Package ‘beachmat’

April 10, 2023

Version 2.14.2

Date 2023-04-06

Title Compiling Bioconductor to Handle Each Matrix Type

Encoding UTF-8

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

Suggests testthat, BiocStyle, knitr, rmarkdown, rcmdcheck,
BiocParallel, HDF5Array

LinkingTo Rcpp

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_16

git_last_commit bfc3e8a

git_last_commit_date 2023-04-06

Date/Publication 2023-04-10

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

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

R topics documented:

`colBlockApply` *Apply over blocks of columns or rows*

Description

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

Usage

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

rowBlockApply(
  x,
  FUN,
  ...,
  grid = NULL,
  coerce.sparse = TRUE,
  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, CsparseMatrix or SparseArraySeed objects may be passed as the first argument.
`...`	Further arguments to pass to `FUN`.
`grid`	An ArrayGrid 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.

`coerce.sparse`	Logical scalar indicating whether blocks of a sparse DelayedMatrix `x` should be automatically coerced into CsparseMatrix objects.
`BPPARAM`	A BiocParallelParam object from the **BiocParallel** 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 `apply`ing 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 of `x` by calling `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.

The default of `coerce.sparse=TRUE` will generate dgCMatrix objects during block processing of a sparse DelayedMatrix `x`. This is convenient as it avoids the need for `FUN` to specially handle SparseArraySeed objects. If the coercion is not desired (e.g., to preserve integer values in `x`), it can be disabled with `coerce.sparse=FALSE`.

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.

`toCsparse`, to convert SparseArraySeeds to CsparseMatrix objects prior to further processing in `FUN`.

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))
```

---

realizeFileBackedMatrix

Realize a file-backed DelayedMatrix

Description

Realize a file-backed DelayedMatrix into its corresponding in-memory format.

Usage

```
realizeFileBackedMatrix(x)

isFileBackedMatrix(x)
```

Arguments

x	A DelayedMatrix object.

Details

A file-backed matrix representation is recognized based on whether it has a `path` method for any one of its seeds. If so, and the `"beachmat.realizeFileBackedMatrix"` option is not `FALSE`, we will load it into memory. This is intended for DelayedMatrix objects that have already been subsetted (e.g., to highly variable genes), which can be feasibly loaded into memory for rapid calculations.

Value

For `realizeFileBackedMatrix`, an ordinary matrix or a dgCMatrix, depending on whether `is_sparse(x)`.

For `isFileBackedMatrix`, a logical scalar indicating whether `x` has file-backed components.

Author(s)

Aaron Lun

Examples

```
mat <- matrix(rnorm(50), ncol=5)
realizeFileBackedMatrix(mat) # no effect

library(HDF5Array)
mat2 <- as(mat, "HDF5Array")
realizeFileBackedMatrix(mat2) # realized into memory
```

`toCsparse` *Convert a SparseArraySeed to a CsparseMatrix*

Description

Exactly what it says in the title.

Usage

```
toCsparse(x)
```

Arguments

x	Any object produced by block processing with `colBlockApply` or `rowBlockApply`. This can be a matrix, sparse matrix or a two-dimensional SparseArraySeed.

Details

This is intended for use inside functions to be passed to `colBlockApply` or `rowBlockApply`. The idea is to pre-process blocks for user-defined functions that don't know how to deal with SparseArraySeed objects, which is often the case for R-defined functions that do not benefit from **beachmat**'s C++ abstraction.

Value

`x` is returned unless it was a SparseArraySeed, in which case an appropriate CsparseMatrix object is returned instead.

Author(s)

Aaron Lun

Examples

```
library(DelayedArray)
out <- SparseArraySeed(c(10, 10),
    nzindex=cbind(1:10, sample(10)),
    nzdata=runif(10))
toCsparse(out)
```

whichNonZero — *Find non-zero entries of a matrix*

Description

Finds the non-zero entries of a matrix in the most efficient manner for each matrix representation. Not sure there's much more to say here.

Usage

```
whichNonZero(x, ...)

## S4 method for signature 'ANY'
whichNonZero(x, ...)

## S4 method for signature 'TsparseMatrix'
whichNonZero(x, ...)

## S4 method for signature 'CsparseMatrix'
whichNonZero(x, ...)

## S4 method for signature 'SparseArraySeed'
whichNonZero(x, ...)

## S4 method for signature 'DelayedMatrix'
whichNonZero(x, BPPARAM = NULL, ...)
```

Arguments

`x`	A numeric matrix-like object, usually sparse in content if not in representation.
`...`	For the generic, additional arguments to pass to the specific methods. For the methods, additional arguments that are currently ignored.
`BPPARAM`	A BiocParallelParam object from the **BiocParallel** package controlling how parallelization should be performed. Only used when `x` is a DelayedMatrix object; defaults to no parallelization.

Value

A list containing `i`, an integer vector of the row indices of all non-zero entries; `j`, an integer vector of the column indices of all non-zero entries; and `x`, a (usually atomic) vector of the values of the non-zero entries.

Author(s)

Aaron Lun

See Also

`which`, obviously.

Examples

```
x <- Matrix::rsparsematrix(1e6, 1e6, 0.000001)
out <- whichNonZero(x)
str(out)
```

Index