takashi Aoki and Kunihiro Nishimura

Abstract
Background: Next-generation sequencing can determine DNA bases and the results of sequence alignments are generally stored in files in the Sequence Alignment/Map (SAM) format and the compressed binary version (BAM) of it. SAMtools is a typical tool for dealing with files in the SAM/BAM format. SAMtools has various functions, including detection of variants, visualization of alignments, indexing, extraction of parts of the data and loci, and conversion of file formats. It is written in C and can execute fast. However, SAMtools requires an additional implementation to be used in parallel with, for example, OpenMP (Open Multi-Processing) libraries. For the accumulation of next-generation sequencing data, a simple parallelization program, which can support cloud and PC cluster environments, is required.

Results: We have developed cljam using the Clojure programming language, which simplifies parallel programming, to handle SAM/BAM data. Cljam can run in a Java runtime environment (e.g., Windows, Linux, Mac OS X) with Clojure.

Conclusions: Cljam can process and analyze SAM/BAM files in parallel and at high speed. The execution time with cljam is almost the same as with SAMtools. The cljam code is written in Clojure and has fewer lines than other similar tools.

Keywords: Next-generation sequencing, DNA, Parallel processing, Clojure, SAM, BAM

Abbreviations: BAI, BAM index; BGZF, Blocked GNU zip format; LOC, Lines of code; NGS, Next generation sequencing; SAM, Sequence alignment/map

Background
Next-generation sequencing (NGS) technologies have allowed DNA sequences to be generated very fast and in parallel. Complete DNA sequences can be obtained by statistical analysis of the raw data from the sequencers. As a result, tools for data analysis and interpretation of the sequencing results are in high demand. For maximum efficiency, data should be processed in parallel and with high speed considering the accumulation speed and size of NGS data. A lightweight program that can deal with NGS data in parallel is required.

Most NGS sequencers generate hundreds of millions of short sequence reads for each DNA or RNA sample. These short read data are small pieces of DNA sequence bases. The DNA and RNA sequence data are saved mainly in FASTQ format, which is a text-based format for sequences and their quality scores. Typically, FASTQ files contain about 300 million reads that are about 200-300 nucleotides long. The short reads in FASTQ files are generally mapped and aligned to a reference genome with alignment mapping tools such as BWA [1] and Bowtie [2]. The alignment data are stored mainly in Sequence Alignment/Map (SAM) format files, which are tab-delimited text files. BAM is the compressed binary version of the SAM format. BAM uses BGZF (Blocked GNU Zip Format) compression and can support indexes to achieve fast random access by generating BAM index (BAI) files.

SAMtools [3, 4] is written in the C programming language and uses SAM/BAM files. It has various functions for manipulating SAM/BAM files, such as viewing, sorting, indexing, and pileup. The 'index' command creates a BAI file for fast random access to the original BAM file. Counting the overlapping short read bases at a specified
location is called pileup. The ‘mpileup’ command executes
pileup and outputs the results in text format, which is use-
ful for visualizing genome histograms and for detecting
variants/insertions/deletions in a genome sequence.
SAM/BAM utilities are also available in other program-
ming languages. Picard [5] is a Java-based command-line
utility for manipulating high-throughput sequencing data
and formats such as SAM and BAM. Because of their per-
formance, some lightweight languages have been used to
wrap SAMtools. For example, pysam [6] is a lightweight
wrapper of SAMtools C-API written in the Python pro-
gramming language, and bio-samtools [7, 8] is a Ruby
language interface to SAMtools.

Implementation
The Clojure programming language
Clojure is a lightweight programming language that is
favored for huge data analysis with parallel processing [9].
It is a functional programming language and is a dialect
of Lisp. Clojure runs on the Java Virtual Machine, which
includes Windows, Mac OS, and Linux environments. It
is based on Java, which allows Java libraries to be used.

Genome sequence analyses processes can be written sim-
ply because Clojure provides many convenient functions
for manipulating list data. Moreover, immutability is the
center of Clojure’s design policy so that parallel processing

can be used efficiently.

Clojure has been used to code some bioinformatics
tools. For example, BioClojure [10] is a convenient library
for bioinformatics under the open source Bio* projects.
It consists of parsers for various kinds of file formats
(UniProtXML, Genbank XML, FASTA, and FASTQ), and
wrappers of data analysis programs (BLAST, SignalP,
TMHMM, and InterProScan). However, BioClojure does
not have functions for SAM/BAM manipulation and is not
fully implemented in Clojure. The CONNJUR-Sandbox
source [11] contains examples of the visualization of pro-
tein structures using PDB data in Clojure and the pre-
diction of neighboring amino acids with Support Vector
Machine algorithms named Clojure Amino Acid Predic-
tor.

cjam
Here, we describe cjam, a SAM/BAM manipulating
library written in Clojure. With cjam, we aimed to pro-
vide a much more simple source code than SAMtools
that is equal in performance and can work in a Clojure
ecosystem.

Cjam is not a SAMtools wrapper. It does not use an
external application programming interface (API) such as
SAMtools and Picard for simple and high maintainable
codebase. Programs in Clojure are not as fast on a sin-
gle thread, but because of its parallel processing functions
it can be easily sped up. Cjam supports multithreaded
processing in high-cost features such as BAM indexing
and pileup. Parts of File I/O are written in Java because of
high-speed processing. Cjam uses an open-source com-
pression library for BGZF, named bgzf4j [12], which was
developed by the authors of this paper.

Cjam has the following functions:
• Reading and writing SAM/BAM/FASTQ
• Converting SAM/BAM
• Normalization
• Sorting
• Indexing BAM
• Pileup
• Indexing FASTA

Results and discussion
Using cjam: a brief tutorial
Here are examples of interacting with SAM/BAM files
using cjam. More information on usage and specific func-
tions is provided in the readme file and https://chrovis.
github.io/cljam/.

Installation
Cjam is available as a Clojure library at Leiningen, a popu-
lar build tool for Clojure projects. The following statement
should be added to a Leiningen configuration.
:dependencies [[cjam "0.1.3"]]

Leiningen automatically downloads the Java Archive of
cjam and resolves its dependency in a project. Then,
cjam functions can be used in the code.

Reading a SAM/BAM file
Cjam provides a file reader and a namespace including
various I/O functions to read a SAM/BAM file. The fol-
lowing code opens a BAM file and retrieves the first five
alignments, where pnext, tlen, flag, qname, and rname
indicate the position of the mate/next read, observed
template length, bitwise flag, query template name, and
reference sequence name, respectively, based on the SAM
format [13].

(cljam.core :refer [reader]
'[cljam.io :as io])
(with-open [r (reader "path/to/file.bam")]
(doall (take 5 (io/read-alignments r)))
=> [{:pnext 37, :tlen 39, :flag 163, :pos 7,
:qname "r001", :rname "ref", ...}
{:pnext 0, :tlen 0, :flag 0, :pos 9,
:qname "r002", :rname "ref", ...}
...]

Sorting a SAM/BAM file
A SAM/BAM file can be sorted by chromosomal
coordinates or reference name using functions in the
‘cjam.sorter.’ For example, to create a BAM file sorted by
chromosomal coordinates,

(require '[cljam.core :refer [reader writer]])
In this case, the input and output files are file.bam and sorted.bam, respectively.

**Indexing a BAM file**

The 'cljam.bam-indexer' has functions for indexing a BAM file. The following code creates a BAI file from a BAM file.

```clojure
(require '[cljam.bam-indexer :as bai])
(bai/create-index "path/to/sorted.bam" "path/to/sorted.bam.bai")
```

**Getting pileup information**

The 'cljam.pileup' provides pileup and mpileup functions equivalent to those of SAMTools. For example, to get simple pileup of the first 10 genomic positions of chr1 reference,

```clojure
(require '[cljam.core :refer [reader]]
 'cljam.pileup :as plp])
(with-open [r (reader "path/to/sorted.bam" :ignore-index false)]
  (take 10 (plp/pileup r "chr1" nil)))
```

```clojure
=> [0 0 1 1 3 3 3 3 2 2]
```

**Command line interface**

The command line interface of cljam provides an additional feature to quickly check its functions. For example, the following command displays contents of a SAM file including header information.

```
$ cljam view --header path/to/file.sam
@HD VN:1.4 SO:coordinate
@SQ LN:45 SN:ref
@SQ LN:40 SN:ref2
r001  163  ref  7  30  8M4..`
r002  0   ref  9  30  1S2..`
```

**Performance of indexing and pileup**

We conducted timing measurement experiments to determine the performance of BAM indexing and pileup under a changing number of thread conditions: 1, 2, 4, 8, and 12 threads with cljam (v0.1.3), SAMtools (v1.2) (single thread), and Picard (v1.134) (single thread). We used a BAM file (about 13.2GB) from the 1000 Genomes Project [14]. The machine specifications were CPU: Intel Core i7-4930K @ 3.40 GHz, 12 MB L2 cache, 12 cores (6 real cores & HT), 64 GB RAM, and SSD storage.

The results for indexing and pileup are shown in Figs. 1 and 2, respectively. Each condition was measured 10 times and the average time of the 10 trials was plotted.

The results indicate that the execution times for cljam were getting shorter until the 4 thread condition in indexing and 3 thread in pileup. However, the execution times under the conditions of above 6 threads in indexing and 4 threads in pileup were almost same. We believe there may be an overhead of the file I/O when reading BAM files; the performance does not improve in parallel conditions. The execution time of pileup in cljam with the 3 thread condition was 1.3 times longer than with SAMtools, which can be considered as almost the same performance.

**Code metrics**

Code readability and maintainability are more important than optimization of code under our software development environment, which uses recent high-speed and multi-core CPU technologies. Thus, we used CLOC [15]...
Table 1 Measurement of LOC

| Language   | Files | Blank | Comment | Code |
|------------|-------|-------|---------|------|
| cljam (Clojure) | 46    | 466   | 165     | 3264 |
| SAMtools (C/C++) | 53    | 1606  | 2403    | 11619|
| Picard (Java)   | 290   | 6409  | 11835   | 28322|

We have developed cljam as an open-source software using Clojure, which is a functional programming language that works on the Java Virtual Machine. Cljam can process and analyze SAM/BAM files in parallel and at high speed. The execution time with cljam is almost the same as with SAMTools. The Clojure code of cljam has fewer lines and an equivalent performance compared with SAMtools and Picard, which are similar tools.

Conclusions

We have developed cljam as an open-source software using Clojure, which is a functional programming language that works on the Java Virtual Machine. Cljam can process and analyze SAM/BAM files in parallel and at high speed. The execution time with cljam is almost the same as with SAMTools. The Clojure code of cljam has fewer lines and an equivalent performance compared with SAMtools and Picard, which are similar tools.

Availability and requirements

Project name: cljam
Project home page: https://github.com/chrovis/cljam
Operating system(s): Platform independent
Programming language: Clojure
Other requirements: none
License: The Apache License, Version 2.0
Any restrictions to use by non-academics: none

Acknowledgements

Not applicable.

Funding

This work was partly supported by JST CREST “Creation of Fundamental Technologies for Understanding and Control of Biosystem Dynamics.”

Authors’ contributions

TT wrote the source code, conducted the experiments, and wrote the manuscript. AY participated in writing the source code and conducted the tests. TA designed the software and provided technical advice to the other programmers. KN participated in the design process, coordination and helped to draft the manuscript. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Consent for publication

Not applicable.

Ethics approval and consent to participate

Not applicable.

References

1. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics. 2009;25(14):1754–60.
2. Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 2009;10(3):25.
3. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Hornier N, Marth G, Abecasis G, Durbin R. The Sequence Alignment/Map format and SAtools. Bioinformatics. 2009;25(16):2078–9.
4. Li H. SAMtools : http://www.htslib.org/. Accessed 11 Aug 2016.
5. Institute B. Picard : http://broadinstitute.github.io/picard/. Accessed 11 Aug 2016.
6. Heger A. PySAM. https://github.com/pySam-developers/pysam. Accessed 11 Aug 2016.
7. Ramirez-Gonzalez RH, Bonnal R, Caccamo M, Maclean D. Bio-samtools: Ruby bindings for SAMtools, a library for accessing BAM files containing high-throughput sequence alignments. Source Code Biol Med. 2012;7(1):6.
8. Etherington GJ, Ramirez-Gonzalez RH, MacLean D. Bio-samtools 2: a package for analysis and visualization of sequence and alignment data with SAMtools in Ruby. Bioinformatics. 2015;31(15):2565–7.
9. Hickey R. The clojure programming language. In: Proceedings of the 2008 Symposium on Dynamic Languages. DLS ’08. New York: ACM; 2008. p. 1–111.
10. Pleskott J, Rinaldi G, Brindley PJ, Jia X, Potriquet J, Bethony J, Hinman ML, Vyas J, Gryk MR, Schiller C, Ellis HJ, Hinman ML, Mulvenna J, Bioclojure: a functional library for the manipulation of biological sequences. Bioinformatics. 2014;30(17):2537–9.
11. Fenwick M, Sesanker C, Schiller MR, Ellis HJ, Hinman ML, Vyas J, Gryk MR, An Open-Source Sandbox for Increasing the Accessibility of Functional Programming to the Bioinformatics and Scientific Communities. Proc Int Conf Inf Technol New Gener. 2012;2012:89–94.
12. Xcoo I. Bgzf4j. https://github.com/chrovis/bgzf4j. Accessed 11 Aug 2016.
13. Group TSFSW. Sequence Alignment/Map Format Specification. http://samtools.github.io/hts-specs/SAMv1.pdf. Accessed 11 Aug 2016.
14. Project TG. HG00125.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam. ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/HG00125/alignment/HG00125.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam. ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/HG00125/alignment/HG00125.mapped.ILLUMINA.bwa.GBR.low_coverage.20120522.bam. Accessed 11 Aug 2016.
15. Danial A. Count Lines of Code, Version: 1.64, Released 2015-06-27. https://github.com/AlDanial/cloc. Accessed 11 Aug 2016.

Received: 18 May 2016 Accepted: 8 August 2016
Published online: 17 August 2016