Package 'beachmat'

March 25, 2024

Version 2.18.1

Date 2024-01-22

Title Compiling Bioconductor to Handle Each Matrix Type

Encoding UTF-8

Imports methods, DelayedArray (>= 0.27.2), SparseArray, 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 a variety of commonly used matrix types.
Ordinary matrices and several sparse/dense Matrix classes are directly supported, along with a subset of the delayed operations implemented in the DelayedArray package.
All other matrix-like objects are supported by calling back into R.

License GPL-3

NeedsCompilation yes

VignetteBuilder knitr

SystemRequirements C++17

URL https://github.com/tatami-inc/beachmat

BugReports https://github.com/tatami-inc/beachmat/issues

RoxygenNote 7.2.3

git_url https://git.bioconductor.org/packages/beachmat

git_branch RELEASE_3_18

git_last_commit 39ef12f

git_last_commit_date 2024-01-22

Repository Bioconductor 3.18

Date/Publication 2024-03-25

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

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

R topics documented:

`checkMemoryCache` *Check the in-memory cache for matrix instances*

Description

Check the in-memory cache for a pre-existing initialized C++ object, and initialize it if it does not exist. This is typically used in `initializeCpp` methods of file-backed representations to avoid redundant reads of the entire matrix.

Usage

```
flushMemoryCache()

checkMemoryCache(namespace, key, fun)
```

Arguments

`namespace`	String containing the namespace, typically the name of the package implementing the method.
`key`	String containing the key for a specific matrix instance.
`fun`	Function that accepts no arguments and returns an external pointer like those returned by `initializeCpp`.

Details

For representations where data extraction is costly (e.g., from file), `initializeCpp` methods may provide a `memorize=` option. Setting this to `TRUE` will load the entire matrix into memory, effectively paying a one-time up-front cost to improve efficiency for downstream operations that pass through the matrix multiple times.

If this option is provided, `initializeCpp` methods are expected to cache the in-memory instance using `checkMemoryCache`. This ensures that all subsequent calls to the same `initializeCpp`

method will return the same instance, avoiding redundant memory loads when the same matrix is used in multiple functions.

Of course, this process saves time at the expense of increased memory usage. If too many instances are being cached, they can be cleared from memory using the `flushMemoryCache` function.

Value

For `checkMemoryCache`, the output of `fun` (possibly from an existing cache) is returned.

For `flushMemoryCache`, all existing cached objects are removed and `NULL` is invisibly returned.

Author(s)

Aaron Lun

Examples

```
# Mocking up a class with some kind of uniquely identifying aspect.
setClass("UnknownMatrix", slots=c(contents="dgCMatrix", uuid="character"))
X <- new("UnknownMatrix",
    contents=Matrix::rsparsematrix(10, 10, 0.1),
    uuid=as.character(sample(1e8, 1)))

# Defining our initialization method.
setMethod("initializeCpp", "UnknownMatrix", function(x, ..., memorize=FALSE) {
    if (memorize) {
        checkMemoryCache("my_package", x@uuid, function() initializeCpp(x@contents))
    } else {
        initializeCpp(x@contents)
    }
})

# Same pointer is returned multiple times.
initializeCpp(X, memorize=TRUE)
initializeCpp(X, memorize=TRUE)

# Flushing the cache.
flushMemoryCache()
```

`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 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 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))
```

`initializeCpp` *Initialize matrix in C++ memory space*

Description

Initialize a **tatami** matrix object in C++ memory space from an abstract numeric R matrix. This object simply references the R memory space and avoids making any copies of its own, so it can be cheaply re-created when needed inside each function.

Usage

```
initializeCpp(x, ...)
```

Arguments

`x`	A matrix-like object, typically from the **Matrix** or **DelayedArray** packages.
`...`	Further arguments used by specific methods. Common arguments include: • `memorize`, a logical scalar indicating whether to load the representation into memory - see `checkMemoryCache` for details.

Details

Do not attempt to serialize the return value; it contains a pointer to external memory, and will not be valid after a save/load cycle. Users should not be exposed to the returned pointers; rather, developers should call `initialize` at the start to obtain a C++ object for further processing. As mentioned before, this initialization process is very cheap so there is no downside from just recreating the object within each function body.

Value

An external pointer to a C++ object containing a tatami matrix.

Examples

```
# Mocking up a count matrix:
x <- Matrix::rsparsematrix(1000, 100, 0.1)
y <- round(abs(x))

stuff <- initializeCpp(y)
stuff
```

`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