Package 'beachmat'

January 3, 2025

Version 2.23.6 Date 2024-12-20

Title Compiling Bioconductor to Handle Each Matrix Type

Imports methods, DelayedArray (>= 0.27.2), SparseArray, BiocGenerics, Matrix, Rcpp

Suggests testthat, BiocStyle, knitr, rmarkdown, rcmdcheck, BiocParallel, HDF5Array, beachmat.hdf5

LinkingTo Rcpp, assorthead

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 7.3.2

Encoding UTF-8

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

git_branch devel

git_last_commit a94957b

git_last_commit_date 2024-12-20

Repository Bioconductor 3.21

Date/Publication 2025-01-03

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

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

Contents

| checkMemoryCache | . 2 |
|-------------------------|------|
| colBlockApply | . 3 |
| initializeCpp | . 6 |
| realizeFileBackedMatrix | . 7 |
| tatami-utils | . 8 |
| toCsparse | . 11 |
| whichNonZero | . 12 |
| | |
| | 13 |

Index

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 implement- ing 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.

colBlockApply

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",</pre>
    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 App

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(
  х,
  FUN,
  ...,
  grid = NULL,
  coerce.sparse = TRUE,
 BPPARAM = getAutoBPPARAM()
)
rowBlockApply(
  х,
  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 SparseMatrix 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 par- allelization 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:

4

colBlockApply

- 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 Sparse-Matrix objects from the **SparseArray** package. 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 SparseMatrix objects to CsparseMatrix objects prior to further processing in FUN.

```
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)
PDPAtM <_ Capabaray(2)</pre>
```

```
library(BiocParallel)
BPPARAM <- SnowParam(2)
str(colBlockApply(x, colSums, BPPARAM=BPPARAM))
str(rowBlockApply(x, rowSums, BPPARAM=BPPARAM))</pre>
```

initializeCpp

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. Alternatively, an external pointer from a previous call to initializeCpp, which is returned without modification. |
|---|--|
| | Further arguments used by specific methods, such as: |
| | • .check.na, a logical vector indicating whether to check for NA values in integer and logical matrices. If TRUE (the default), any NAs are cast to their double-precision equivalents when reading from the tatami matrix. This can be set to FALSE to improve performance if the caller knows that x does not contain NAs. |
| | Fields should generally be prefixed by the matrix type, to avoid conflicts with arguments from other packages. For example, hdf5.realize can be used in beachmat.hdf5 to load a HDF5-backed matrix into memory. |

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.

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

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 ×

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

```
mat <- matrix(rnorm(50), ncol=5)
realizeFileBackedMatrix(mat) # no effect
library(HDF5Array)</pre>
```

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

tatami-utils

Description

Utility functions that directly operate on the pointers produced by initializeCpp. Some of these are used internally by initializeCpp methods operating on **DelayedArray** classes.

Usage

```
tatami.bind(xs, by.row)
tatami.transpose(x)
tatami.subset(x, subset, by.row)
tatami.arith(x, op, val, by.row, right)
tatami.compare(x, op, val, by.row, right)
tatami.logic(x, op, val, by.row)
tatami.round(x)
tatami.log(x, base)
tatami.math(x, op)
tatami.not(x)
tatami.binary(x, y, op)
tatami.dim(x)
tatami.row(x, i)
tatami.column(x, i)
tatami.row.sums(x, num.threads)
tatami.column.sums(x, num.threads)
tatami.row.nan.counts(x, num.threads)
tatami.column.nan.counts(x, num.threads)
tatami.is.sparse(x)
```

tatami.prefer.rows(x)

tatami.realize(x, num.threads)

tatami.multiply(x, val, right, num.threads)

Arguments

| XS | A list of pointers produced by initializeCpp. All matrices should have the same number of rows (if by.row=FALSE) or columns (otherwise). |
|--------|---|
| by.row | Logical scalar indicating whether to apply the operation on the rows. |
| | • For tatami.bind, this will combine the matrices by rows, i.e., the output matrix has a number of rows equal to the sum of the number of rows in xs. |
| | For tatami.subset, this will subset the matrix by row. |
| | • For tatami.arith, tatami.compare and tatami.logic with a vector val, the vector should have length equal to the number of rows.k |
| Х | A pointer produced by initializeCpp. |
| subset | Integer vector containing the subset of interest. These should be 1-based row or column indices depending on by.row. |
| ор | String specifying the operation to perform. |
| | • For tatami.arith, this should be one of the operations in Arith. |
| | • For tatami.compare, this should be one of the operations in Compare. |
| | • For tatami.logic, this should be one of the operations in Logic. |
| | • For tatami.math, this should be one of the operations in Math. |
| | • For tatami.binary, this may be any operation in Arith, Compare or Logic. |
| val | For tatami.arith, tatami.compare and tatami.logic, the value to be used in the operation specified by op. This may be a: |
| | • Numeric scalar, which is used in the operation for all entries of the matrix. |
| | • Numeric vector of length equal to the number of rows, where each value is used in the operation with the corresponding row when by.row=TRUE. |
| | • Numeric vector of length equal to the number of column, where each value is used with the corresponding column when by.row=FALSE. |
| | For tatami.multiply, the value to be used in the matrix multiplication. This may be a: |
| | • Numeric vector of length equal to the number of columns of x (if right=FALSE) or rows (otherwise). |
| | • Numeric matrix with number of rows equal to the number of columns of x (if right=FALSE) or rows (otherwise). |
| | • Pointer produced by initializeCpp, referencing a matrix with number of rows equal to the number of columns of x (if right=FALSE) or rows (otherwise). |

| right | For tatami.arith and tatami.compare, a logical scalar indicating that val is on the right-hand side of the operation. |
|-------------|---|
| | For tatami.multiply, a logical scalar indicating that val is on the right-hand side of the multiplication. |
| base | Numeric scalar specifying the base of the log-transformation. |
| У | A pointer produced by initializeCpp, referencing a matrix of the same di- mensions as x. |
| i | Integer scalar containing the 1-based index of the row (for tatami.row) or col- umn (for tatami.column) of interest. |
| num.threads | Integer scalar specifying the number of threads to use. |

Value

For tatami.dim, an integer vector containing the dimensions of the matrix.

For tatami.is.sparse, a logical scalar indicating whether the matrix is sparse.

For tatami.prefer.rows, a logical scalar indicating whether the matrix prefers iteration by row.

For tatami.row or tatami.column, a numeric vector containing the contents of row or column i, respectively.

For tatami.row.sums or tatami.column.sums, a numeric vector containing the row or column sums, respectively.

For tatami.row.nan.counts or tatami.column.nan.counts, a numeric vector containing the number of NaNs in each row or column, respectively.

For tatami.realize, a numeric matrix or dgCMatrix with the matrix contents. The exact class depends on whether x refers to a sparse matrix.

For tatami.multiply, a numeric matrix containing the matrix product of x and other.

For all other functions, a new pointer to a matrix with the requested operations applied to x or xs.

Author(s)

Aaron Lun

Examples

```
x <- Matrix::rsparsematrix(1000, 100, 0.1)
ptr <- initializeCpp(x)
tatami.dim(ptr)
tatami.row(ptr, 1)
rounded <- tatami.round(ptr)</pre>
```

tatami.row(rounded, 1)

toCsparse

Description

Exactly what it says in the title.

Usage

toCsparse(x)

Arguments

х

Any object produced by block processing with colBlockApply or rowBlockApply. This can be a matrix, sparse matrix or a SparseMatrix object from the **SparseAr**ray package.

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 Sparse-Matrix 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 is a **SparseMatrix** object from the **SparseArray** package, in which case an appropriate CsparseMatrix object is returned instead.

Author(s)

Aaron Lun

```
library(SparseArray)
out <- COO_SparseArray(c(10, 10),
    nzcoo=cbind(1:10, sample(10)),
    nzdata=runif(10))
toCsparse(out)</pre>
```

whichNonZero

Description

This function is soft-deprecated; users are advised to use nzwhich and nzvals instead.

Usage

```
whichNonZero(x, ...)
```

Arguments

| Х | A numeric matrix-like object, usually sparse in content if not in representation |
|---|--|
| | Further arguments, ignored. |

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.

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

Index

```
apply, 4
                                               initializeCpp,dgCMatrix-method
Arith, 9
                                                        (initializeCpp), 6
ArrayGrid, 4
                                                initializeCpp,dgeMatrix-method
                                                        (initializeCpp), 6
blockApply, 4, 5
                                                initializeCpp,dgRMatrix-method
                                                        (initializeCpp), 6
checkMemoryCache, 2
                                                initializeCpp, externalptr-method
colBlockApply, 3, 11
                                                        (initializeCpp), 6
Compare, 9
                                                initializeCpp,lgCMatrix-method
CsparseMatrix, 4, 11
                                                        (initializeCpp), 6
currentViewport, 4
                                                initializeCpp,lgeMatrix-method
                                                        (initializeCpp), 6
DelayedMatrix, 4, 7
                                                initializeCpp,lgRMatrix-method
dgCMatrix, 5, 7, 10
                                                        (initializeCpp), 6
                                                initializeCpp,matrix-method
flushMemoryCache (checkMemoryCache), 2
                                                        (initializeCpp), 6
                                                initializeCpp,SVT_SparseMatrix-method
getAutoBlockSize, 5
                                                        (initializeCpp), 6
                                                is_sparse, 7
initializeCpp, 2, 6, 8–10
                                                isFileBackedMatrix
initializeCpp,ANY-method
                                                        (realizeFileBackedMatrix), 7
        (initializeCpp), 6
initializeCpp,ConstantArraySeed-method
                                               Logic, 9
        (initializeCpp), 6
                                               Math, 9
initializeCpp, DelayedAbind-method
        (initializeCpp), 6
                                               nzvals, 12
initializeCpp,DelayedAperm-method
                                               nzwhich, 12
        (initializeCpp), 6
initializeCpp,DelayedMatrix-method
                                               path, 7
        (initializeCpp), 6
initializeCpp,DelayedNaryIsoOp-method
                                                realizeFileBackedMatrix, 7
        (initializeCpp), 6
                                                rowBlockApply, 11
initializeCpp, DelayedSetDimnames-method
                                                rowBlockApply (colBlockApply), 3
        (initializeCpp), 6
initializeCpp,DelayedSubset-method
                                                SparseMatrix, 4, 5, 11
        (initializeCpp), 6
initializeCpp,DelayedUnaryIsoOpStack-method tatami-utils, 8
        (initializeCpp), 6
                                                tatami.arith(tatami-utils), 8
initializeCpp,DelayedUnaryIsoOpWithArgs-methodatami.binary(tatami-utils),8
        (initializeCpp), 6
                                               tatami.bind(tatami-utils), 8
```

INDEX

```
tatami.column(tatami-utils), 8
tatami.compare(tatami-utils), 8
tatami.dim(tatami-utils), 8
tatami.is.sparse(tatami-utils), 8
tatami.log(tatami-utils), 8
tatami.logic(tatami-utils), 8
tatami.math(tatami-utils), 8
tatami.multiply(tatami-utils), 8
tatami.not(tatami-utils), 8
tatami.prefer.rows(tatami-utils), 8
tatami.realize(tatami-utils), 8
tatami.round(tatami-utils), 8
tatami.row(tatami-utils), 8
tatami.subset(tatami-utils), 8
tatami.transpose (tatami-utils), 8
toCsparse, 5, 11
```

which, *12* whichNonZero, *12*

14