

# Package ‘beachmat’

March 29, 2021

**Version** 2.6.4

**Date** 2020-12-19

**Title** Compiling Bioconductor to Handle Each Matrix Type

**Encoding** UTF-8

**Imports** methods, DelayedArray (>= 0.15.14), BiocGenerics, Matrix

**Suggests** testthat, BiocStyle, knitr, rmarkdown, rcmdcheck,  
BiocParallel

**biocViews** DataRepresentation, DataImport, Infrastructure

**Description** Provides a consistent C++ class interface for reading from and writing data to a variety of commonly used matrix types. Ordinary matrices and several sparse/dense Matrix classes are directly supported, third-party S4 classes may be supported by external linkage, while all other matrices are handled by DelayedArray block processing.

**License** GPL-3

**NeedsCompilation** yes

**VignetteBuilder** knitr

**SystemRequirements** C++11

**RoxygenNote** 7.1.1

**git\_url** <https://git.bioconductor.org/packages/beachmat>

**git\_branch** RELEASE\_3\_12

**git\_last\_commit** 7d9dc63

**git\_last\_commit\_date** 2020-12-19

**Date/Publication** 2021-03-29

**Author** Aaron Lun [aut, cre],  
Hervé Pagès [aut],  
Mike Smith [aut]

**Maintainer** Aaron Lun <[infinite.monkeys.with.keyboards@gmail.com](mailto:infinite.monkeys.with.keyboards@gmail.com)>

## R topics documented:

colBlockApply . . . . .	2
<b>Index</b>	<b>4</b>

---

colBlockApply	<i>Apply over blocks of columns or rows</i>
---------------	---

---

### Description

Apply a function over blocks of columns or rows using **DelayedArray**'s block processing mechanism.

### Usage

```
colBlockApply(x, FUN, ..., grid = NULL, BPPARAM = getAutoBPPARAM())
```

```
rowBlockApply(x, FUN, ..., grid = NULL, BPPARAM = getAutoBPPARAM())
```

### Arguments

x	A matrix-like object to be split into blocks and looped over. This can be of any class that respects the matrix contract.
FUN	A function that operates on columns or rows in x, for colBlockApply and rowBlockApply respectively. Ordinary matrices, *gCMatrix or <a href="#">SparseArray-Seed</a> objects may be passed as the first argument.
...	Further arguments to pass to FUN.
grid	An <a href="#">ArrayGrid</a> object specifying how x should be split into blocks. For colBlockApply and rowBlockApply, blocks should consist of consecutive columns and rows, respectively. Alternatively, this can be set to TRUE or FALSE, see Details.
BPPARAM	A BiocParallelParam object from the <b>BiocParallel</b> package, specifying how parallelization should be performed across blocks.

### Details

This is a wrapper around [blockApply](#) that is dedicated to looping across rows or columns of x. The aim is to provide a simpler interface for the common task of [applying](#) across a matrix, along with a few modifications to improve efficiency for parallel processing and for natively supported x.

Note that the fragmentation of x into blocks is not easily predictable, meaning that FUN should be capable of operating on each row/column independently. Users can retrieve the current location of each block within x with [currentViewport](#) inside FUN.

If grid is not explicitly set to an [ArrayGrid](#) object, it can take several values:

- If TRUE, the function will choose a grid that (i) respects the memory limits in [getAutoBlockSize](#) and (ii) fragments x into sufficiently fine chunks that every worker in BPPARAM gets to do something. If FUN might make large allocations, this mode should be used to constrain memory usage.
- The default grid=NULL is very similar to TRUE except that that memory limits are ignored when x is of any type that can be passed directly to FUN. This avoids unnecessary copies of x and is best used when FUN itself does not make large allocations.
- If FALSE, the function will choose a grid that covers the entire x. This is provided for completeness and is only really useful for debugging.

**Value**

A list of length equal to the number of blocks, where each entry is the output of FUN for the results of processing each the rows/columns in the corresponding block.

**See Also**

[blockApply](#), for the original **DelayedArray** implementation.

**Examples**

```
x <- matrix(runif(10000), ncol=10)
str(colBlockApply(x, colSums))
str(rowBlockApply(x, rowSums))

library(Matrix)
y <- rsparsematrix(10000, 10000, density=0.01)
str(colBlockApply(y, colSums))
str(rowBlockApply(y, rowSums))

library(DelayedArray)
z <- DelayedArray(y) + 1
str(colBlockApply(z, colSums))
str(rowBlockApply(z, rowSums))

# We can also force multiple blocks:
library(BiocParallel)
BPPARAM <- SnowParam(2)
str(colBlockApply(x, colSums, BPPARAM=BPPARAM))
str(rowBlockApply(x, rowSums, BPPARAM=BPPARAM))
```

# Index

`apply`, [2](#)

`ArrayGrid`, [2](#)

`blockApply`, [2](#), [3](#)

`colBlockApply`, [2](#)

`currentViewport`, [2](#)

`getAutoBlockSize`, [2](#)

`rowBlockApply (colBlockApply)`, [2](#)

`SparseArraySeed`, [2](#)