

Package: BamScaleR (via r-universe)

October 13, 2026

Type Package

Title Bioconductor-Friendly Multithreaded BAM Processing

Version 0.99.2

Description Multithreaded sequential BAM processing built on top of the ompBAM C++ engine. BamScaleR provides user-friendly BAM read and scan interfaces designed for compatibility with existing Bioconductor workflows.

License MIT + file LICENSE

biocViews Software, DataImport, Sequencing, Coverage, Alignment, QualityControl

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

Imports BiocGenerics, BiocParallel, Biostrings, GenomeInfoDb, GenomicAlignments, GenomicRanges, IRanges, methods, ompBAM, Rcpp, Rsamtools, S4Vectors

LinkingTo Rcpp, ompBAM, S4Vectors, IRanges, XVector

Suggests BiocStyle, chipseqDBData, dplyr, ExperimentHub, ggplot2, knitr, readr, rmarkdown, rtracklayer, scales, tibble, testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements C++17, OpenMP

NeedsCompilation yes

URL <https://cparsania.github.io/BamScaleR/>

BugReports <https://github.com/cparsania/BamScaleR/issues>

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libbz2-dev liblzma-dev libssl-dev xz-utils zlib1g-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-10-13 04:12:14 UTC
RemoteUrl <https://github.com/BiocStaging/BamScaleR>
RemoteRef HEAD
RemoteSha 5918290c5d218b00e29ac3e93a1f9e85e7e46817

Contents

bam_count	2
bam_coverage	3
bam_coverage_bigwig	4
bam_read	6
decode_compact_qual	9
decode_compact_seq	10
decode_seqqual_compact	10
fragment_sizes	11
mapq_dist	13
Index	15

bam_count	<i>Count BAM records with Bioconductor-compatible filtering</i>
-----------	---

Description

bam_count() provides a fast chromosome-level count summary, honoring key filtering fields from ScanBamParam (mapqFilter, flag, and which).

Usage

```
bam_count(  
  file,  
  param = NULL,  
  threads = 1L,  
  BPPARAM = BiocParallel::bpparam(),  
  auto_threads = FALSE,  
  include_unmapped = TRUE  
)
```

Arguments

file	BAM input (character, BamFile, or BamFileList).
param	Optional Rsamtools::ScanBamParam (or compatible list).
threads	Requested number of OpenMP threads. May be capped when auto_threads = TRUE.
BPPARAM	BiocParallel parameter for multi-file operation. Defaults to BiocParallel::bpparam(). Set to NULL to force serial file processing.

`auto_threads` Logical; when TRUE and BPPARAM has multiple workers, BamScaleR adaptively avoids oversubscription by preserving higher per-file OpenMP thread counts when possible and reducing the number of concurrently active file workers before shrinking per-file threads.

`include_unmapped` Whether to include an extra * row for unmapped records.

Details

Parallelism behavior matches `bam_read()`: BPPARAM distributes work across BAM files, while `threads` controls OpenMP work within each file. If `auto_threads = TRUE` and BPPARAM has multiple workers, BamScaleR first limits the number of concurrently active workers to preserve the requested per-file thread count within the detected core budget, then caps per-file OpenMP threads only if a single file would still oversubscribe the machine.

Value

For one file: a `data.frame` with columns `seqname`, `seqlength`, `count`. For multiple files: named list of such `data.frame`s.

Examples

```
bam <- ompBAM::example_BAM("Unsorted")
bam_count(bam, threads = 2)
```

<code>bam_coverage</code>	<i>Per-base coverage, aggregated in C++</i>
---------------------------	---

Description

Computes per-base read coverage by folding a per-contig difference array inside the multithreaded C++ reader and run-length encoding the result, without ever materialising the alignments as an `RGAAlignments` object. This is the fast path for coverage / bigWig generation on large BAMs: the standard route (`GenomicAlignments::coverage(readGAAlignments(file))`) builds a full `GAAlignments` in R first, which is the dominant cost on 100M+-read files.

Usage

```
bam_coverage(
  file,
  param = NULL,
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE
)
```

Arguments

file	A BAM file path, Rsamtools::BamFile , or a vector / BamFileList of them.
param	Optional Rsamtools::ScanBamParam (or a compatible list). Only its flag and mapqFilter are honoured, applied per read in C++. With the default NULL the filter matches readGAlignments() (drop only unmapped reads). Note that which (region) restriction is not applied.
threads	Number of OpenMP threads for within-file decoding.
BPPARAM	A BiocParallel::BiocParallelParam for across-file parallelism when file has more than one element.
auto_threads	Logical; if TRUE, balance the thread / worker split.

Details

The result is **byte-identical** (identical()) to GenomicAlignments::coverage(GenomicAlignments::readGAlignments). CIGAR operations M, =, X and D contribute to coverage, N introduces a gap, and I/S/H/P are ignored; only unmapped reads (SAM flag 0x4) are dropped by default – secondary, supplementary, duplicate and QC-fail reads are counted, exactly as readGAlignments() does.

Value

An [IRanges::RleList](#) (SimpleRleList) of integer coverage, one element per BAM-header sequence in header order, each of length equal to the header sequence length (contigs with no reads are all-zero runs). For multiple input files, a named list of such RleLists.

See Also

[fragment_sizes\(\)](#), [mapq_dist\(\)](#), [bam_read\(\)](#).

Examples

```
bam <- ompBAM::example_BAM("Unsorted")

# Per-base coverage as an RleList, with no alignments materialised in R
cov <- bam_coverage(bam, threads = 2)
cov[1]
```

Description

Computes per-base coverage (identical to [bam_coverage\(\)](#)) and writes it directly to a bigWig file entirely in C++ via the bundled libBigWig, without ever materialising the coverage as an R object. This is the fast path for generating coverage tracks from large BAMs: the usual route (`rtracklayer::export.bw(bam_coverage(file), out)`) round-trips a large `RleList` through R, and the serialisation dominates. Only non-zero coverage runs are written (uncovered bases are implicitly zero, the bigWig convention); importing the file reconstructs full-length coverage from the header lengths.

Usage

```
bam_coverage_bigwig(
  file,
  outfile,
  param = NULL,
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE,
  n_zooms = 10L,
  compress_level = -1L,
  parallel = TRUE,
  verbose = FALSE
)
```

Arguments

file	A BAM file path, Rsamtools::BamFile , or a vector / <code>BamFileList</code> of them.
outfile	Output bigWig path(s); must have the same length as file.
param	Optional Rsamtools::ScanBamParam (or a compatible list). Only its <code>flag</code> and <code>mapqFilter</code> are honoured, applied per read in C++. With the default <code>NULL</code> the filter matches <code>readGAlignments()</code> (drop only unmapped reads).
threads	Number of OpenMP threads for within-file decoding.
BPPARAM	A BiocParallel::BiocParallelParam for across-file parallelism when file has more than one element.
auto_threads	Logical; if <code>TRUE</code> , balance the thread / worker split.
n_zooms	Maximum number of bigWig zoom levels to generate (default 10).
compress_level	Integer zlib deflate level for the bigWig blocks: -1 (default) uses zlib's default (level 6, matching other bigWig writers); 1 to 9 trade file size for speed (1 is the fastest write and gives the largest file). Coverage values are unaffected – only file size and write time change.
parallel	Logical; if <code>TRUE</code> (default), compress the bigWig data and zoom blocks across the OpenMP threads (deferred parallel compression). The output is byte-identical to serial compression. Set <code>FALSE</code> for the classic single-threaded write.
verbose	Logical; if <code>TRUE</code> , print a per-phase timing breakdown (coverage compute / interval write / zoom + index finalize) to stderr.

Details

The written values are identical to `GenomicAlignments::coverage(GenomicAlignments::readGAlignments(file))` (M/=X/D contribute, N is a gap, I/S/H/P are ignored; only unmapped reads are dropped by default). Coverage values are integers stored as the bigWig float type, which is exact for depths up to 2^{24} .

Value

The output path(s), invisibly.

See Also

[bam_coverage\(\)](#) for the in-memory RleList.

Examples

```
bam <- ompBAM::example_BAM("Unsorted")

# Single pass from BAM to a finished bigWig track
out <- tempfile(fileext = ".bw")
bam_coverage_bigwig(bam, out, threads = 2)
file.exists(out)
```

bam_read

Fast BAM reading with Bioconductor-compatible arguments

Description

`bam_read()` is a multithreaded sequential BAM reader built on top of `ompBAM`. The interface is designed to be familiar to users of `Rsamtools::scanBam()`, `GenomicAlignments::readGAlignments()`, and `GenomicAlignments::readGAlignmentPairs()`.

Usage

```
bam_read(
  file,
  param = NULL,
  what = NULL,
  tag = NULL,
  as = c("DataFrame", "data.frame", "GAlignments", "GAlignmentPairs", "scanBam"),
  seqqual_mode = c("compatible", "compact"),
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE,
  use.names = FALSE,
  with.which_label = FALSE,
  include_unmapped = TRUE
)
```

Arguments

file	A BAM input. Supported values are: • a single BAM path (<code>character(1)</code>) or multiple BAM paths, • a <code>Rsamtools::BamFile</code>, • a <code>Rsamtools::BamFileList</code>.
param	Optional <code>Rsamtools::ScanBamParam</code> (or a compatible list for lightweight use). The following fields are honored: <code>mapqFilter</code>, <code>flag</code>, <code>which</code>, <code>what</code>, and <code>tag</code>.
what	Character vector of fields to return, similar to <code>scanBam(what=...)</code>. Supported fields are <code>qname</code>, <code>flag</code>, <code>rname</code>, <code>strand</code>, <code>pos</code>, <code>qwidth</code>, <code>mapq</code>, <code>cigar</code>, <code>mrnm</code>, <code>mpos</code>, <code>isize</code>, <code>seq</code>, <code>qual</code>.
tag	Character vector of 2-letter tag names to extract.
as	Output format: • <code>"DataFrame"</code>: returns <code>S4Vectors::DataFrame</code> (default), • <code>"data.frame"</code>: returns base <code>data.frame</code>, • <code>"GAlignments"</code>: returns <code>GenomicAlignments::GAlignments</code>, • <code>"GAlignmentPairs"</code>: returns <code>GenomicAlignments::GAlignmentPairs</code>, • <code>"scanBam"</code>: returns a <code>scanBam()</code>-shaped list-of-lists.
seqqual_mode	Controls representation of <code>seq/qual</code> when those fields are requested: • <code>"compatible"</code> (default): return character vectors matching <code>scanBam</code>-style expectations, • <code>"compact"</code>: return lower-level raw list-columns for faster/lower-overhead extraction. In compact mode, <code>seq</code> is returned as one raw vector per read containing BAM-native packed sequence bytes (two bases per byte), and <code>qual</code> is returned as one raw vector per read containing per-base Phred bytes. These are not plain character strings. <code>qwidth</code> is needed to decode compact <code>seq</code> back to base letters, and <code>qual</code> values of 255 correspond to missing quality values. This mode is currently supported for <code>as = "data.frame"</code> or <code>as = "DataFrame"</code>.
threads	Requested number of OpenMP threads used for reading/decompression. May be capped when <code>auto_threads = TRUE</code>.
BPPARAM	<code>BiocParallel</code> parameter used when file contains more than one BAM. Defaults to <code>BiocParallel::bpparam()</code>. Set to <code>NULL</code> to force serial file processing.
auto_threads	Logical; when <code>TRUE</code> and <code>BPPARAM</code> has multiple workers, <code>BamScaleR</code> adaptively avoids oversubscription by preserving higher per-file OpenMP thread counts when possible and reducing the number of concurrently active file workers before shrinking per-file threads.
use.names	Passed to alignment object conversion. When <code>TRUE</code>, read names (<code>qname</code>) are used as object names.
with.which_label	Logical; if <code>TRUE</code> and <code>param</code> includes <code>which</code>, an extra <code>which_label</code> column is returned.
include_unmapped	Logical; whether unmapped records are retained (subject to <code>param\$flag</code> constraints).

Details

`bam_read()` is intentionally column-compatible with common BAM fields used by Bioconductor workflows and can be used as a fast drop-in reader before conversion to downstream classes.

Parallelism model:

- BPPARAM parallelizes across files (one file per BiocParallel worker).
- threads parallelizes within each file via OpenMP.
- Effective total concurrency is approximately `min(length(file), BiocParallel::bpnworkers(BPPARAM)) * threads`.
- If `auto_threads = TRUE` and BPPARAM has multiple workers, BamScaleR first limits the number of concurrently active workers to preserve the requested per-file thread count within the detected core budget, then caps per-file OpenMP threads only if a single file would still over-subscribe the machine.

Compatibility notes:

- `rname`, `mrnm`, and `strand` are returned as factor columns, matching `Rsamtools::scanBam()`. `rname`/`mrnm` levels are the BAM header reference names; unmapped or unset references are NA (not "*"). `strand` has levels `c("+", "-", "*")`.
- Region filtering via `param$which` is supported as a sequential filter (not index-jump random access).
- Flag filtering uses `ScanBamFlag` semantics by converting logical flag requirements into required-set and required-unset bit masks.
- Tag values are returned with their native BAM-derived type, matching `scanBam()`: integer-valued tags (`c`, `C`, `s`, `S`, `i`, `I`) become integer columns (or double when a value exceeds the R integer range), floating-point (`f`) tags become double columns, and `A/Z/H/B` tags become character columns. `B` tags are comma-separated character values. Absent tag values are NA.
- `seqqual_mode = "compact"` is optimized for throughput-oriented benchmarking and returns raw list-columns for `seq/qual`, not ordinary sequence or quality strings. In this representation, `seq` contains BAM-packed nucleotide bytes and `qual` contains raw Phred bytes. Compact output is intended for users who want to defer or avoid full string-materialization costs; use `decode_compact_seq()`, `decode_compact_qual()`, or `decode_seqqual_compact()` to decode compact output back to standard string form when needed.
- `"GAlignments"` and `"GAlignmentPairs"` output exclude unmapped records.
- `as = "scanBam"` returns a strict scan-like list-of-lists: without `param$which`, it returns one unnamed batch; with `param$which`, it returns one batch per range label (including empty ranges), with requested what fields and tag values under `$tag`. In this output mode, `seq` and `qual` are returned as `Biostrings::DNAStringSet` and `Biostrings::PhredQuality` for closer `scanBam()` compatibility.

Value

If `file` is length 1: one object in the format specified by `as`. If `file` has length > 1 (or is a `BamFileList`): a named list of outputs, one per BAM file.

Examples

```

bam <- ompBAM::example_BAM("Unsorted")

# Familiar scanBam-like field selection
x <- bam_read(bam, what = c("qname", "flag", "rname", "pos", "cigar"))

# Include sequence + quality
y <- bam_read(bam, what = c("qname", "seq", "qual"), threads = 2)

# scanBam-shaped output
z <- bam_read(bam, what = c("qname", "flag"), tag = "NM", as = "scanBam")

```

decode_compact_qual	<i>Decode compact BamScaleR quality output</i>
---------------------	--

Description

Decodes qual values returned by `bam_read(..., seqqual_mode = "compact")` back to ASCII Phred-quality strings.

Usage

```
decode_compact_qual(qual)
```

Arguments

qual	A list (or list-column) of raw vectors produced by compact BamScaleR quality extraction.
------	--

Value

A character vector containing decoded quality strings. Entries with all-missing quality bytes are returned as "*", matching BamScaleR's compatibility mode.

See Also

[decode_compact_seq\(\)](#), [decode_seqqual_compact\(\)](#), [bam_read\(\)](#)

Examples

```

decode_compact_qual(
  qual = list(as.raw(c(0L, 1L, 2L, 3L)))
)

```

decode_compact_seq	<i>Decode compact BamScaleR sequence output</i>
--------------------	---

Description

Decodes seq values returned by `bam_read(..., seqqual_mode = "compact")` back to ordinary character strings.

Usage

```
decode_compact_seq(seq, qwidth)
```

Arguments

seq	A list (or list-column) of raw vectors produced by compact BamScaleR sequence extraction.
qwidth	Integer vector of read widths. This is required because compact sequence bytes use BAM's 4-bit packed encoding (two bases per byte).

Value

A character vector containing decoded sequence strings.

See Also

[decode_compact_qual\(\)](#), [decode_seqqual_compact\(\)](#), [bam_read\(\)](#)

Examples

```
decode_compact_seq(
  seq = list(as.raw(c(0x12, 0x48))),
  qwidth = 4L
)
```

decode_seqqual_compact	<i>Decode compact seq and qual columns in BamScaleR output</i>
------------------------	--

Description

Convenience wrapper for converting a compact `bam_read()` result back to ordinary sequence and quality strings.

Usage

```
decode_sequal_compact(
  x,
  seq_col = "seq",
  qual_col = "qual",
  qwidth_col = "qwidth"
)
```

Arguments

x	A <code>data.frame</code> , <code>S4Vectors::DataFrame</code> , or list-like object containing compact BamScaleR seq and/or qual columns.
seq_col	Name of the compact sequence column.
qual_col	Name of the compact quality column.
qwidth_col	Name of the read-width column used to decode compact sequence bytes.

Value

x with compact seq and/or qual columns replaced by decoded character vectors. The input class is preserved.

See Also

[decode_compact_seq\(\)](#), [decode_compact_qual\(\)](#), [bam_read\(\)](#)

Examples

```
x <- data.frame(qwidth = 4L)
x$seq <- I(list(as.raw(c(0x12, 0x48))))
x$qual <- I(list(as.raw(c(0L, 1L, 2L, 3L))))
decode_sequal_compact(x)
```

fragment_sizes	<i>Fragment-size distribution, aggregated in C++</i>
----------------	--

Description

Computes the paired-end fragment-size (insert-size) distribution – a table of `abs(TLEN)` counts – by accumulating the histogram inside the multithreaded C++ reader, without ever materialising the per-read insert sizes as an R vector. This is the fast path for ATAC-seq / paired-end fragment-size QC on large BAMs: the histogram is folded together inside the OpenMP threads and only the compact (`fragment_size`, `count`) table crosses back into R (cf. `scanBam(what = "isize")` followed by `table(abs(isize))`, which materialises every insert size in R first).

Usage

```
fragment_sizes(
  file,
  param = NULL,
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE,
  max_fragment = 100000L,
  drop_mate_unmapped = TRUE
)
```

Arguments

file	A BAM file path, Rsamtools::BamFile , or a vector / BamFileList of them.
param	Optional Rsamtools::ScanBamParam (or a compatible list). Only its flag and mapqFilter are honoured, applied per read in C++ – e.g. <code>Rsamtools::ScanBamParam(flag = Rsamtools::scanBamFlag(isSecondaryAlignment = FALSE, isUnmappedQuery = FALSE, isNotPassingQualityControls = FALSE))</code> to reproduce the standard ATAC fragment-size filter.
threads	Number of OpenMP threads for within-file decoding.
BPPARAM	A BiocParallel::BiocParallelParam for across-file parallelism when file has more than one element.
auto_threads	Logical; if TRUE, balance the thread / worker split.
max_fragment	Integer; fragment sizes up to this value are tallied in a dense histogram and larger ones in an overflow map. Every observed size is returned exactly regardless of this value (default 1e5).
drop_mate_unmapped	Logical; if TRUE (default), reads whose mate is unmapped (SAM flag 0x8) are excluded – matching <code>Rsamtools::scanBam</code> , which reports their isize as NA (no defined template length), so that <code>table(abs(isize))</code> drops them. Set FALSE to tally <code>abs(TLEN)</code> over all flag-passing reads.

Value

A data.frame with columns fragment_size (integer `abs(TLEN)`) and count (numeric), one row per observed fragment size in ascending order. For multiple input files, a named list of such data.frames.

See Also

[bam_read\(\)](#) for reading records, [bam_count\(\)](#) for per-chromosome counts.

Examples

```
bam <- ompBAM::example_BAM("Unsorted")

# Fragment-size (insert-size) distribution computed inside the reader
```

```
fs <- fragment_sizes(bam, threads = 2)
head(fs)
```

mapq_dist

Mapping-quality (MAPQ) distribution, aggregated in C++

Description

Computes the MAPQ distribution – a table of per-value alignment-count – by accumulating the histogram inside the multithreaded C++ reader, without ever materialising the per-read MAPQ values as an R vector. This is the fast path for mapping-quality QC on large BAMs: each thread tallies MAPQ into a private 256-slot histogram inside the OpenMP region and only the compact (mapq, count) table crosses back into R (cf. scanBam(what = "mapq") followed by table(mapq), which materialises every MAPQ in R first).

Usage

```
mapq_dist(
  file,
  param = NULL,
  threads = 1L,
  BPPARAM = BiocParallel::bpparam(),
  auto_threads = FALSE
)
```

Arguments

file	A BAM file path, Rsamtools::BamFile , or a vector / BamFileList of them.
param	Optional Rsamtools::ScanBamParam (or a compatible list). Only its flag and mapqFilter are honoured, applied per read in C++. Note that a non-zero mapqFilter removes reads below that MAPQ from the distribution.
threads	Number of OpenMP threads for within-file decoding.
BPPARAM	A BiocParallel::BiocParallelParam for across-file parallelism when file has more than one element.
auto_threads	Logical; if TRUE, balance the thread / worker split.

Value

A data.frame with columns mapq (integer, 0-255) and count (numeric), one row per observed MAPQ value in ascending order. For multiple input files, a named list of such data.frames.

See Also

[fragment_sizes\(\)](#) for the fragment-size distribution, [bam_count\(\)](#) for per-chromosome counts.

Examples

```
bam <- ompBAM::example_BAM("Unsorted")

# Mapping-quality distribution computed inside the reader
mq <- mapq_dist(bam, threads = 2)
head(mq)
```

Index

`bam_count`, [2](#)
`bam_count()`, [12](#), [13](#)
`bam_coverage`, [3](#)
`bam_coverage()`, [5](#), [6](#)
`bam_coverage_bigwig`, [4](#)
`bam_read`, [6](#)
`bam_read()`, [4](#), [9–12](#)
`BiocParallel::BiocParallelParam`, [4](#), [5](#),
[12](#), [13](#)

`decode_compact_qual`, [9](#)
`decode_compact_qual()`, [10](#), [11](#)
`decode_compact_seq`, [10](#)
`decode_compact_seq()`, [9](#), [11](#)
`decode_seqqual_compact`, [10](#)
`decode_seqqual_compact()`, [9](#), [10](#)

`fragment_sizes`, [11](#)
`fragment_sizes()`, [4](#), [13](#)

`IRanges::RleList`, [4](#)

`mapq_dist`, [13](#)
`mapq_dist()`, [4](#)

`Rsamtools::BamFile`, [4](#), [5](#), [12](#), [13](#)
`Rsamtools::ScanBamParam`, [4](#), [5](#), [12](#), [13](#)