Package ‘RCUDA’

November 20, 2016

Type Package

Title GPU Enabled BLAS, LAPACK, Statistical Functions and Random Number Generators

Version 1.0

Date 2016-11-07

Author Yuan Li, Hua Zhou

Maintainer Yuan Li <yli16@ncsu.edu>

Description Provides GPU-accelerated algebra and random number generating functions by wrapping CUDA library. It also includes some self-defined high level statistical functions based on NVIDIA CUDA framework.

License CPL

Depends R (>= 3.2.0)

NeedsCompilation yes

SystemRequirements Nvidia's CUDA toolkit (>= release 7.5); Linux operating system; GNU make.

URL `https://github.com/yuanli22/RCUDA`

RoxygenNote 5.0.1

R topics documented:

`addgpu` *addgpu*

Description

This function computes the element-wise addition of two given vectors/matrices by using CUDA cublas function cublasDgeam

Usage

```
addgpu(x, y)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension
`y`	list consisting of R external GPU pointer and dimension

Value

element-wise addition of two vectors/matrices (x + y), a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`subtractgpu`

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
addgpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

`amaxgpu` *amaxgpu*

Description

This function finds the (smallest) index of the element with the maximum magnitude of given vector/matrix by using CUDA cublas function cublasIdamax

Usage

```
amaxgpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

the resulting index

See Also

`amingpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
amaxgpu(a_gpu)
```

`amingpu` *amingpu*

Description

This function finds the (smallest) index of the element with the minimum magnitude of given vector by using CUDA cublas function cublasIdamin

Usage

```
amingpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

the resulting index

See Also

`amaxgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
amingpu(a_gpu)
```

`asumgpu` *asumgpu*

Description

This function computes the summation of the elements' absolute values of given vector/matrix by using CUDA cublas function cublasDasum

Usage

```
asumgpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

the vector/matrix's elements absolute values summation

See Also

`amaxgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
asumgpu(a_gpu)
```

axpygpu *axpygpu*

Description

This function multiplies the vector x by the scalar a and adds it to the vector y, and overwrites y as the result. by using CUDA cublas function cublasDaxpy. y = a x + y

Usage

```
axpygpu(x, y, alpha = 1)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension
`y`	list consisting of R external GPU pointer and dimension
`alpha`	scale factor alpha; default 1

Value

updated y vector/matrix

See Also

`scalgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
b_gpu <- creategpu(a)
axpygpu(a_gpu, b_gpu, 1)
```

betagpu *betagpu*

Description

This function computes the beta function of the given vector/matrix by using self-defined CUDA function

Usage

```
betagpu(x, y)
```

Arguments

x	list consisting of R external GPU pointer and dimension
y	list consisting of R external GPU pointer and dimension

Value

beta function result of given vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

gammagpu

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
betagpu(a_gpu, a_gpu) -> b_gpu
gathergpu(b_gpu)
```

copygpu *copygpu*

Description

This function copies the vector x into the vector y by using CUDA cublas function cublasDcopy

Usage

```
copygpu(x, y)
```

Arguments

x	list consisting of R external GPU pointer and dimension
y	list consisting of R external GPU pointer and dimension

Value

copied vector/matrix

See Also

axpygpu

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
copygpu(a_gpu, b_gpu)
```

creategpu *creategpu*

Description

Create a GPU vector/matrix by copying from the input R vector

Usage

```
creategpu(input, nrow = NULL, ncol = NULL)
```

Arguments

`input`	R vector to be copied
`nrow`	the desired number of rows
`ncol`	the desired number of columns

Details

This function creates a vector/matrix in GPU by calling the CUDA cudamalloc function, and then copys from input R vector. The output of this function is a list consisting of the GPU pointer and its dimension.

If either one of nrow or ncol is not given, an one column matrix/vector is returned. This function returns row-major matrix.

Value

a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

Note

output is a R external GPU pointer and can only be used in Rcublas functions

Author(s)

Yuan Li

See Also

gathergpu

Examples

```
a <- rnorm(6)
a_gpu <- creategpu(a, 2, 3)
gathergpu(a_gpu)
```

---

dbetagpu *dbetagpu*

Description

This function computes the beta pdf function of given vector/matrix by using self-defined CUDA function

Usage

```
dbetagpu(input, k = 1, theta = 1)
```

Arguments

input	list consisting of R external GPU pointer and dimension
k	shape parameter of Beta distribution; default value 1
theta	scale parameter of Beta distribution; default value 1

Value

beta pdf result of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

dbetagpu

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
dbetagpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

`dgammagpu` *dgammagpu*

Description

This function computes the gammma pdf function of given vector/matrix by using self-defined CUDA function

Usage

```
dgammagpu(input, k = 1, theta = 1)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension
`k`	shape parameter of Gamma distribution; default value 1
`theta`	scale parameter of Gamma distribution; default value 1

Value

gamma pdf result of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`dbetagpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
dgammagpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

`dgmmgpu` *dgmmgpu*

Description

This function performs the matrix-matrix multiplication C = A diag(x) or C = diag(x) A by using CUDA cublas function cublasDdgmm

Usage

```
dgmmgpu(sidemode = 1, A, x, C)
```

Arguments

`sidemode`	indicates whether the given matrix is on the left or right side in the matrix equation solved by a particular function. If sidemode == 1, the matrix is on the left side in the equation If sidemode == 2, the matrix is on the right side in the equation.
`A`	input matrix; list of R external GPU pointer and dimension
`x`	input vector; list of R external GPU pointer and dimension
`C`	input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`symmgpu`

`dividegpu` *dividegpu*

Description

This function computes the element-wise division of two given vectors/matrices by using self-defined CUDA function

Usage

```
dividegpu(x, y)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension
`y`	list consisting of R external GPU pointer and dimension

Value

element-wise division of vectors/matrices (x / y), a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`multiplygpu`

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
dividegpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

`dnormgpu` *dnormgpu*

Description

This function computes the normal distribution density of given vector/matrix

Usage

```
dnormgpu(input, mean = 0, sd = 1)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension
`mean`	vector/matrix of mean
`sd`	vector/matrix of standard deviation

Details

If mean or sd are not specified they assume the default values of 0 and 1, respectively.

Value

normal distribution density vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`pnormgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
dnormgpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

`dotgpu` *dotgpu*

Description

This function computes the dot product of two given vectors/matrix by using CUDA cublas function cublasDdot

Usage

```
dotgpu(x, y)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension
`y`	list consisting of R external GPU pointer and dimension

Value

the resulting dot product

See Also

`nrm2gpu`

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
dotgpu(a_gpu, b_gpu)
```

expgpu *expgpu*

Description

This function computes the exponential of given vector/matrix by using self-defined CUDA function

Usage

```
expgpu(input)
```

Arguments

input	list consisting of R external GPU pointer and dimension

Value

exponential of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

loggpu

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
expgpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

gammagpu *gammagpu*

Description

This function computes the gammma function of given vector/matrix by using self-defined CUDA function

Usage

```
gammagpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

gamma result of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`betagpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
gammagpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

`gathergpu` *gathergpu*

Description

Copy GPU matrix/vector to R vector

Usage

```
gathergpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and its dimension

Details

This function copys GPU vector/matrix to R vector

The output is always R vector, and GPU matrix will be copied by row-major. For example, an m by n GPU matrix will be converted to a m*n R vector.

Value

R vector

Note

output is R vector and can be used by any R functions

Author(s)

Yuan Li

See Also

gathergpu creategpu

Examples

```
a <- 1:6
am_gpu <- creategpu(a, 3, 2)
gathergpu(am_gpu)
```

---

gbmvgpu *gbmvgpu*

Description

This function computes banded matrix-vector multipication y = a A x + b y by using CUDA cublas function cublasDgbmv

Usage

```
gbmvgpu(trans = 1, kl, ku, alpha = 1, A, x, beta = 0, y)
```

Arguments

trans	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
kl	number of subdiagonals
ku	number of superdiagonals
alpha	scale factor a of banded matrix A; default 1
A	input matrix; list of R external GPU pointer and dimension
x	input vector; list of R external GPU pointer and dimension
beta	scale factor b of vector y; default 0
y	input/output vector; list of R external GPU pointer and dimension

Value

vector y, a list consisting of

- ptr: GPU pointer
- m: length of vector y
- n: 1

See Also

`gergpu`

`geamgpu` *geamgpu*

Description

This function computes the matrix-matrix addition/trasportation C = a op (A) + b op (B) by using CUDA cublas function cublasDgeam

Usage

```
geamgpu(transa = 1, transb = 1, alpha = 1, A, B, beta = 0, C)
```

Arguments

`transa`	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`transb`	matrix B transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`alpha`	scale factor a of matrix A; default 1
`A`	input matrix; list of R external GPU pointer and dimension
`B`	input matrix; list of R external GPU pointer and dimension
`beta`	scale factor b of matrix B; default 0
`C`	output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`gemvgpu`

gemmgpu *gemmgpu*

Description

This function computes the matrix-matrix multiplication C = a op (A) op (B) + b C by using CUDA cublas function cublasDgemm

Usage

```
gemmgpu(transa = 1, transb = 1, alpha = 1, A, B, beta = 0, C)
```

Arguments

`transa` matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`transb` matrix B transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`alpha` scale factor a of matrix A; default 1

`A` input matrix; list of R external GPU pointer and dimension

`B` input matrix; list of R external GPU pointer and dimension

`beta` scale factor b of matrix C; default 0

`C` input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`gemvgpu`

Examples

```
A_gpu <- creategpu(1:6, 3, 2)
B_gpu <- creategpu(1:6, 3, 2)
C_gpu <- creategpu(1:4, 2, 2)
gemmgpu(2, 1, 1, A_gpu, B_gpu, beta=1, C_gpu)
gathergpu(C_gpu)
```

gemvgpu *gemvgpu*

Description

This function computes matrix-vector multipication y = a A x + b y by using CUDA cublas function cublasDgemv

Usage

```
gemvgpu(trans = 1, alpha = 1, A, x, beta = 0, y)
```

Arguments

`trans` matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`alpha` scale factor a of matrix A; default 1

`A` input matrix; list of R external GPU pointer and dimension

`x` input vector; list of R external GPU pointer and dimension

`beta` scale factor b of vector y; default 0

`y` input/output vector; list of R external GPU pointer and dimension

Value

vector y, a list consisting of

- ptr: GPU pointer
- m: length of vector y
- n: 1

See Also

`gergpu`

Examples

```
A <- 1:12
x <- 1:3
y <- 1:4
A_gpu <- creategpu(A, 4, 3)
x_gpu <- creategpu(x)
y_gpu <- creategpu(y)
gemvgpu(trans = 1, alpha = 1, A_gpu, x_gpu, beta = 1, y_gpu)
gathergpu(y_gpu)
```

gergpu *gergpu*

Description

This function perform the the rank-1 update A = a x y T + A, by using CUDA cublas function cublasDger

Usage

```
gergpu(alpha = 1, x, y, A)
```

Arguments

`alpha`	scale factor a of matrix A; default 1
`x`	input vector; list of R external GPU pointer and dimension
`y`	input vector; list of R external GPU pointer and dimension
`A`	input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix A, a list consisting of

- ptr: GPU pointer
- m: matrix A's number of rows
- n: matrix A's number of columns

See Also

`gemvgpu`

Examples

```
A <- 1:12
x <- 1:3
y <- 1:4
A_gpu <- creategpu(A, 3, 4)
x_gpu <- creategpu(x)
y_gpu <- creategpu(y)
gergpu(1,x_gpu, y_gpu, A_gpu)
gathergpu(A_gpu)
```

`GPUobject` *GPUobject*

Description

classify the input as GPU vector/matrix and assign its dimension

Usage

```
GPUobject(input, nrow, ncol)
```

Arguments

`input`	R external pointer
`nrow`	number of rows
`ncol`	number of columns

Details

This function classifies the input object as GPU vector/matrix and assign its dimension The output of this function is a list consisting of the GPU pointer and its dimension

Value

a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

Note

output is a R external GPU pointer and can only be used in Rcublas functions

Author(s)

Yuan Li

See Also

`gathergpu`

`gpuquery` *gpuquery This function returns the information of available GPU device in system*

Description

gpuquery

This function returns the information of available GPU device in system

Usage

```
gpuquery()
```

See Also

`creategpu`

Examples

```
gpuquery()
```

`inversegpu` *inversegpu*

Description

This function computes the inversion of given matrix (squared) by using CUDA cublas function cublasDgetrfBatched and cublasDgetriBatched (LU decomposition)

Usage

```
inversegpu(X)
```

Arguments

`X`	input matrix; list of R external GPU pointer and dimension

Value

matrix inversion, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`mmgpu` `creategpu`

Examples

```
a <- 1:9
a_gpu <- creategpu(a, 3, 3)
inversegpu(a_gpu) -> c_gpu
gathergpu(c_gpu)
```

`loggpu` *loggpu*

Description

This function computes the natural logarithms of given vector/matrix by using self-defined CUDA function

Usage

```
loggpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

natural logarithms of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`expgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
loggpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

meangpu *meangpu*

Description

Compute the mean of given vector/matrix

Usage

```
meangpu(x)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension

Details

This function computes the mean of given vector/matrix by using self-defined CUDA function

Value

vector/matrix mean

Author(s)

Yuan Li

See Also

`sumgpu`

Examples

```
a <- creategpu(1:4)
meangpu(a)
```

mmgpu *mmgpu*

Description

This function computes the matrix-matrix multiplication (X * Y) by using CUDA cublas function cublasDgemm

Usage

```
mmgpu(X, Y)
```

Arguments

X	input matrix; list of R external GPU pointer and dimension
Y	input matrix; list of R external GPU pointer and dimension

Value

matrix-matrix multiplication (X * Y), a list consisting of

- ptr: GPU pointer
- m: matrix X's number of rows
- n: matrix Y's number of columns

See Also

mmgpu

Examples

```
a <- 1:6
b <- 2:7
a_gpu <- creategpu(a, 2, 3)
b_gpu <- creategpu(b, 3, 2)
mmgpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

multiplygpu *multiplygpu*

Description

This function computes the element-wise multiplication of two given vectors/matricesby using CUDA cublas function cublasDdgmm

Usage

```
multiplygpu(x, y)
```

Arguments

x	list consisting of R external GPU pointer and dimension
y	list consisting of R external GPU pointer and dimension

Value

element-wise multiplication of vectors/matrices (x * y), a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

dividegpu

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
multiplygpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

mvgpu *mvgpu*

Description

This function computes the matrix-vector multiplication (X * y) by using CUDA cublas function cublasDgemv

Usage

```
mvgpu(X, y)
```

Arguments

`X`	input matrix; list of R external GPU pointer and dimension
`y`	input vector; list of R external GPU pointer and dimension

Value

matrix-vector multiplication (X * y), a list consisting of

- ptr: GPU pointer
- m: matrix X's number of rows
- n: matrix X's number of columns; vector y's number of elements

See Also

`mmgpu`

Examples

```
a <- 1:4
b <- 2:3
a_gpu <- creategpu(a, 2, 2)
b_gpu <- creategpu(b)
mvgpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

nrm2gpu *nrm2gpu*

Description

This function computes Euclidean norm of given vector/matrix by using CUDA cublas function cublasDnrm2

Usage

```
nrm2gpu(input)
```

Arguments

input	list consisting of R external GPU pointer and dimension

Value

vector Euclidean norm, a non-negative number

Author(s)

Yuan Li

See Also

dotgpu

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
nrm2gpu(a_gpu)
```

pnormgpu *pnormgpu*

Description

This function computes the standard normal distribution cumulative density (CDF) of given vector/matrix

Usage

```
pnormgpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

standard normal CDF, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`dnormgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
pnormgpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

`powergpu` *powergpu*

Description

This function computes the power of given vector/matrix by using self-defined CUDA function

Usage

```
powergpu(input, alpha = 1)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension
`alpha`	power factor

Value

powered vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

```
sqrtgpu
```

Examples

```
a <- 1:4
b <- 2
a_gpu <- creategpu(a)
powergpu(a_gpu, b) -> b_gpu
gathergpu(b_gpu)
```

rbetagpu *rbetagpu*

Description

This function generates Beta distributed random numbers by using self-defined CUDA function based on George Marsaglia and Wai Wan Tsang's method and gamma/beta relationship

Usage

```
rbetagpu(n, alpha = 1, beta = 1, seed = 1)
```

Arguments

`n`	number of random numbers
`alpha`	shape parameter of Beta distribution; default value 1
`beta`	shape parameter of Beta distribution; default value 1
`seed`	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

```
runifgpu
```

Examples

```
a_gpu <- rbetagpu(100, 2, 1)
```

rdirichletgpu *rdirichletgpu*

Description

This function generates Dirichlet distributed random numbers by using self-defined CUDA function based on George Marsaglia and Wai Wan Tsang's method and gamma/Dirichlet relationship

Usage

```
rdirichletgpu(n, alpha, seed = 1)
```

Arguments

n	number of random numbers
alpha	concentration parameters of Dirichlet distribution;
seed	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

runifgpu

Examples

```
a_gpu <- rdirichletgpu(100, 2, 1)
```

rgammagpu *rgammagpu*

Description

This function generates Gamma distributed random numbers by using self-defined CUDA function based on George Marsaglia and Wai Wan Tsang's method

Usage

```
rgammagpu(n, k = 1, theta = 1, seed = 1)
```

Arguments

`n`	number of random numbers
`k`	shape parameter of Gamma distribution; default value 1
`theta`	scale parameter of Gamma distribution; default value 1
`seed`	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`runifgpu`

Examples

```
a_gpu <- rgammagpu(100, 2, 1)
```

`rlognormgpu` *rlognormgpu*

Description

This function generates log-normally distributed random numbers by using CUDA curand function CURAND_RNG_PSEUDO_DEFAULT and curandGenerateLogNormalDouble

Usage

```
rlognormgpu(n, mean = 0, sd = 1, seed = 1)
```

Arguments

`n`	number of random numbers
`mean`	mean of log-normal distribution; default value 0
`sd`	standard deviation of log-normal distribution; default value 1
`seed`	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`rnormgpu`

Examples

```
a_gpu <- rlognormgpu(100, 0, 1, 15)
gathergpu(a_gpu)
```

`rnormgpu` *rnormgpu*

Description

This function generates normally distributed random numbers by using CUDA curand function CURAND_RNG_PSEUDO_DEFAULT and curandGenerateNormalDouble

Usage

```
rnormgpu(n, mean = 0, sd = 1, seed = 1)
```

Arguments

`n`	number of random numbers
`mean`	mean of normal distribution; default value 0
`sd`	standard deviation of normal distribution; default value 1
`seed`	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`rlognormgpu`

Examples

```
a_gpu <- rnormgpu(100, 0, 1, 15)
gathergpu(a_gpu)
```

rpoisgpu *rpoisgpu*

Description

This function generates Poisson distributed random numbers by using CUDA curand function CURAND_RNG_PSEUDO_DEFAULT and curandGeneratePoisson

Usage

```
rpoisgpu(n, lambda = 1, seed = 1)
```

Arguments

`n`	number of random numbers
`lambda`	mean of Poisson distribution; default value 1
`seed`	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`runifgpu`

Examples

```
a_gpu <- rpoisgpu(100, 1)
```

runifgpu *runifgpu*

Description

This function generates uniformly distributed random numbers between 0 and 1 by using CUDA curand function CURAND_RNG_PSEUDO_DEFAULT and curandGenerateUniformDouble

Usage

```
runifgpu(n, seed = 1)
```

Arguments

n	number of random numbers
seed	random number generator seed; default value 1

Value

generated random numbers vector, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`creategpu`

Examples

```
a_gpu <- runifgpu(100, 15)
gathergpu(a_gpu)
```

sbmvgpu *sbmvgpu*

Description

This function computes symmetric banded matrix-vector multipication y = a A x + b y by using CUDA cublas function cublasDsbmv

Usage

```
sbmvgpu(fillmode = 1, k, alpha = 1, A, x, beta = 0, y)
```

Arguments

`fillmode` indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the symmetric banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the symmetric banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.

`k` number of subdiagonals

`alpha` scale factor a of symmetric banded matrix A; default 1

`A` input matrix; list of R external GPU pointer and dimension

`x` input vector; list of R external GPU pointer and dimension

`beta` scale factor b of vector y; default 0

`y` input/output vector; list of R external GPU pointer and dimension

Value

vector y, a list consisting of

- ptr: GPU pointer
- m: length of vector y
- n: 1

See Also

`gemvgpu`

`scalegpu` *scalegpu*

Description

This function scales the given vector/matrix by a scalar by using CUDA cublas function cublasDcopy

Usage

```
scalegpu(input, alpha)
```

Arguments

`input` list consisting of R external GPU pointer and dimension

`alpha` scale factor

Value

scaled vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`expgpu`

Examples

```
a <- 1:4
b <- 2
a_gpu <- creategpu(a)
scalegpu(a_gpu, b) -> b_gpu
gathergpu(b_gpu)
```

`scalgpu` *scalgpu*

Description

This function scales the vector x by the scalar a and overwrites it with the result by using CUDA cublas function cublasDscal

Usage

```
scalgpu(x, alpha = 1)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension
`alpha`	scale factor alpha, default 1

Value

scaled vector/matrix

See Also

`scalegpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
scalgpu(a_gpu, 2)
```

sqrtgpu *sqrtgpu*

Description

This function computes the square root of given vector/matrix by using self-defined CUDA function

Usage

```
sqrtgpu(input)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension

Value

square root of vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`expgpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
sqrtgpu(a_gpu) -> b_gpu
gathergpu(b_gpu)
```

subsetgpu *subsetgpu*

Description

This function returns the specified subset of given GPU vector/matrix by using self-defined CUDA function

Usage

```
subsetgpu(input, index)
```

Arguments

`input` list consisting of R external GPU pointer and dimension

`index` index of the vector/matrix subset

Value

subset of the given vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`creategpu`

Examples

```
a <- 1:4
a_gpu <- creategpu(a)
subsetgpu(a_gpu,c(1, 2))->b_gpu
gathergpu(b_gpu)
```

`subtractgpu` *subtractgpu*

Description

This function computes the element-wise subtraction of two given vectors/matrices by using CUDA cublas function cublasDgeam

Usage

```
subtractgpu(x, y)
```

Arguments

`x` list consisting of R external GPU pointer and dimension

`y` list consisting of R external GPU pointer and dimension

Value

element-wise subtraction of vectors or matrices (x - y), a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`addgpu`

Examples

```
a <- 1:4
b <- 2:5
a_gpu <- creategpu(a)
b_gpu <- creategpu(b)
subtractgpu(a_gpu, b_gpu) -> c_gpu
gathergpu(c_gpu)
```

`sumgpu` *sumgpu*

Description

Compute the summation of given vector/matrix

Usage

```
sumgpu(x)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension

Details

This function computes the summation of given vector/matrix by using self-defined CUDA function

Value

vector/matrix summation

Author(s)

Yuan Li

See Also

`meangpu`

Examples

```
a <- creategpu(1:4)
sumgpu(a)
```

`symmgpu` *symmgpu*

Description

This function computes the symmetric matrix-matrix multiplication C = a A B + b C by using CUDA cublas function cublasDsymm

Usage

```
symmgpu(sidemode = 1, fillmode = 1, alpha = 1, A, B, beta = 0, C)
```

Arguments

`sidemode` indicates whether the given matrix is on the left or right side in the matrix equation solved by a particular function. If sidemode == 1, the matrix is on the left side in the equation If sidemode == 2, the matrix is on the right side in the equation.

`fillmode` indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.

`alpha` scale factor a of matrix AB; default 1

`A` input matrix; list of R external GPU pointer and dimension

`B` input matrix; list of R external GPU pointer and dimension

`beta` scale factor b of matrix C; default 0

`C` input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`gemmgpu`

symvgpu *symvgpu*

Description

This function computes symmetric matrix-vector multipication y = a A x + b y by using CUDA cublas function cublasDsymv

Usage

```
symvgpu(fillmode = 1, alpha = 1, A, x, beta = 0, y)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the symmetric banded matrix A is stored in lower mode if fillmode == 2 then the symmetric banded matrix A is stored in upper mode
`alpha`	scale factor a of symmetric banded matrix A; default 1
`A`	input matrix; list of R external GPU pointer and dimension
`x`	input vector; list of R external GPU pointer and dimension
`beta`	scale factor b of vector y; default 0
`y`	input/output vector; list of R external GPU pointer and dimension

Value

vector y, a list consisting of

- ptr: GPU pointer
- m: length of vector y
- n: 1

See Also

`sbmvgpu`

syr2gpu *syr2gpu*

Description

This function performs rank 2 update, A = a (x y T + y x T) + A, where A is symmetric matrix, x is vector, a is scalar by using CUDA cublas function cublasDsyr2

Usage

```
syr2gpu(fillmode = 1, alpha = 1, x, y, A)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the symmetric banded matrix A is stored in lower mode if fillmode == 2 then the symmetric banded matrix A is stored in upper mode
`alpha`	scale factor a of symmetric banded matrix A; default 1
`x`	input vector; list of R external GPU pointer and dimension
`y`	input vector; list of R external GPU pointer and dimension
`A`	input matrix; list of R external GPU pointer and dimension

Value

updated matrix A

See Also

`syrgpu`

syr2kgpu *syr2kgpu*

Description

This function performs the symmetric rank- 2k update C = a(op (A)op (B) T + op (B)op (A)T) + b C by using CUDA cublas function cublasDsyr2k

Usage

```
syr2kgpu(fillmode = 1, trans = 1, alpha = 1, A, B, beta = 0, C)
```

Arguments

`fillmode` indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.

`trans` matrix A and B transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`alpha` scale factor a ; default 1

`A` input matrix; list of R external GPU pointer and dimension

`B` input matrix; list of R external GPU pointer and dimension

`beta` scale factor b; default 0

`C` input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`syrkgpu`

`syrgpu` *syrgpu*

Description

This function performs rank 1 update, A = a x x T + A, where A is symmetric matrix, x is vector, a is scalar by using CUDA cublas function cublasDsyr

Usage

```
syrgpu(fillmode = 1, alpha = 1, x, A)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the symmetric banded matrix A is stored in lower mode if fillmode == 2 then the symmetric banded matrix A is stored in upper mode
`alpha`	scale factor a of symmetric banded matrix A; default 1
`x`	input vector; list of R external GPU pointer and dimension
`A`	input matrix; list of R external GPU pointer and dimension

Value

updated matrix A

See Also

`gergpu`

`syrkgpu` *syrkgpu*

Description

This function performs the symmetric rank- k update C = a op (A) op (A) T + b C by using CUDA cublas function cublasDsyrk

Usage

```
syrkgpu(fillmode = 1, trans = 1, alpha = 1, A, beta = 0, C)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.
`trans`	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`alpha`	scale factor a; default 1
`A`	input matrix; list of R external GPU pointer and dimension
`beta`	scale factor b; default 0
`C`	input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`gemmgpu`

`tbmvgpu` *tbmvgpu*

Description

This function computes triangular banded matrix-vector multipication x = op(A) x by using CUDA cublas function cublasDtbmv

Usage

```
tbmvgpu(fillmode = 1, trans = 1, diagmode = 1, k, A, x)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.
`trans`	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`diagmode`	indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements
`k`	number of sub- or super- diagonals
`A`	input matrix; list of R external GPU pointer and dimension
`x`	input/output vector; list of R external GPU pointer and dimension

Value

updated vector x, a list consisting of

- ptr: GPU pointer
- m: length of vector x
- n: 1

See Also

`gemvgpu`

`tbsvgpu` *tbsvgpu*

Description

This function solves the triangular banded linear system op(A) x = b by using CUDA cublas function cublasDtbsv

Usage

```
tbsvgpu(fillmode = 1, trans = 1, diagmode = 1, k, A, x)
```

Arguments

`fillmode` indicates if matrix A lower or upper part is stored, the other symmetric part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.

`trans` matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`diagmode` indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements

`k` number of sub- or super- diagonals

`A` input matrix; list of R external GPU pointer and dimension

`x` input/output vector; list of R external GPU pointer and dimension

Value

updated vector x, a list consisting of

- ptr: GPU pointer
- m: length of vector x
- n: 1

See Also

`tbmvgpu`

`tgpu` *tgpu*

Description

This function transposes the given matrix by using CUDA cublas cublasDgeam

Usage

```
tgpu(X)
```

Arguments

`X`	input matrix; list of R external GPU pointer and dimension

Value

matrix transpose, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`creategpu`

Examples

```
a <- 1:12
a_gpu <- creategpu(a, 3, 4)
tgpu(a_gpu) -> c_gpu
gathergpu(c_gpu)
```

trmmgpu *trmmgpu*

Description

This function computes the triangle matrix-matrix multiplication C = a A B or C = a B A by using CUDA cublas function cublasDtrmm

Usage

```
trmmgpu(sidemode = 1, fillmode = 1, trans = 1, diagmode = 1,
  alpha = 1, A, B, C)
```

Arguments

sidemode	indicates whether the given matrix is on the left or right side in the matrix equation solved by a particular function. If sidemode == 1, the matrix is on the left side in the equation If sidemode == 2, the matrix is on the right side in the equation.
fillmode	indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.
trans	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
diagmode	indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements.
alpha	scale factor a of matrix AB; default 1
A	input matrix; list of R external GPU pointer and dimension
B	input matrix; list of R external GPU pointer and dimension
C	input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix C, a list consisting of

- ptr: GPU pointer
- m: matrix C's number of rows
- n: matrix C's number of columns

See Also

`symmgpu`

`trmvgpu` *trmvgpu*

Description

This function computes triangular matrix-vector multipication x = op(A) x by using CUDA cublas function cublasDtrmv

Usage

```
trmvgpu(fillmode = 1, trans = 1, diagmode = 1, A, x)
```

Arguments

`fillmode`	indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.
`trans`	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`diagmode`	indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements
`A`	input matrix; list of R external GPU pointer and dimension
`x`	input/output vector; list of R external GPU pointer and dimension

Value

updated vector x, a list consisting of

- ptr: GPU pointer
- m: length of vector x
- n: 1

See Also

`gemvgpu`

trsmgpu *trsmgpu*

Description

This function solves the triangle linear system A X = a B or X A = a B by using CUDA cublas function cublasDtrsm

Usage

```
trsmgpu(sidemode = 1, fillmode = 1, trans = 1, diagmode = 1,
  alpha = 1, A, B)
```

Arguments

`sidemode`	indicates whether the given matrix is on the left or right side in the matrix equation solved by a particular function. If sidemode == 1, the matrix is on the left side in the equation If sidemode == 2, the matrix is on the right side in the equation.
`fillmode`	indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.
`trans`	matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)
`diagmode`	indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements.
`alpha`	scale factor a; default 1
`A`	input matrix; list of R external GPU pointer and dimension
`B`	input/output matrix; list of R external GPU pointer and dimension

Value

updated matrix B, a list consisting of

- ptr: GPU pointer
- m: matrix B's number of rows
- n: matrix B's number of columns

See Also

`trmmgpu`

`trsvgpu` *trsvgpu*

Description

This function solves triangular linear system op(A) x = b by using CUDA cublas function cublasDtrsv

Usage

```
trsvgpu(fillmode = 1, trans = 1, diagmode = 1, A, x)
```

Arguments

`fillmode` indicates if matrix A lower or upper part is stored, the other part is not referenced and is inferred from the stored elements. if fillmode == 1 then the triagular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row 1, the first subdiagonal in row 2 (starting at first position), the second subdiagonal in row 3 (starting at first position), etc. if fillmode == 2 then the triangular banded matrix A is stored column by column, with the main diagonal of the matrix stored in row k+1, the first superdiagonal in row k (starting at second position), the second superdiagonal in row k-1 (starting at third position), etc.

`trans` matrix A transpose operator, 1 (non-transpose), 2 (transpose), 3 (conjugate transpose); default at 1 (non-transpose)

`diagmode` indicates whether the main diagonal of the matrix A is unity and consequently should not be touched or modified by the function. if diagmode = 1, the matrix diagonal has non-unit elements, if diagmode = 2, the matrix diagonal has unit elements

`A` input matrix; list of R external GPU pointer and dimension

`x` input/output vector; list of R external GPU pointer and dimension

Value

updated vector x, a list consisting of

- ptr: GPU pointer
- m: length of vector x
- n: 1

See Also

`tbsvgpu`

`vargpu` *vargpu*

Description

Compute the variance of given vector/matrix

Usage

```
vargpu(x)
```

Arguments

`x`	list consisting of R external GPU pointer and dimension

Details

This function computes the variance of given vector/matrix by using self-defined CUDA function

Value

vector/matrix variance

Author(s)

Yuan Li

See Also

`sumgpu`

Examples

```
a <- creategpu(1:4)
vargpu(a)
```

vectincregpu *vectincregpu*

Description

This function computes the constant increment of given vector/matrix by using self-defined CUDA function

Usage

```
vectincregpu(input, alpha = 1)
```

Arguments

`input`	list consisting of R external GPU pointer and dimension
`alpha`	increment factor

Value

powered vector/matrix, a list consisting of

- ptr: GPU pointer
- m: number of rows
- n: number of columns

See Also

`sqrtgpu`

Examples

```
a <- 1:4
b <- 2
a_gpu <- creategpu(a)
powergpu(a_gpu, b) -> b_gpu
gathergpu(b_gpu)
```