Package 'HiPLARM'

February 19, 2015

Type Package

Title High Performance Linear Algebra in R

Version 0.1

Date 2012-09-12

Author Peter Nash and Vendel Szeremi

Acknowledgements Martin Maechler and Douglas Bates (Matrix package)

Maintainer Giovanni Montana <g.montana@imperial.ac.uk>

License GPL (>= 2)

Depends R (>= 2.14), Matrix, methods

Description Provides multi-core or GPU support (or both if the system has GPU and multi-core CPU) for the recommended R package, Matrix.

OS_type unix

SystemRequirements BLAS/LAPACK libraries, PLASMA library , NVIDIA CUDA toolkit, MAGMA library

URL http://www.hiplar.org , http://icl.cs.utk.edu/magma/ , http://icl.cs.utk.edu/plasma/, http://www.nvidia.com/object/cuda_home_new.html

Repository CRAN

Date/Publication 2012-10-18 18:25:17

NeedsCompilation yes

R topics documented:

`checkFile` *Startup function that reads in the optimised crossover points*

Description

This function is run automatically at load time and is the function that reads in the already generated optimised crossover points or a default value.

Usage

```
checkFile()
```

Details

This function is run automatically at load time and is the function that reads in the already generated optimised crossover points. If these do not exist a default value is read in. The user is also informed of how many functions have been optimised.

Examples

```
checkFile()
```

`chol` *Cholesky Decomposition using GPU or multi-core CPU*

Description

Compute the Cholesky factorization of a real symmetric positive-definite square matrix using GPU or multi-core CPU.

Usage

```
chol(x, ...)
## S4 method for signature 'dpoMatrix'
chol(x, pivot = FALSE, ...)
```

Arguments

`x`	if `x` is not positive definite, an error is signalled.
`pivot`	logical indicating if pivoting is used. This is not supported for the GPU implementation
`...`	potentially further arguments passed to methods.

Details

For further details on classes and methods see the full Matrix package documentation.

Methods

chol `signature(x = "dgeMatrix")`: works via `"dpoMatrix"`

chol `signature(x = "dpoMatrix")`: Returns (and stores) the Cholesky decomposition of `x`, via LAPACK routines `dlacpy` and either `magma_dpotrf` or `PLASMA_dpotrf`.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
showMethods(chol, inherited = FALSE) # show different methods

sy2 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, NA,32,77))
(c2 <- chol(sy2))#-> "Cholesky" matrix
stopifnot(all.equal(c2, chol(as(sy2, "dpoMatrix")), tol= 1e-13))
str(c2)

## An example where chol() can't work
(sy3 <- new("dsyMatrix", Dim = as.integer(c(2,2)), x = c(14, -1, 2, -7)))
try(chol(sy3)) # error, since it is not positive definite
```

`chol2inv-methods` *Inverse from Cholesky*

Description

Invert a symmetric, positive definite square matrix from its Cholesky decomposition. Equivalently, compute $(X'X)^{-1}$ from the (R part) of the QR decomposition of X.
Even more generally, given an upper triangular matrix R, compute $(R'R)^{-1}$.

Usage

```
## S4 method for signature 'dtrMatrix'
chol2inv(x, ...)
```

Arguments

`x`	a matrix(-like) object; see below.
`...`	not used here; for compatibility with other methods.

Methods

x = "dtrMatrix" method for the numeric triangular matrices, built on the MAGMA `magma_dpotri` and PLASMA `PLASMA_dpotri`.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
(M  <- Matrix(cbind(1, 1:3, c(1,3,7))))
(cM <- chol(M)) # a "Cholesky" object, inheriting from "dtrMatrix"
chol2inv(cM) %*% M # the identity
stopifnot(all(chol2inv(cM) %*% M - Diagonal(nrow(M))) < 1e-10)
```

crossprod *Crossproduct using GPU or multi-core CPU*

Description

Given matrices x and y as arguments, return a matrix cross-product. This is formally equivalent to (but usually slightly faster than) the call `t(x) %*% y` (crossprod) or `x %*% t(y)` (tcrossprod).

Usage

```
crossprod(x, y)
## S4 method for signature 'dgeMatrix'
crossprod(x, y)
## S4 method for signature 'dtrMatrix'
crossprod(x, y)
```

Arguments

x	dense matrix or vector represented as a single row matrix.
y	dense matrix or vector represented as single row matrix.

Details

For further details on classes and methods see the full Matrix package documentation.

Methods

crossprod (signature(x = "dgeMatrix", y = "missing"): Calls CUBLAS function cublasDsyrk for GPU enabled systems and PLASMA_dsyrk for multi-core systems. Library settings also affect choice between GPU and CPU. If y = NULL, then it is taken to be the same matrix as x.

crossprod (signature(x = "dgeMatrix", y = "dgeMatrix"): Calls MAGMA function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU.

crossprod (signature(x = "dgeMatrix", y = "Matrix"): Calls MAGMA function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU.

crossprod (signature(x = "dgeMatrix", y = "numeric"): Calls MAGMA function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU. y is coerced to a base::matrix

crossprod (signature(x = "dgeMatrix", y = "matrix"): Calls MAGMA function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU.

crossprod (signature(x = "dtrMatrix", y = "dtrMatrix"): Calls the CUBLAS function cublasDtrmm for GPU enabled systems and PLASMA_dtrmm for multi-core systems. Library settings also affect choice between GPU and CPU.

crossprod (signature(x = "dtrMatrix", y = "ddenseMatrix"): y inherits from virtual class ddenseMatrix Calls CUBLAS function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU.

crossprod (signature(x = "dtrMatrix", y = "matrix"): Calls CUBLAS function cublasDtrmm for GPU enabled systems and PLASMA_dtrmm for multi-core systems. Library settings also affect choice between GPU and CPU.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
m <- matrix(0, 400, 500)
set.seed(12)
m[runif(314, 0, length(m))] <- 1
mm <- as(m, "dgeMatrix")
object.size(m) / object.size(mm) # smaller by a factor of > 200

## tcrossprod() is very fast:
system.time(tCmm <- tcrossprod(mm))
system.time(cm <- crossprod(t(m)))
system.time(cm. <- tcrossprod(m))

stopifnot(cm == as(tCmm, "matrix"))
```

---

determinant *Calculate the determinant using GPU and multi-core CPU*

Description

Estimate the determinant of a matrix using the MAGMA library for GPU or PLASMA library for multi-core CPUs.

Usage

```
## S4 method for signature 'dgeMatrix'
determinant(x, logarithm, ...)
## S4 method for signature 'dgeMatrix'
determinant(x, logarithm, ...)
```

Arguments

x	an R object of type dgeMatrix.
logarithm	logical; if TRUE (default) return the logarithm of the modulus of the determinant.
...	No further arguments at present.

Details

Uses the LU decomposition using magma_dgetrf or PLASMA_dgetrf.For further details and methods see the Matrix package documentation or indeed the base package.

Methods

determinant signature(x = "Matrix", logarithm = "missing"): and

determinant signature(x = "Matrix", logarithm = "logical"):compute the (log) determinant of x. The method chosen depends on the actual Matrix class of x. Note that base::det also works for all our matrices, calling the appropriate determinant() method. The Matrix::det is an exact copy of base::det, but in the correct namespace, and hence calling the S4-aware version of determinant().

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
slotNames("Matrix")

cl <- getClass("Matrix")
names(cl@subclasses) # more than 40 ..

showClass("Matrix")#> output with slots and all subclasses

(M <- Matrix(c(0,1,0,0), 6, 4))
dim(M)
diag(M)
cm <- M[1:4,] + 10*Diagonal(4)
diff(M)
## can reshape it even :
dim(M) <- c(2, 12)
M
stopifnot(identical(M, Matrix(c(0,1,0,0), 2,12)),all.equal(det(cm),
determinant(as(cm,"matrix"), log=FALSE)$modulus, check.attr=FALSE))
```

HiPLARM *HiPLARM*

Description

Multi-core and GPU support for linear algebra functions in the recommended R package Matrix.

Details

Package:	HiPLARM
Type:	Package
Version:	0.1
Date:	2012-08-04
License:	GPL(>=2)

The HiPLARM package can be used in exactly the same manner as the Matrix package on which it depends. The functionality maps that of the Matrix package exactly bar some specialised functions which are documented here. These Optimise functions are only run once to allow HiPLARM to be optimised for particular systems.

Author(s)

Peter Nash, Vendel Szeremi

Maintainer: Giovanni Montana <support@hiplar.org>

References

http://icl.cs.utk.edu/plasma

http://icl.cs.utk.edu/magma/

Song, F., Tomov, S., Dongarra, J. (2012) *Enabling and Scaling Matrix Computations on Heterogeneous Multi-Core and Multi-GPU Systems,* "26th ACM International Conference on Supercomputing (ICS 2012), ACM, San Servolo Island, Venice, Italy"

Kurzak, J., Luszczek, P., Faverge, M., Dongarra, J. (April 2012) *Programming the LU Factorization for a Multicore System with Accelerators* "Proceedings of VECPAR 12, Kobe, Japan"

Haidar, A., Ltaief, H., Dongarra, J. (2011) *Parallel Reduction to Condensed Forms for Symmetric Eigenvalue Problems using Aggregated Fine-Grained and Memory-Aware Kernels,* "Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC11), Seattle, WA"

Examples

```
library(HiPLARM)
x <- Matrix(rnorm(3 * 3), ncol = 3)
y <- Matrix(rnorm(3 * 3), ncol = 3)
```

```
z <- x%*%y
y <- lu(z)
```

`hiplarSet` *~~ Methods for Function* `hiplarSet` *in Package* **HiPLARM** *~~*

Description

`hiplarSet` allows the user to access internal variables within HiPLAR, these are generally set automatically but should the user wish they can over ride these settings by following the instructions below. The `hiplarSet` function is useful if users want to choose either the MAGMA or PLASMA libraries or if they wish to set the crossover point within the function. The crossover point (for users with GPU and CPU support) is the matrix size at which the function switches between the PLASMA, CPU libraries and the MAGMA, GPU libraries. This is generally automatic but the user may change if they so wish.

Details

The `var` argument accesses the particular settings that the user wishes to change In setting the xover_<function_name> the `val` argument should be set to a suitable value. These values are generally set automatically during setup so the user should be wary when changing it. When accessing the library setting the user should choose values from 1 to 3. 1 sets the PLASMA library for use, 2 sets the MAGMA library and 3 sets it to automatic.

Methods

```
signature(var = "character", val = "numeric")
```

Examples

```
# Sets the PLASMA libary to be used exclusively #
hiplarSet("hiplar_library",  1)
# Sets the MAGMA library to be used exclusively #
hiplarSet("hiplar_library", 2)
# Enables autotune which selects PLASMA or MAGMA depending on the problem size.#
hiplarSet("hiplar_library", 3)

## Methods for setting crossover values ##
optsize <- 512
hiplarSet("xover_dgeMatrix_LU", optsize)
hiplarSet("xover_dgeMatrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_dgeMatrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_matrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_determinant", optsize)
hiplarSet("xover_dgeMatrix_matrix_mm", optsize)
hiplarSet("xover_dgeMatrix_matrix_mm",optsize)
hiplarSet("xover_dgeMatrix_norm",optsize)
hiplarSet("xover_dgeMatrix_solve", optsize)
```

```
hiplarSet("xover_dgeMatrix_matrix_solve", optsize)
hiplarSet("xover_dgeMatrix_rcond", optsize)
hiplarSet("xover_dgeMatrix_LU", optsize)
hiplarSet("xover_dgeMatrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_dgeMatrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_matrix_crossprod", optsize)
hiplarSet("xover_dgeMatrix_determinant", optsize)
hiplarSet("xover_dgeMatrix_matrix_mm", optsize)
hiplarSet("xover_dgeMatrix_matrix_mm",optsize)
hiplarSet("xover_dgeMatrix_norm",optsize)
hiplarSet("xover_dgeMatrix_solve", optsize)
hiplarSet("xover_dgeMatrix_matrix_solve", optsize)
hiplarSet("xover_dgeMatrix_rcond", optsize)
hiplarSet("xover_dpoMatrix_chol",optsize)
hiplarSet("xover_dpoMatrix_rcond", optsize)
hiplarSet("xover_dpoMatrix_solve", optsize)
hiplarSet("xover_dpoMatrix_dgeMatrix_solve", optsize)
hiplarSet("xover_dpoMatrix_matrix_solve", optsize)
hiplarSet("xover_dtrMatrix_chol2inv", optsize)
hiplarSet("xover_dtrMatrix_dtrMatrix_mm", optsize)
hiplarSet("xover_dtrMatrix_matrix_mm", optsize)
hiplarSet("xover_dtrMatrix_solve", optsize)
hiplarSet("xover_dtrMatrix_matrix_solve", optsize)
hiplarSet("xover_dsyMatrix_matrix_mm", 0)
hiplarSet("xover_dsyMatrix_norm", 0)
```

`hiplarShow` *Shows the crossover points for all functions*

Description

Shows crossover points for all the functions.

Usage

```
hiplarShow()
```

Details

No specific details required. This function simply shows the user what the crossover points are for the different functions.

Examples

```
hiplarShow()
```

lu *(Generalized) Triangular Decomposition of a Matrix*

Description

Computes (generalized) triangular decompositions of square and other dense matrices using the MAGMA GPU library or the PLASMA library for multi-core CPUs.

Usage

```
lu(x, ...)
## S4 method for signature 'dgeMatrix'
lu(x, warnSing = TRUE, ...)
```

Arguments

x	a dense matrix. No missing values or IEEE special values are allowed.
warnSing	(when x is a denseMatrix logical specifying if a warning should be signalled when x is singular.
...	further arguments passed to or from other methods.

Details

lu() is a generic function with special methods for different types of matrices. Use showMethods("lu") to list all the methods for the lu generic.

The method for class dgeMatrix (and all dense matrices) is based on the MAGMA "magma_dgetrf" subroutine and PLASMA "PLASMA_dgetrf". It returns a decomposition also for singular and non-square matrices. For further details on classes etc. see the Matrix package documentation.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
##--- Dense  ------------------------
x <- Matrix(rnorm(9), 3, 3)
lu(x)
dim(x2 <- round(10 * x[,-3]))# non-square
expand(lu2 <- lu(x2))
```

norm *Matrix Norms*

Description

Computes a matrix norm of x, using MAGMA for GPUs or PLASMA for multi-core CPUs. The norm can be the one norm, the infinity norm, the Frobenius norm, or the maximum modulus among elements of a matrix, as determined by the value of type. Not all norms are supported by MAGMA or PLASMA and these are documented below.

Usage

```
## S4 method for signature 'dsyMatrix'
norm(x, type, ...)
## S4 method for signature 'dgeMatrix'
norm(x, type, ...)
## S4 method for signature 'dpoMatrix'
norm(x, type, ...)
## S4 method for signature 'ldenseMatrix'
norm(x, type, ...)
## S4 method for signature 'ndenseMatrix'
norm(x, type, ...)
```

Arguments

x	a real matrix.
type	A character indicating the type of norm desired. All these norms are supported in the latest PLASMA library (v2.4.6 asof writing).
	"O", "o" **or** "1" specifies the one norm, (maximum absolute column sum); This is not supported in MAGMA currently.
	"I" **or** "i" specifies the infinity norm (maximum absolute row sum);
	"F" **or** "f" specifies the Frobenius norm (the Euclidean norm of x treated as if it were a vector); This is not supported in PLASMA versions < 2.4.6 or in MAGMA.
	"M" **or** "m" specifies the maximum modulus of all the elements in x. This is not supported in MAGMA for dgeMatrix, but is for symmetric matrices dpoMatrix and dsyMatrix.
	The default is "O". Only the first character of type[1] is used.
...	further arguments passed to or from other methods.

Details

When or if the GPU is used magmablas_dlange for dgeMatrix is called or magmablas_dlansy is used for symmetric dsyMatrix or dpoMatrix. When the multi-core library PLASMA is used PLASMA_dlange and PLASMA_dlansy are used for the respective matrix types mentioned previously.

Value

A numeric value of class "norm", representing the quantity chosen according to type.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
x <- Hilbert(9)
norm(x, "1")
norm(x, "I")
norm(x, "F")
norm(x, "M")
```

OptimiseAll *Optimise all given routines*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for all routines in this package and saves it for future use.

Usage

```
OptimiseAll(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs through each routine testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseAll(256,TRUE)
```

OptimiseChol *Optimise the chol routine for dpo Matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for chol function in this package and saves it for future use.

Usage

```
OptimiseChol(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the chol function, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseChol(256,TRUE)
```

OptimiseChol2invDtr *Optimise the chol2inv routine for dtr matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for chol2inv function in this package and saves it for future use.

Usage

```
OptimiseChol2invDtr(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the chol2inv function, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseChol2invDtr(256,TRUE)
```

OptimisecrossprodDge *Optimise the crossprod routine for a single dge matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for crossprod function in this package and saves it for future use.

Usage

```
OptimisecrossprodDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the crossprod function, for a dge matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisecrossprodDge(256,TRUE)
```

`OptimisecrossprodDgeDge` *Optimise the crossprod routine for two dge matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for crossprod function in this package and saves it for future use.

Usage

```
OptimisecrossprodDgeDge(increment = 128, verbose = FALSE)
```

Arguments

`increment`	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
`verbose`	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the crossprod function, for two dge matrices, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisecrossprodDgeDge(256,TRUE)
```

OptimisecrossprodDgemat *Optimise the crossprod routine for a dge matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for crossprod function in this package and saves it for future use.

Usage

```
OptimisecrossprodDgemat(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the crossprod function, for a dge matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisecrossprodDgemat(256,TRUE)
```

OptimisedetDge *Optimise the det routine for dge matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for det function in this package and saves it for future use.

Usage

```
OptimisedetDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the det function for dge matrices, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisedetDge(256,TRUE)
```

OptimiseLU *Optimise the LU routine for dge Matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for lu function in this package and saves it for future use.

Usage

```
OptimiseLU(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the lu function, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseLU(256,TRUE)
```

`OptimisematmulDgeDge` *Optimise the matmul routine for two dge matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for matrix multiplication function in this package and saves it for future use.

Usage

```
OptimisematmulDgeDge(increment = 128, verbose = FALSE)
```

Arguments

`increment`	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
`verbose`	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the matrix multiplication function, for two dge matrices, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisematmulDgeDge(256,TRUE)
```

OptimisematmulDgemat *Optimise the matmul routine for a dge matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for matrix multiplication function in this package and saves it for future use.

Usage

```
OptimisematmulDgemat(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the matrix multiplication function, for a dge matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisematmulDgemat(256,TRUE)
```

OptimisematmulDtrDtr *Optimise the matmul routine for two dtr matrices*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for matrix multiplication function in this package and saves it for future use.

Usage

```
OptimisematmulDtrDtr(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the matrix multiplication function, for two dtr matrices, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisematmulDtrDtr(256,TRUE)
```

OptimisematmulDtrmat *Optimise the matmul routine for a dtr matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for matrix multiplication function in this package and saves it for future use.

Usage

```
OptimisematmulDtrmat(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the matrix multiplication function, for a dtr matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisematmulDtrmat(256,TRUE)
```

OptimisenormDge *Optimise the norm routine for a dge matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for norm function in this package and saves it for future use.

Usage

```
OptimisenormDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the norm function, for a dge matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisenormDge(256,TRUE)
```

OptimisercondDge *Optimise the rcond routine for a dge matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for rcond function in this package and saves it for future use.

Usage

```
OptimisercondDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the rcond function, for a dge matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisercondDge(256,TRUE)
```

OptimisercondDpo *Optimise the rcond routine for a dpo matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for rcond function in this package and saves it for future use.

Usage

```
OptimisercondDpo(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the rcond function, for a dpo matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimisercondDpo(256,TRUE)
```

OptimiseSolveDge *Optimise the solve routine for a dge matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dge matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDge(256,TRUE)
```

`OptimiseSolveDgemat` *Optimise the solve routine for a dge matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for Solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDgemat(increment = 128, verbose = FALSE)
```

Arguments

`increment`	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
`verbose`	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dge matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDgemat(256,TRUE)
```

OptimiseSolveDpo *Optimise the solve routine for a dpo matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDpo(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dpo matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDpo(256,TRUE)
```

OptimiseSolveDpoDge *Optimise the solve routine for a dpo matrix and a dge matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for Solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDpoDge(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dge matrix and a dpo matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDpoDge(256,TRUE)
```

OptimiseSolveDpomat *Optimise the solve routine for a dpo matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDpomat(increment = 128, verbose = FALSE)
```

Arguments

increment	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
verbose	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dpo matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDpomat(256,TRUE)
```

`OptimiseSolveDtr` *Optimise the solve routine for a dtr matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDtr(increment = 128, verbose = FALSE)
```

Arguments

`increment`	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
`verbose`	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dtr matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDtr(256,TRUE)
```

`OptimiseSolveDtrmat` *Optimise the solve routine for a dtr matrix and an R base matrix*

Description

This computes the optimal crossover point between the PLASMA (CPU) and MAGMA (GPU) libraries. For smaller matrices PLASMA is generally used and for larger matrices MAGMA will be used. This function calculates this for solve function in this package and saves it for future use.

Usage

```
OptimiseSolveDtrmat(increment = 128, verbose = FALSE)
```

Arguments

`increment`	This is the step between the problem sizes being calculate. We compare the timings at different problem sizes for the PLASMA and MAGMA implementations. This is the size between those problem sizes. A smaller increment will give a more accurate crossover point, however, 128 should be sufficient.
`verbose`	This displays the timing information for the operations if it is set to TRUE. It is FALSE by default.

Details

This simply runs the solve function, for a dtr matrix and an R base matrix, testing it numerous times at different problem sizes. It benchmarks the PLASMA and MAGMA libraries against each other. For smaller problem sizes PLASMA will be faster, as the size increases the MAGMA library will become more optimal. This routine will find and save that optimal point for future use.

Examples

```
OptimiseSolveDtrmat(256,TRUE)
```

`rcond` *Estimate the Reciprocal Condition Number using GPU and multi-core CPU*

Description

Estimate the reciprocal of the condition number of a matrix using the MAGMA library for GPU or PLASMA library for multi-core CPUs.

Usage

```
## S4 method for signature 'dgeMatrix'
rcond(x, norm, ...)
## S4 method for signature 'dpoMatrix'
rcond(x, norm, ...)
```

Arguments

x	an R object that inherits from the Matrix class.
norm	Character indicating the type of norm to be used in the estimate. The default is "O" for the 1-norm ("O" is equivalent to "1"). The other possible value is "I" for the infinity norm, see also norm.
...	further arguments passed to or from other methods.

Details

Uses the LU decomposition using magma_dgetrf or PLASMA_dgetrf with the respective MAGMA or PLASMA norms if available (see norm documentation for more info).It also calls LAPACK dgecon as this is not supported in MAGMA/PLASMA. For further details on classes and methods see the Matrix package documentation.

Value

An estimate of the reciprocal condition number of x.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
x <- Matrix(rnorm(9), 3, 3)
rcond(x)
## typically "the same" (with more computational effort):
1 / (norm(x) * norm(solve(x)))
rcond(Hilbert(9))  # should be about 9.1e-13

## For non-square matrices:
rcond(x1 <- cbind(1,1:10))# 0.05278
rcond(x2 <- cbind(x1, 2:11))# practically 0, since x2 does not have full rank
```

setGPU ~~ *Methods for Function* setGPU *in Package* **HiPLARM** ~~

Description

~~ Methods for function setGPU in package **HiPLARM** ~~

Usage

```
setGPU(proc, interface)
```

Arguments

proc	A numeric or logical value that enables or disable the argument passed to interface, i.e the CPU or GPU interface or if no argument is passed to interface then the MAGMA library is enabled or disabled.
interface	A character string of either "CPU" or "GPU" determining which interface to use. If no argument is given the MAGMA library is disabled.

Details

The MAGMA library provides two different interfaces to the GPU. The "CPU" interface automatically transfers the data to and from the GPU as well as choosing the multi-gpu option automatically should the user have multiple GPUs. This is the most flexible or options and should it be available is used by default. However, some MAGMA functions are only implemented for the "GPU" interface. In this case the data transfers and other options must be given explicitly for the function call. If this is the case, this is the default option, however, at an R level the user will see no difference. We also provide the option of disabling the GPU and using the regular LAPACK call should the user wish to test results.

Methods

`signature(proc = "logical", interface = "character")` This allows the user to choose the CPU or GPU interface. Should the user wish to enable a certain interface the string "CPU" or "GPU" are passed as the `interface` argument and the `proc` argument is set to `TRUE` or `FALSE`.

`signature(proc = "logical", interface = "missing")` This allows the user to enable or disable the GPU if no argument is passed to `interface`. Setting `proc` to `TRUE` or `FALSE` enables or disables the GPU.

`signature(proc = "numeric", interface = "character")` This allows the user to choose the CPU or GPU interface. Should the user wish to enable a certain interface the string "CPU" or "GPU" are passed as the `interface` argument and the `proc` argument is set to `1` or `0`.

`signature(proc = "numeric", interface = "missing")` This allows the user to enable or disable the GPU if no argument is passed to `interface`. Setting `proc` to `1` or `0` enables or disables the GPU.

Examples

```
setGPU(TRUE,"CPU") # enables CPU interface where available
setGPU(TRUE,"GPU") # enables GPU interface only
setGPU(FALSE) #disables MAGMA library
```

---

solve *Solve a linear system Ax=b using GPU or multi-core architectures*

Description

Solve, using a GPU or multi-core CPU, a linear system Ax=b where A is one of dgeMatrix, dpoMatrix, dtrMatrix or dtpMatrix.

Usage

```
## S4 method for signature 'dpoMatrix'
solve(A, b, ...)
## S4 method for signature 'dgeMatrix'
solve(A, b, ...)
## S4 method for signature 'dtrMatrix'
solve(A, b, ...)
## S4 method for signature 'dtpMatrix'
solve(A, b, ...)
```

Arguments

A	Square dense matrix inheriting from dMatrix.
b	Matrix/vector inheriting from dMatrix or base::matrix.
...	potentially further arguments passed to methods.

Details

For further details on classes and methods see the full Matrix package documentation.

Methods

solve signature(A = "dgeMatrix", b = "missing"): Sets b as the identity matrix and calculates inverse of A; Uses PLASMA_dgetri or magma_dgetri for multi-core CPU and GPU respectively. Also uses some LAPACK dgetri to test singularity.

solve signature(A = "dgeMatrix", b = "ddenseMatrix"): Solves a linear system where b inherits from ddenseMatrix. If the routine is calling MAGMA "magma_dgetrs_gpu" is called. If PLASMA is chosen "PLASMA_dgetrs" is called.

solve signature(A = "dgeMatrix", b = "matrix"): Solves a linear system where b is of type matrix from the R base. If the routine is calling MAGMA "magma_dgetrs_gpu" is called. If PLASMA is chosen "PLASMA_dgetrs" is called.

solve `signature(A = "dgeMatrix", b = "sparseMatrix")`: Solves a linear system where b is a sparseMatrix from the Matrix package. For this routine the sparse matrix is coerced to a real dense matrix. If the routine is calling MAGMA `"magma_dgetrs_gpu"` is called. If PLASMA is chosen `"PLASMA_dgetrs"` is called.

solve `signature(A = "dpoMatrix", b = "missing")`: Sets b as the identity matrix and calculates inverse of `A`; Uses `"PLASMA_dgetri"` or `"magma_dpotri_gpu"` for multi-core CPU and GPU respectively.

solve `signature(A = "dpoMatrix", b = "dgeMatrix")`: Solves a linear system where b is of type `dgeMatrix`. If the routine is calling MAGMA `"magma_dpotrs_gpu"` is called. If PLASMA is chosen `"PLASMA_dpotrs"` is called.

solve `signature(A = "dpoMatrix", b = "matrix")`: Solves a linear system where b is of type matrix from the R `base`. If the routine is calling MAGMA `"magma_dpotrs_gpu"` is called. If PLASMA is chosen `"PLASMA_dpotrs"` is called.

solve `signature(A = "dtpMatrix", b = "missing")`: Sets b as the identity matrix and calculates inverse of `A`; This is only supported on the GPU so there is no PLASMA call here. Also the CUBLAS library is called here using `cublasDtpsv`.

solve `signature(A = "dtpMatrix", b = "ddenseMatrix")`: Solves a linear system where b inherits from `ddenseMatrix`. Again there is no MAGMA or PLASMA support for the `dtpMatrix` type but for GPU capable systems we call `cublasDtpsv`.

solve `signature(A = "dtrMatrix", b = "missing")`: Sets b as the identity matrix and calculates inverse of `A`; Uses `"PLASMA_dtrtri"` or `"magma_dtrtri"` for multi-core CPU and GPU respectively.

solve `signature(A = "dtrMatrix", b = "ddenseMatrix")`: Solves a linear system where b inherits from `ddenseMatrix`. If the routine is calling MAGMA `"magma_dtrsm"` is called. If PLASMA is chosen `"PLASMA_dtrsm"` is called.

solve `signature(A = "dtrMatrix", b = "matrix")`: Solves a linear system where b is of type matrix from the R `base`. If the routine is calling MAGMA `"magma_dtrsm"` is called. If PLASMA is chosen `"PLASMA_dtrsm"` is called.

solve `signature(A = "dtrMatrix", b = "Matrix")`: Solves a linear system where b inherits from `Matrix`. If the routine is calling MAGMA `"magma_dtrsm"` is called. If PLASMA is chosen `"PLASMA_dtrsm"` is called.

solve `signature(A = "dtrMatrix", b = "dMatrix")`: Solves a linear system where b inherits from `Matrix`. If the routine is calling MAGMA `"magma_dtrsm"` is called. If PLASMA is chosen `"PLASMA_dtrsm"` is called.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
p <- 128
A <- Matrix(rnorm(p*p), p, p) # random square matrix for large p
x_init <- vector("numeric", p)
b <- A
x <- solve(A, b)
```

```
stopifnot(identical(x, x_init))
```

`tcrossprod` *tcrossproduct using GPU or multi-core CPU*

Description

Given matrices `x` and `y` as arguments, return a matrix cross-product. This is formally equivalent to (but usually slightly faster than) the call `t(x) %*% y` (tcrossprod) or `x %*% t(y)` (tcrossprod).

Usage

```
tcrossprod(x, y)
## S4 method for signature 'dgeMatrix'
tcrossprod(x, y)
## S4 method for signature 'dtrMatrix'
tcrossprod(x, y)
```

Arguments

`x`	dense matrix or vector represented as a single row matrix.
`y`	dense matrix or vector represented as single row matrix.

Details

For further details on classes and methods see the full Matrix package documentation.

Methods

tcrossprod `(signature(x = "dgeMatrix", y = "missing")`: Calls CUBLAS function `cublasDsyrk` for GPU enabled systems and `PLASMA_dsyrk` for multi-core systems. Library settings also affect choice between GPU and CPU. If `y = NULL`, then it is taken to be the same matrix as `x`.

tcrossprod `(signature(x = "dgeMatrix", y = "dgeMatrix")`: Calls MAGMA function `magma_dgemm` for GPU enabled systems and `PLASMA_dgemm` for multi-core systems. Library settings also affect choice between GPU and CPU.

tcrossprod `(signature(x = "dgeMatrix", y = "Matrix")`: Calls MAGMA function `magma_dgemm` for GPU enabled systems and `PLASMA_dgemm` for multi-core systems. Library settings also affect choice between GPU and CPU.

tcrossprod `(signature(x = "dgeMatrix", y = "numeric")`: Calls MAGMA function `magma_dgemm` for GPU enabled systems and `PLASMA_dgemm` for multi-core systems. Library settings also affect choice between GPU and CPU. `y` is coerced to a `base::matrix`.

tcrossprod `(signature(x = "dgeMatrix", y = "matrix")`: Calls MAGMA function `magma_dgemm` for GPU enabled systems and `PLASMA_dgemm` for multi-core systems. Library settings also affect choice between GPU and CPU.

tcrossprod (signature(x = "dtrMatrix", y = "dtrMatrix"): Calls CUBLAS function cublasDtrmm for GPU enabled systems and PLASMA_dtrmm for multi-core systems. Library settings also affect choice between GPU and CPU.

tcrossprod (signature(x = "denseMatrix", y = "dtrMatrix"): y inherits from virtual class ddenseMatrix Calls MAGMA function magma_dgemm for GPU enabled systems and PLASMA_dgemm for multi-core systems. Library settings also affect choice between GPU and CPU.

tcrossprod (signature(x = "matrix", y = "dtrMatrix"): Calls CUBLAS function cublasDtrmm for GPU enabled systems and PLASMA_dtrmm for multi-core systems. Library settings also affect choice between GPU and CPU.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
m <- matrix(0, 400, 500)
set.seed(12)
m[runif(314, 0, length(m))] <- 1
mm <- as(m, "dgeMatrix")
object.size(m) / object.size(mm) # smaller by a factor of > 200

## tcrossprod() is very fast:
system.time(tCmm <- tcrossprod(mm))
system.time(cm <- crossprod(t(m)))
system.time(cm. <- tcrossprod(m))

stopifnot(cm == as(tCmm, "matrix"))
```

`%*%` *Matrix Multiplication of two matrices using GPU or multi-core architectures*

Description

Matrix multiplication using a GPU or multi-core CPU for most dense matrix types

Usage

```
## S4 method for signature 'dpoMatrix'
x %*% y
## S4 method for signature 'dgeMatrix'
x %*% y
## S4 method for signature 'dtrMatrix'
x %*% y
## S4 method for signature 'dtpMatrix'
x %*% y
```

Arguments

x	A dense matrix inheriting from Matrix or of type base::matrix
y	A dense matrix inheriting from dMatrix or of type base::matrix

Details

For further details on classes and methods see the full Matrix package documentation.

Methods

%*% signature(x = "ddenseMatrix", y = "ddenseMatrix"):x and y inherit from ddenseMatrix, x is coerced to dgeMatrix. For multi-core machines the PLASMA_dgemm is used or for GPU enabled machines magma_dgemm may be used also.

%*% signature(x = "dgeMatrix", y = "dgeMatrix"):x and y are of type dgeMatrix. For multi-core machines the PLASMA_dgemm is used or for GPU enabled machines magma_dgemm may be used also.

%*% signature(x = "dgeMatrix", y = "matrix"):For multi-core machines the PLASMA_dgemm is used or for GPU enabled machines magma_dgemm may be used also.

%*% signature(x = "dsyMatrix", y = "Matrix"): For multi-core machines the PLASMA_dsymm is used or for GPU enabled machines cublasDsymm may be used also.

%*% signature(x = "dsyMatrix", y = "ddenseMatrix"):For multi-core machines the PLASMA_dsymm is used or for GPU enabled machines cublasDsymm may be used also.

%*% signature(x = "dsyMatrix", y = "dsyMatrix"):For multi-core machines the PLASMA_dsymm is used or for GPU enabled machines cublasDsymm may be used also.

%*% signature(x = "ddenseMatrix", y = "dsyMatrix"):For multi-core machines the PLASMA_dsymm is used or for GPU enabled machines cublasDsymm may be used also.

%*% signature(x = "matrix", y = "dsyMatrix"):For multi-core machines the PLASMA_dsymm is used or for GPU enabled machines cublasDsymm may be used also.

%*% signature(x = "dspMatrix", y = "ddenseMatrix"):For GPU enabled machines cublasDspmv may be used. The PLASMA library does not support this type so there is no multi-core support for this function

%*% signature(x = "dspMatrix", y = "Matrix"):For GPU enabled machines cublasDspmv may be used also. The PLASMA library does not support this type so there is no multi-core support for this function.

%*% signature(x = "dtrMatrix", y = "dtrMatrix"):For multi-core machines the PLASMA_dtrmm is used or for GPU enabled machines cublasDtrmm may be used also.

%*% signature(x = "dtrMatrix", y = "ddenseMatrix"):For multi-core machines the PLASMA_dtrmm is used or for GPU enabled machines cublasDtrmm may be used also.

%*% signature(x = "dtrMatrix", y = "Matrix"):For multi-core machines the PLASMA_dtrmm is used or for GPU enabled machines cublasDtrmm may be used also.

%*% signature(x = "ddenseMatrix", y = "dtrMatrix"):For multi-core machines the PLASMA_dtrmm is used or for GPU enabled machines cublasDtrmm may be used also.

%*% signature(x = "matrix", y = "dtrMatrix"):For multi-core machines the PLASMA_dtrmm is used or for GPU enabled machines cublasDtrmm may be used also.

%*% `signature(x = "dtpMatrix", y = "ddenseMatrix")`:For GPU enabled machines `cublasDtpmv` may be used. The PLASMA library does not support this type so there is no multi-core support for this function

%*% `signature(x = "dgeMatrix", y = "dtpMatrix")`:For GPU enabled machines `cublasDtpmv` may be used also. The PLASMA library does not support this type so there is no multi-core support for this function.

References

Martin Maechler, Douglas Bates (Matrix package)

Examples

```
p <- 256
X <- Matrix(rnorm(p*p), p, p) # random square matrix for large p
Y <- Matrix(rnorm(p*p), p ,p)
Z <- X

#dtr triangular Matrix
X <- triu(X)
Y <- triu(y)
Z <- X
```

Index