Package 'rkeops'

September 4, 2020

Type Package

Title Kernel Operations on the GPU, with Autodiff, without Memory Overflows

Version 1.4.1.1

Date 2020-09-04

Description The 'KeOps' library lets you compute generic reductions of very large arrays whose entries are given by a mathematical formula. It combines a tiled reduction scheme with an automatic differentiation engine, and can be used through 'R', 'Matlab', 'NumPy' or 'PyTorch' backends. It is perfectly suited to the computation of Kernel dot products and the associated gradients, even when the full kernel matrix does not fit into the GPU memory.

License MIT + file LICENSE

Depends R (>= 3.1.0)

LinkingTo Rcpp (>= 1.0.1), RcppEigen (>= 0.3.3.5)

Imports Rcpp (>= 1.0.1), openssl (>= 1.3), stringr (>= 1.4.0)

Suggests testthat (>= 2.1.0), knitr, rmarkdown

OS_type unix

SystemRequirements C++11, cmake (>= 3.10), clang (optional), CUDA (optional but recommended)

URL https://www.kernel-operations.io/, https://github.com/getkeops/keops/

BugReports https://github.com/getkeops/keops/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation yes

Author Benjamin Charlier [aut] (<http://imag.umontpellier.fr/~charlier/>),
Jean Feydy [aut] (<https://www.math.ens.fr/~feydy/>),
Joan A. Glaunès [aut] (<https://www.mi.parisdescartes.fr/~glaunes/>),
Ghislain Durif [aut, cre] (<https://gdurif.perso.math.cnrs.fr/>),
François-David Collin [ctb] (Development-related consulting and support),
Daniel Frey [ctb] (Author of the included C++ library 'sequences')

Maintainer Ghislain Durif <gd.dev@libertymail.net>

Repository CRAN

Date/Publication 2020-09-04 12:10:08 UTC

R topics documented:

`rkeops-package` *rkeops RKeOps: kernel operations on GPU, with autodiff, without memory overflows in R*

Description

RKeOps is the R package interfacing the cpp/cuda library KeOps. It provides standard R functions that can be used in any R (>=3) codes.

Details

The KeOps library provides seamless kernel operations on GPU, with auto-differentiation and without memory overflows.

With RKeOps, you can compute generic reductions of very large arrays whose entries are given by a mathematical formula. It combines a tiled reduction scheme with an automatic differentiation engine. It is perfectly suited to the computation of Kernel dot products and the associated gradients, even when the full kernel matrix does not fit into the GPU memory.

For more information, please read the vignettes (`browseVignettes("rkeops")`) and visit https://www.kernel-operations.io/.

Author(s)

- Benjamin Charlier
- Ghislain Durif
- Jean Feydy
- Joan Alexis Glaunès
- François-David Collin

`clean_rkeops` *Clean build directory*

Description

Remove all dll files generated from compilations of user-defined operators.

Usage

```
clean_rkeops()
```

Details

When compiling a user-defined operators, a shared object (so) library (or dynamic link library, dll) file is created in the directory `build_dir` specified in compile options of `rkeops`. For every new operators, such a file is created.

Calling `clean_rkeops()` allows you to empty the directory `build_dir`.

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::compile_options(), rkeops::set_rkeops_option()

Examples

```
library(rkeops)
clean_rkeops()
```

---

compile4cpu *Disable compilation of GPU-compatible user-defined operators*

Description

Set up rkeops compile options to compile user-defined operators that run be computed on CPU.

Usage

```
compile4cpu()
```

Details

Note: Default behavior is to compile GPU-compatible operators thus, if you do not modify rkeops options, you have to call the function compile4cpu to disable GPU-support.

CPU-compatible operators cannot run on GPU.

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::compile4gpu(), rkeops::use_cpu()

Examples

```
library(rkeops)
compile4cpu()
```

`compile4float32` *Enable compiling of user-defined operators using float 32bits precision.*

Description

Set up `rkeops` compile options to compile user-defined operators that use float 32bits precision in computation.

Usage

```
compile4float32()
```

Details

Note: Default behavior is to compile operators operators that use float 32bits precision in computation. Hence, if you do not modify `rkeops` options, you do not have to call the function `compile4float32` to compile operators using float 32bits precision.

Since R only manages float 64bits or double numbers, the input and output are casted to float 32bits before and after computations respectively.

Value

None

Author(s)

Ghislain Durif

See Also

`rkeops::compile4float64()`

Examples

```
library(rkeops)
compile4float32()
```

`compile4float64` *Enable compiling of user-defined operators using float 64bits precision.*

Description

Set up `rkeops` compile options to compile user-defined operators that use float 64bits precision in computation.

Usage

```
compile4float64()
```

Details

Note: Default behavior is to compile operators operators that use float 32bits precision in computation. Hence, if you do not modify `rkeops` options, you have to call the function `compile4float64` to compile operators using float 64bits precision.

Using float 64bits (or double) precision is likely to result in a loss of performance regarding computing time on GPU. If you want to get the best performance but worry about computation precision, you can use float 32bits precision and compensated sums that are implemented in KeOps.

Value

None

Author(s)

Ghislain Durif

See Also

`rkeops::compile4float32()`

Examples

```
library(rkeops)
compile4float64()
```

compile4gpu *Enable compilation of GPU-compatible user-defined operators if possible*

Description

Set up rkeops compile options to compile user-defined operators that run on GPU. If CUDA is not available, user-defined operators will still be compiled without GPU support.

Usage

```
compile4gpu()
```

Details

Compiling GPU-compatible user-defined operators requires CUDA and nvcc (Nvidia compiler). If not available, user-defined operators will only be CPU-compatible.

Note: Default behavior is to compile GPU-compatible operators thus, if you do not modify rkeops options, it is optional to use the function compile4gpu.

When a GPU-compatible operator is compiled, you should call rkeops::use_gpu() to ensure that computation will be run on GPU (difference between compilation and runtime options). GPU-compatible operators can run on CPU.

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::compile4cpu(), rkeops::use_gpu(),

Examples

```
library(rkeops)
compile4gpu()
```

`compile_options` *Define a list of user-defined options used for compilation in* `rkeops` *package*

Description

To compile new user-defined operators, `rkeops` requires compilation options that control the compilation process and the way user-defined operators behave (precision, verbosity, use of GPU, storage order, debug flag, and path to different required files).

The function `default_compile_options` returns a list of class `rkeops_compile_options` with default values for the corresponding options (see Details).

Usage

```
compile_options(
  precision = "float",
  verbosity = FALSE,
  use_cuda_if_possible = TRUE,
  col_major = TRUE,
  debug = FALSE,
  rkeops_dir = NULL,
  build_dir = NULL,
  src_dir = NULL
)
```

Arguments

`precision`	string, precision for floating point computations (`float` or `double`). Default value is `float`.
`verbosity`	boolean indicator regarding verbosity level. Default value is `FALSE`.
`use_cuda_if_possible`	boolean indicator regarding compilation of the user-defined operators to be GPU-compatible (if possible on the system, i.e. if CUDA is available). Default value is `TRUE`. If set to `TRUE` and CUDA is not available, user-defined operators are compiled for CPU computations.
`col_major`	boolean indicator regarding storage order (default is TRUE).
`debug`	boolean indicator regarding debuging flag for compilation. Default value is `FALSE`.
`rkeops_dir`	string, path to `rkeops` install directory on the system. If NULL, default path described in Details section is used. Default value is `NULL`.
`build_dir`	string, path to the directory where new custom user-defined operators will be compiled. If NULL, default path described in Details section is used. Default value is `NULL`.
`src_dir`	string, path to `keops` (C++) source files required for compilation of user-defined operators. If NULL, default path described in Details section is used. Default value is `NULL`.

Details

The aforementioned compile options are the following:

- `rkeops_dir`: path to `rkeops` install directory on the system (e.g. /path/to/R_package_install/rkeops on Unix system).
- `build_dir`: path to directory where new user-defined operators will be compiled and corresponding share objects (`.so` files) will be saved (so that they can be found upon reuse to avoid useless recompilation). Default value is the `build` sub-folder in `rkeops` install directory (e.g. /path/to/R_package_install/rkeops/build on Unix system).
- `src_dir`: path to `keops` (C++) source files required for compilation of user-defined operators. Default value is the `include` sub-folder in `rkeops` install directory (e.g. /path/to/R_package_install/rkeops/include on Unix system).
- `precision`: precision for floating point computations (`float` or `double`). Default value is `float`.
- `verbosity`: 0-1 indicator (boolean) for verbosity level. Default value is `0`.
- `use_cuda_if_possible`: 0-1 indicator (boolean) regarding use of GPU in computations (if possible on the system). Default value is `1`.
- `col_major`: 0-1 indicator (boolean) regarding matrix storage order in C++ KeOps API. `1` is column-major storage (or `f_contiguous`) and `0` is row-major storage (or `c_contiguous`). Default value is `1`. This is independent from the storage order in R. Always keep in mind that matrices are stored with column-major order in R.
- `debug`: 0-1 indicator (boolean) regarding compilation debugging flag. `1` means that user-defined operators will be compiled with a debug flag, and `0` means no debug flag. Default value is `0`

Note on storage order: Column-major storage means that elements of each column of a matrix are contiguous in memory (called Fortran-style). Row-major storage means that each row of a matrix are contiguous in memory (called C-style). In R, matrices are stored with column-major order in R, so we recommend to use column-major order in for KeOps compilation (to avoid useless matrix conversion).

Note: Default options are set up when loading `rkeops`. To reset `rkeops` options to default, you should use the function `rkeops::set_rkeops_options()`. To set up a particular option, you should use the function `rkeops::set_rkeops_option()`.

Some wrappers are available to enable some compilation options, see `rkeops::compile4float32()`, `rkeops::compile4float64()`, `rkeops::compile4cpu()`, `rkeops::compile4gpu()`.

Value

a list (of class `rkeops_compile_options`) with the following elements:

`rkeops_dir`	string, path to `rkeops` install directory on the system.
`build_dir`	string, path to the directory where new custom user-defined operators will be compiled.
`src_dir`	string, path to `keops` (C++) source files required for compilation of user-defined operators.
`precision`	string, precision for floating point computations (`float` or `double`).

verbosity	integer, 0-1 indicator (boolean) for verbosity.
use_cuda_if_possible	integer, 0-1 indicator (boolean) regarding use of GPU in computations (if possible).
col_major	integer, 0-1 indicator (boolean) for storage order.
debug	integer, 0-1 indicator (boolean) for debugging flag.

Author(s)

Ghislain Durif

See Also

rkeops::default_compile_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option(), rkeops::compile4float32(), rkeops::compile4float64(), rkeops::compile4cpu(), rkeops::compile4gpu()

Examples

```
compile_options(
    precision = 'float', verbosity = FALSE,
    use_cuda_if_possible = TRUE,
    col_major = TRUE, debug = FALSE,
    rkeops_dir = NULL, build_dir = NULL,
    src_dir = NULL)
```

---

default_compile_options

Define a list of default options used for compilation in rkeops *package*

Description

To compile user-defined operators, rkeops requires compilation options.

The function default_compile_options returns a list with default values for the corresponding options (see Details).

Usage

```
default_compile_options()
```

Details

Please refer to rkeops::compile_options() for a detailed description of these options.

Note: Default options are set up when loading rkeops. To reset rkeops options to default, you should use the function rkeops::set_rkeops_options(). To set up a particular option, you should use the function rkeops::set_rkeops_option(). Some wrappers are available to enable some compilation options, see rkeops::compile4float32(), rkeops::compile4float64(), rkeops::compile4cpu(), rkeops::compile4gpu().

Value

a list of class rkeops_compile_options (see rkeops::compile_options() for the detailed output).

Author(s)

Ghislain Durif

See Also

rkeops::compile_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option(), rkeops::compile4float32(), rkeops::compile4float64(), rkeops::compile4cpu(), rkeops::compile4gpu()

Examples

```
default_compile_options()
```

---

default_runtime_options

Define a list of default options used at runtime in rkeops *package*

Description

To call user-defined operators, rkeops requires runtime options.

The function default_runtime_options returns a list with default values for the corresponding options (see Details).

Usage

```
default_runtime_options()
```

Details

Please refer to rkeops::runtime_options() for a detailed description of these options.

Note: Default options are set up when loading rkeops. To reset rkeops options to default, you should use the function rkeops::set_rkeops_options(). To set up a particular option, you should use the function rkeops::set_rkeops_option().

Some wrappers are available to enable some compilation options, see rkeops::use_cpu(), rkeops::use_gpu().

Value

a list of class rkeops_runtime_options (see rkeops::runtime_options() for the detailed output).

Author(s)

Ghislain Durif

See Also

rkeops::runtime_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option(), rkeops::use_cpu(), rkeops::use_gpu()

Examples

```
default_runtime_options()
```

`get_rkeops_option` *Get the current value of a specific compile or runtime options of* `rkeops`

Description

The function `get_rkeops_option` returns the current value of a specific `rkeops` option (in `R` global options scope) provided as input.

Usage

```
get_rkeops_option(option)
```

Arguments

`option`	string, name of the options to set up (see Details).

Details

`rkeops` global options includes two lists defining options used at compilation of user-defined operators or at runtime. These two list contains specific informations (see rkeops::compile_options() and rkeops::runtime_options() respectively, in particular for default values).

With the function `get_rkeops_option`, you get the value of a specific `rkeops` option among:

- `rkeops` compile options: rkeops_dir(not recommended),build_dir, src_dir(not recommended),precision, verbosity, use_cuda_if_possible, col_major(not recommended),debug‘
- `rkeops` runtime options: `tagCpuGpu`, `tag1D2D`, `tagHostDevice`, `device_id` These options are set with the functions rkeops::set_rkeops_options() and rkeops::set_rkeops_option(). To know which values are allowed for which options, you can check rkeops::compile_options() and rkeops::runtime_options().

Value

the value of the requested option (see Details).

Author(s)

Ghislain Durif

See Also

rkeops::get_rkeops_options(), rkeops::compile_options(), rkeops::runtime_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option()

Examples

```
library(rkeops)
# to get the GPU id used for computations
get_rkeops_option("device_id")
```

---

get_rkeops_options *Get the current* rkeops *options in* R *global options scope*

Description

rkeops uses two sets of options: compile options (see rkeops::compile_options()) and runtime options (see rkeops::runtime_options()). These options define the behavior of rkeops when compiling or when calling user-defined operators.

You can read the current states of rkeops options by calling get_rkeops_options().

Usage

```
get_rkeops_options(tag = NULL)
```

Arguments

tag	text string being "compile" or "runtime" to get corresponding options. If missing (default), both are returned.

Details

rkeops global options includes two lists defining options used at compilation of user-defined operators or at runtime. These two list contains specific informations (see rkeops::compile_options() and rkeops::runtime_options() respectively, in particular for default values).

If the tag input parameter is specified (e.g. "compile" or "runtime"), only the corresponding option list is returned.

These options are set with the functions rkeops::set_rkeops_options() and rkeops::set_rkeops_option(). To know which values are allowed for which options, you can check rkeops::compile_options() and rkeops::runtime_options().

Value

a list with rkeops current options values (see Details).

Author(s)

Ghislain Durif

See Also

rkeops::get_rkeops_option(), rkeops::compile_options(), rkeops::runtime_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option()

Examples

```
library(rkeops)
get_rkeops_options()
```

---

keops_grad *Compute the gradient of a rkeops operator*

Description

The function keops_grad defines a new operator that is a partial derivative from a previously defined KeOps operator supplied as input regarding a specified input variable of this operator.

Usage

```
keops_grad(operator, var)
```

Arguments

operator	a function returned by keops_kernel implementing a formula.
var	a text string or an integer number indicating regarding to which variable/parameter (given by name or by position starting at 0) the gradient of the formula should be computed.

Details

The use of the function keops_grad is detailed in the vignettes. Run browseVignettes("rkeops") to access the vignettes.

KeOps gradient operators are defined based on KeOps formula and on operator Grad. The function keops_grad is a wrapper to define a new formula deriving the gradient of the formula associated to a previously defined operator. The user just needs to choose regarding which variable (given by name or by position starting at 0), they want to compute the partial derivative.

The function keops_grad then calls the function rkeops::keops_kernel() to compile a new operator corresponding to the partial derivative of the input operator.

To decide regarding which variable the input operator should be derived, you can specify its name or its position starting as 0 with the input parameter var.

Value

a function that can be used to compute the value of the formula on actual data. This function takes as input a list of data corresponding to the formula arguments and return the computed values (generally a vector or a matrix depending on the reduction). It has an additional integer input parameter inner_dim indicating if the inner dimension (c.f. browseVignettes("rkeops")) corresponds to columns, i.e. inner_dim=1 (default), or rows, i.e. inner_dim=0, in the data.

Author(s)

Ghislain Durif

See Also

rkeops::keops_kernel()

Examples

```
## Not run:
set_rkeops_options()

# defining an operator (reduction on squared distance)
formula <- "Sum_Reduction(SqNorm2(x-y), 0)"
args <- c("x=Vi(0,3)", "y=Vj(1,3)")
op <- keops_kernel(formula, args)
# defining its gradient regarding x
grad_op <- keops_grad(op, var="x")

# data
nx <- 100
ny <- 150
x <- matrix(runif(nx*3), nrow=nx, ncol=3)     # matrix 100 x 3
y <- matrix(runif(ny*3), nrow=ny, ncol=3)     # matrix 150 x 3
eta <- matrix(runif(nx*1), nrow=nx, ncol=1)   # matrix 100 x 1

# computation
input <- list(x, y, eta)
res <- grad_op(input)

# OR you can directly define gradient in a formula
# defining a formula with a Gradient
formula <- "Grad(Sum_Reduction(SqNorm2(x-y), 0), x, eta)"
args <- c("x=Vi(0,3)", "y=Vj(1,3)", "eta=Vi(2,1)")
# compiling the corresponding operator
op <- keops_kernel(formula, args)

# data
nx <- 100
ny <- 150
x <- matrix(runif(nx*3), nrow=nx, ncol=3)     # matrix 100 x 3
y <- matrix(runif(ny*3), nrow=ny, ncol=3)     # matrix 150 x 3
eta <- matrix(runif(nx*1), nrow=nx, ncol=1)   # matrix 100 x 1

# computation
input <- list(x, y, eta)
res <- op(input)

## End(Not run)
```

keops_kernel *Defines a new operators*

Description

This function is the core of the KeOps library, it allows you to create new operators based on kernel operation and matrix reduction discribed as a mathematic formula.

Usage

```
keops_kernel(formula, args)
```

Arguments

formula	text string, an operator formula (see Details).
args	vector of text string, formula arguments (see Details).

Details

The use of the function keops_kernel is detailled in the vignettes, especially how to write formulae, specified input arguments, how to format data to apply the created operators, etc. Run browseVignettes("rkeops") to access the vignettes.

KeOps operators are defined thanks to formula, i.e. a text string describing the mathematical operations that you want to apply to your data, and a list defining the input arguments of your formula.

The function keops_kernel compiles and imports a new operator that implements the formala given in input, it returns a function that can be used to compute the result of the formula on actual data.

The returned function expects a list of arguments, as data matrices, whose order corresponds to the order given in args to keops_kernel. We use a list to avoid useless copies of data.

Note: Data are input as a list, because list are references and since argument passing is done by copy in R, it is better to copy a list of reference than the actual input data, especially for big matrices.

You should be careful with the input dimension of your data, to correspond to the input dimension specified in args (see inner ou outer dimension in browseVignettes("rkeops").

It is possible to compute partial derivatives of user defined operators with the function rkeops::keops_grad().

Value

a function that can be used to compute the value of the formula on actual data. This function takes as input a list of data corresponding to the formula arguments and return the computed values (generally a vector or a matrix depending on the reduction). It has an additional integer input parameter inner_dim indicating if the inner dimension (c.f. browseVignettes("rkeops")) corresponds to columns, i.e. inner_dim=1 (default), or rows, i.e. inner_dim=0, in the data.

Author(s)

Ghislain Durif

See Also

rkeops::keops_grad()

Examples

```
## Not run:
set_rkeops_options()

## Example 1
# Defining a function that computes for each j the sum over i
# of the scalar products between `x_i` and `y_j` (both 3d vectors),
# i.e. the sum over the rows of the result of the matrix product `X * t(Y)`
# where `x_i` and `y_j` are the respective rows of the matrices `X` and `Y`.
op <- keops_kernel(formula = "Sum_Reduction((x|y), 1)",
                   args = c("x=Vi(3)", "y=Vj(3)"))
# data
nx <- 10
ny <- 15
# x_i = rows of the matrix X
X <- matrix(runif(nx*3), nrow=nx, ncol=3)
# y_j = rows of the matrix Y
Y <- matrix(runif(ny*3), nrow=ny, ncol=3)
# compute the result (here, by default `inner_dim=1` and columns corresponds
# to the inner dimension)
res <- op(list(X,Y))

## Example 1 bis
# In example 1, the inner dimension (i.e. the commun dimension of vectors
# `x_i` and `y_j` corresponds to columns of the matrices `X` and `Y` resp.).
# We know consider the inner dimension to be the rows of the matrices `X`
# and `Y`.

# data
nx <- 10
ny <- 15
# x_i = columns of the matrix X
X <- matrix(runif(nx*3), nrow=3, ncol=nx)
# y_j = columns of the matrix Y
Y <- matrix(runif(ny*3), nrow=3, ncol=ny)
# compute the result (we specify `inner_dim=0` to indicate that the rows
# corresponds to the inner dimension)
res <- op(list(X,Y), inner_dim=0)

## Example 2
# Defining a function that computes the convolution with a Gaussian kernel
# i.e. the sum over i of `e^(lambda * ||x_i - y_j||^2) * beta_j` where `x_i`,
# `y_j` and `beta_j` are 3d vectors, and `lambda` is a scalar parameter.
op = keops_kernel(formula = "Sum_Reduction(Exp(lambda*SqNorm2(x-y))*beta, 1)",
                  args = c("x=Vi(3)", "y=Vj(3)",
                           "beta=Vj(3)", "lambda=Pm(1)"))

# data
```

```
nx <- 10
ny <- 15
# x_i = rows of the matrix X
X <- matrix(runif(nx*3), nrow=nx, ncol=3)
# y _j = rows of the matrix Y
Y <- matrix(runif(ny*3), nrow=ny, ncol=3)
# beta_j = rows of the matrix beta
beta <- matrix(runif(ny*3), nrow=ny, ncol=3)
# !! important !! y and beta should have the same dimension

# parameter
lambda <- 0.25

# compute the result
res <- op(list(X, Y, beta, lambda))

## End(Not run)
```

`runtime_options` *Define a list of user-defined options used at runtime in* `rkeops` *package*

Description

When calling user-defined operators, `rkeops` requires runtime options that control how the computations are done (memory management, data partition for parallelization, GPU device if relevant).

The function `runtime_options` returns a list of class `rkeops_runtime_options` with default values for the corresponding options (see Details).

Usage

```
runtime_options(tagCpuGpu = 0, tag1D2D = 0, tagHostDevice = 0, device_id = 0)
```

Arguments

`tagCpuGpu`	integer, indicator for CPU or GPU computations (see Details). Default value is `0`.
`tag1D2D`	integer, indicator regarding data partitioning for parallelization (see Details). Default value is `0`.
`tagHostDevice`	integer, indicator regarding the data location (see Details). Default value is `0`.
`device_id`	integer, id of GPU device on the machine (see Details). Default value is `0`.

Details

The aforementioned runtime options are the following:

- `tagCpuGpu`: `0` means computations on CPU, `1` means computations on GPU, `2` means computations on GPU using data on device (i.e. in GPU memory). Default value is `1`. The mode `2` is not available for the moment in R.

- tag1D2D: 0 means 1D parallelization (over rows of matrices), and 1 parallelization over blocks of rows and columns (useful with small columns large rows matrices). Default value is 0.
- tagHostDevice: 0 means that data are stored on host memory (i.e. in RAM), 1 means that data are stored on GPU memory. Default value is 0. The mode 1 is not available for the moment in R.
- device_id: id of GPU device (if relevant, i.e. with tagCpuGpu != 0) where the computations will be made. Default value is 0. Ideally, GPU assignation should be handled outside of R and rkeops.

Note: Default options are set up when loading rkeops. To reset rkeops options to default, you should use the function rkeops::set_rkeops_options(). To set up a particular option, you should use the function rkeops::set_rkeops_option().

Some wrappers are available to enable some compilation options, see rkeops::use_cpu(), rkeops::use_gpu().

Value

a list (of class rkeops_runtime_options) with the following elements:

tagCpuGpu	integer, indicator for CPU or GPU computations (see Details).
tag1D2D	integer, indicator regarding data partitioning for parallelization (see Details).
device_id	integer, id of GPU device on the machine (see Details).

Author(s)

Ghislain Durif

See Also

rkeops::default_runtime_options(), rkeops::set_rkeops_options(), rkeops::set_rkeops_option(), rkeops::use_cpu(), rkeops::use_gpu()

Examples

```
runtime_options(tagCpuGpu = 0, tag1D2D = 0,
                tagHostDevice=0, device_id = 0)
```

---

set_rkeops_option — *Set up a specific compile or runtime options of* rkeops *in* R *global options scope*

Description

The function set_rkeops_option allows to modify the value of a single specific rkeops options in R global options scope.

Usage

```
set_rkeops_option(option, value)
```

Arguments

option	string, name of the option to set up (see Details).
value	whatever value to assign to the chosen option (see Details).

Details

rkeops uses two sets of options: compile options (see rkeops::compile_options()) and runtime options (see rkeops::runtime_options()). These options define the behavior of rkeops when compiling or when calling user-defined operators.

With the function set_rkeops_option, you can set up a specific rkeops option among:

- rkeops compile options: rkeops_dir(not recommended),build_dir, src_dir(not recommended),precision, verbosity, use_cuda_if_possible, col_major(not recommended),debug‘
- rkeops runtime options: tagCpuGpu, tag1D2D, tagHostDevice, device_id with a value that you provide in input.

To know which values are allowed for which options, you can check rkeops::compile_options() and rkeops::runtime_options().

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::set_rkeops_options(), rkeops::compile_options(), rkeops::runtime_options(), rkeops::use_gpu(), rkeops::compile4gpu(), rkeops::get_rkeops_options()

Examples

```
library(rkeops)
# to enable GPU computing
set_rkeops_option("tagCpuGpu", 1)
# to set up the GPU id used for computations
set_rkeops_option("device_id", 0L) # integer value
```

set_rkeops_options *Initialize or update* rkeops *options in* R *global options scope*

Description

rkeops uses two sets of options: compile options (see rkeops::compile_options()) and runtime options (see rkeops::runtime_options()). These options define the behavior of rkeops when compiling or when calling user-defined operators.

If no input is provided, the functions set_rkeops_options initializes the rkeops options in the R global options scope (i.e. options available by calling options() or getOptions(<option_name>)) with default values.

If some input is provided, i.e. objects defining compile options and/or runtime options (see Details), rkeops global options are updated accordingly.

Usage

```
set_rkeops_options(
  custom_compile_options = NULL,
  custom_runtime_options = NULL
)
```

Arguments

custom_compile_options
: a list (of class rkeops_compile_options). See rkeops::compile_options() for a detailled description. Default value is NULL and default compile options are set up (see rkeops::default_compile_options()).

custom_runtime_options
: a list (of class rkeops_runtime_options). See rkeops::runtime_options() for a detailled description. Default value is NULL and default runtime options are set up (see rkeops::default_runtime_options()).

Details

rkeops global options includes two lists defining options used at compilation of user-defined operators or at runtime. These two list contains specific informations (see rkeops::compile_options() and rkeops::runtime_options() respectively, in particular for default values).

In order to update, the corresponding options, user should provide objects returned by the functions rkeops::compile_options() and/or rkeops::runtime_options() respectively, being lists of class rkeops_compile_options and rkeops_runtime_options respectively, with specific attributes.

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::set_rkeops_option(), rkeops::compile_options(), rkeops::runtime_options(), rkeops::default_compile_options(), rkeops::default_runtime_options(), rkeops::use_gpu(), rkeops::compile4gpu(), rkeops::get_rkeops_options()

Examples

```
set_rkeops_options()
```

---

use_cpu *Disable GPU-computing when calling user-defined operators*

Description

Set up rkeops runtime options to use CPU computing when calling user-defined operators.

Usage

```
use_cpu()
```

Details

Note: The default behavior in rkeops is to use CPU computing, thus calling the function use_gpu is mandatory to run computations on GPU.

To enable GPU computing, run rkeops::use_gpu().

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::compile4cpu(), rkeops::compile4gpu(), rkeops::use_gpu()

Examples

```
library(rkeops)
use_cpu()
```

use_gpu *Enable GPU-computing when calling user-defined operators*

Description

Set up rkeops runtime options to use GPU computing when calling user-defined operators.

Usage

```
use_gpu(device = 0)
```

Arguments

device	integer, GPU device id to be used for computations. Default is 0. It is recommended to use default GPU and manage GPU assignment outside R by setting the environment variable CUDA_VISIBLE_DEVICES.

Details

If you have compiled GPU-compatible operators (see rkeops::compile4gpu()), you can call the function use_gpu to specificly run computations on GPU.

Note: The default behavior in rkeops is to use CPU computing, thus calling the function use_gpu is mandatory to run computations on GPU.

To disable GPU computing, run rkeops::use_cpu().

Value

None

Author(s)

Ghislain Durif

See Also

rkeops::compile4cpu(), rkeops::compile4gpu(), rkeops::use_cpu()

Examples

```
library(rkeops)
use_gpu()
```

Index