Package 'gpuMagic'

November 13, 2024

Type Package

Title An openCL compiler with the capacity to compile R functions and run the code on GPU

Version 1.22.0

Date 2018-10-07

Description The package aims to help users write openCL code with little or no effort. It is able to compile an user-defined R function and run it on a device such as a CPU or a GPU. The user can also write and run their openCL code directly by calling .kernel function.

License GPL-3

LinkingTo Rcpp

Depends R (>= 3.6.0), methods, utils

Imports Deriv, DescTools, digest, pryr, stringr, BiocGenerics

Suggests testthat, knitr, rmarkdown, BiocStyle

biocViews Infrastructure

BugReports https://github.com/Jiefei-Wang/gpuMagic/issues

SystemRequirements 1. C++11, 2. a graphic driver or a CPU SDK. 3. ICD loader For Windows user, an ICD loader is required at C:/windows/system32/OpenCL.dll (Usually it is installed by the graphic driver). For Linux user (Except mac): ocl-icd-opencl-dev package is required. For Mac user, no action is needed for the system has installed the dependency. 4. GNU make

RoxygenNote 7.1.0

Roxygen list(markdown = TRUE)

Collate 'ParserFramework.R' 'RCParserFunc_Rlevel.R' 'RCParser_elementOP.R' 'RCParserFunc.R' 'RProfilerFunc.R' 'pkgFunc.R' 'RCParser.R' 'RCParserSupportFunc.R' 'ROptimizer.R' 'ROptimizerSupportFunc.R' 'RParser.R' 'RParserSupportFunc.R' 'RProfiler.R' 'RProfilerSettings.R' 'RProfilerSupportFunc.R' 'extCodeManager.R' 'tools.R' 'hash.R' 'gpuApply.R' 'gpuFunctions.R' 'gpuMatix-class.R' 'gpuRefAddress.R' 'gpuResourcesManager.R' 'kernelManager.R' 'managerTools.R' 'toolsSupportFuncs.R' 'utils.R' 'utils_gpuApply.R' 'varInfoManager.R' 'zzz.R'

VignetteBuilder knitr

Encoding UTF-8

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

git_branch RELEASE_3_20

git_last_commit cc5ec08

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-11-12

Author Jiefei Wang [aut, cre],
Martin Morgan [aut]

Maintainer Jiefei Wang <szwjf08@gmail.com>

Contents

`.kernel` *Excute the openCL function*

Description

The function serves as a bridge between R and openCL, it sends the openCL code and R matrix object to the device and excutes it on the device. The function has an auto-type ability which can make the openCL code independent with the type of its function argument, see detail and examples for the usage.

Usage

```
.kernel(
  src = "",
  kernel,
  parms,
  .device = "auto",
  .globalThreadNum = "length(FirstArg)",
  .options = kernel.getOption()
)
```

Arguments

src	the source code, it can be either a file directory or the code
kernel	the kernel function that will be called on the device
parms	a list containing the function arguments. The number of elements in the list has to match the number of function arguments.
.device	the device that will excute the function. If not specified, all the selected devices will be used.
.globalThreadNum	the number of threads that will be created to excute the kernel. If not specified, the length of the first argument will be used as the thread number
.options	the kernel options

Details

The function `.kernel()` is the low level API to communicate with openCL device. It provides a way to run the customized code on the device, the source code should be openCL code and the kernel is the kernel function that you want to run on the device.

You can specify with device the code should be run on by specifying the `.device` argument. By default, if you do not specify any device, the first device in the device list will be used

The argument `.globalThreadNum` specifys the number of threads that will be used to excute the kernel. The concept is the same as ‘global_work_size“ in openCL functions

There are multiple options that you can change in the kernel function. You can call the function `kernel.getOption()` to obtain the default setting. The most distinguishable feature in this package is probably the auto type function, which can set the type of the kernel arguments as an macro in the openCL code. This feature allows the user to create a type-free code. If the `kernelOption$autoType` in `.options` is true(Default), four macros will be defined, they are(X is the position of the function arguments):

autoX: The variable type

gAutoX: Short for global autoX

lAutoX: short for local autoX

autoX_v4: Define a vector of length 4 with the same variable type as the X th function argument

Please refer to the example for the usage

Value

A vector or a matrix

Examples

```
#The GPU code
code='
kernel void matAdd(gAuto1* A,gAuto2* B,gAuto3* C,gAuto4* size){
uint col_id=get_global_id(0);
uint rowNum=*size;
for(uint i=0;i<rowNum;i++){
C[i+col_id*rowNum]=A[i+col_id*rowNum]+B[i+col_id*rowNum];
}
}
'
#Create data in R
m=100
n=200
A=matrix(runif(m*n),m,n)
B=matrix(runif(m*n),m,n)
#Send the data to GPU
A_dev=gpuMatrix(A,type='double')
B_dev=gpuMatrix(B,type='double')
#Create an empty data matrix in GPU
C_dev=gpuEmptMatrix(row=m,col=n,type='double')

#Get the default options
options=kernel.getOption()
#Run the GPU function with n threads, each thread computes one column addition
.kernel(src = code,kernel='matAdd',parms=list(A_dev,B_dev,C_dev,m),
.globalThreadNum = n,.options = options)

#This is just a patch to fix check error
if(!is.null(C_dev)){
#Retrieve the data
C_dev=download(C_dev)
C=as.matrix(C_dev)
#Check the error
range(C-A-B)
}
```

`as.matrix.gpuMatrix` *Convert the gpuMatrix object into a matrix*

Description

The function will convert the gpuMatrix object into a matrix, if you have run any GPU functions on the gpuMatrix object, please call `download(x)` to synchronize the data before calling this function.

Usage

```
## S3 method for class 'gpuMatrix'
as.matrix(x, ...)
```

Arguments

`x`	an gpuMatrix object
`...`	This argument is only for compatibility. It does not take any effect.

Value

A matrix

`as.vector.gpuMatrix` *Convert the gpuMatrix object into a vector*

Description

The function will convert the gpuMatrix object into a vector, if you have run any GPU functions on the gpuMatrix object, please call `download(x)` to synchronize the data before calling this function.

Usage

```
## S3 method for class 'gpuMatrix'
as.vector(x, mode = NULL)
```

Arguments

`x`	an gpuMatrix object
`mode`	This argument is only for compatibility. It does not take any effect.

Value

A vector

`compileGPUCode` *Compile the R function without excute it in the device.*

Description

Compile the R function without excute it in the device.

Usage

```
compileGPUCode(
  X,
  FUN,
  ...,
  .macroParms = NULL,
  .options = gpuSapply.getOption()
)
```

Arguments

`X`	a vector that `FUN` will loop over.
`FUN`	The function to be applied to each element of `X`
`...`	optional arguments to `FUN`
`.macroParms`	The function argument that will be treated as macro in the code. If an argument is treated as macro, its value cannot be changed by the code
`.options`	The package and openCL compilation options, please call `gpuSapply.getOption()` to get all the available options

Value

A list of compilation information

Examples

```
#matrix add function
matAdd = function(ind,A,B){
C = A[,ind]+B[,ind]
return(C)
}

n = 100
m = 200
#Create the data
A = matrix(runif(n*m),n,m)
B = matrix(runif(n*m),n,m)
#Compile the R code
res = compileGPUCode(1:m,matAdd,A,B)
#print GPU code
cat(res$gpu_code)
```

---

getDeviceList *Query and select the devices*

Description

This is a set of functions to query the device information and select which device should be used in the computation

Usage

```
getDeviceList()

getDeviceInfo(i)

getCurDevice()

setDevice(i)

getDeviceIndex()

getJobStatus(i)
```

Arguments

i	A 1-based device index, it should be an integer

Details

'getDeviceList()': The function is used to obtain all the opencl-enable devices

'getDeviceInfo()': Get the ith device information, call 'getDeviceList()' first to figure out the index before using this function

'getCurDevice()': Get the information of the current devices

'setDevice()': Set which device will be used in the opencl, call 'getDeviceList()' first to figure out the index before use this function

'getDeviceIndex()': Get the index of the current devices

'getJobStatus()': Query the current job status in a device

Value

'getDeviceList()': A data.frame that contains all device info

'getDeviceInfo()': A list with the device information

'getCurDevice()': No return value, the result will be printed in the console

'setDevice()': No return value

'getDeviceIndex()': An integer representing the device index

'getJobStatus()': A character representing the device status

Examples

```
#Get the available devices
getDeviceList()

#Get the information of the first device
getDeviceInfo(1)
#Get the information of current used devices
getCurDevice()
#Use the first device
setDevice(1)
#Use two devices
#setDevice(c(1,2))
#Get the index of the current devices
getDeviceIndex()
#Get the job status in the first device
getJobStatus(1)
```

---

gpuMagic.getAvailableType

Get all the available openCL variable type

Description

Get all the available openCL variable type

Usage

```
gpuMagic.getAvailableType()
```

Value

A vector of all the available data type.

Examples

```
gpuMagic.getAvailableType()
```

---

gpuMagic.getMemUsage *Get the device memory usage*

Description

The function will print the memory usage on the console

Usage

```
gpuMagic.getMemUsage()
```

Value

No return value, the result will be printed in the console.

Examples

```
gpuMagic.getMemUsage()
```

---

gpuMagic.getOptions *Get the openCL options*

Description

The functions gets the computing precision when compile the GPU code and the number of workers in a computing group.

Usage

```
gpuMagic.getOptions(opt = "all")
```

Arguments

opt	The options that the function will return. It can be either 'all' or a vector of the option names.

Details

The fields `default.float`, `default.int` and `default.index.type` are used to control the computing precision. When transferring data from R to GPU, if the data in R has a numeric or double storage mode, `default.float` will be used to convert data type. Similarly, If the data has an Integer storage model. `default.int` will be used.

`default.index.type` controls the variable type for the for loop index, variable dimension etc.

`default.thread.num` is used to control the number of workers in a group in openCL. It is not expected to be changed unless you know what you are doing.

Value

A list of the options

Examples

```
#Get all the available options
opt=gpuMagic.getOptions()
opt
```

---

gpuMagic.setOptions *Set the openCL options*

Description

The functions set the computing precision when compile the GPU code and the number of workers in a computing group.

Usage

```
gpuMagic.setOptions(...)
```

Arguments

...	There are two possible ways to set the options. You can either provide 1. A named argument which name is the same as the name of the options. 2. An R object obtaining from `gpuMagic.getOptions()` to change the options.

Value

No return value

See Also

`gpuMagic.getOptions()` for the name of the options.

Examples

```
#Get all the available options
opt=gpuMagic.getOptions()
#change the default float type
opt$default.float='float'
#set the options
gpuMagic.setOptions(opt)

#set the options(Alternative way)
gpuMagic.setOptions(default.float='float')
```

gpuMatrix *gpuMatrix class*

Description

gpuMatrix class

Usage

```
gpuMatrix(data, type = "auto", device = "auto")

gpuEmptMatrix(row = 1, col = 1, type = "auto", device = "auto")

upload(x)

download(x)

## S4 method for signature 'gpuMatrix'
download(x)

## S4 method for signature 'ANY'
download(x)

nrow(x)

## S4 method for signature 'gpuMatrix'
dim(x)

## S4 method for signature 'gpuMatrix'
length(x)

getSize(x)
```

Arguments

data	It can be a matrix or an R object that can be converted into a matrix.
type	The precision that is used to store the data, the default is gpuMagic.getOptions('default.float').
device	The device that the data is sent to, the default is the first device.
row, col	the row and column number of the matrix
x	an gpuMatrix object

Details

gpuMatrix(): Create a matrix in an openCL device

gpuEmptMatrix(): Create an empty matrix without initialization in an openCL device

upload(): The function will automatically be called when an gpuMatrix object is created. It is only needed when you want to update value of the matrix.

download(): Get the data from the device. You should explicitly call it when you want to collect the data from the device.

nrow(),ncol(): return the number of rows or columns present in x

dim(): Retrieve the dimension of an gpuMatrix object

length(): Get the length of an gpuMatrix object.

'getSize()': Get the matrix size in byte

Value

gpuMatrix(): A gpuMatrix object

gpuEmptMatrix(): A gpuMatrix object

Examples

```
n=10
m=20
A=matrix(runif(n*m),n,m)
#Create a 64 bit floating point GPU matrix
A_dev=gpuMatrix(A,'double')

#Create an empty matrix
B_dev=gpuEmptMatrix(row=n,col=m)
```

---

gpuSapply *A GPU version of the sapply function*

Description

Please refer to sapply to see the basic usage

Usage

```
gpuSapply(
  X,
  FUN,
  ...,
  .macroParms = NULL,
  .device = "auto",
  loading = "auto",
  .options = gpuSapply.getOption()
)
```

Arguments

X	a vector that FUN will loop over.
FUN	The function to be applied to each element of X
...	optional arguments to FUN
.macroParms	The function argument that will be treated as macro in the code. If an argument is treated as macro, its value cannot be changed by the code
.device	the device ID(s) indicates the device that the function will be excuted on. Running the code on Multiple devices is supported but is still under development
loading	The loading of each device, only useful when having multiple devices.
.options	The package and openCL compilation options, please call gpuSapply.getOption() to get all the available options

Details

This function compiles the R code and runs it on the openCL-compatible devices. The usage is similar to the sapply function with some addtional opencl-related arguments.

Value

A vector or a matrix

Examples

```
#matrix multiplication function
matMul = function(ind,A,B){
C = A%*%B[,ind]
return(C)
}

n = 100
m = 200
k = 100
#Create the data
A = matrix(runif(n*m),n,m)
B = matrix(runif(k*m),m,k)
#Perform matrix multiplication
#GPU
res_gpu = gpuSapply(1:k,matMul,A,B)
#CPU
res_cpu = sapply(1:k,matMul,A,B)

#error
range(res_gpu-res_cpu)
```

---

gpuSapply.getOption *Get the package compilation options*

Description

Get the package compilation options, the openCl compilation options(`kernel.getOption()`) are also included.

Usage

```
gpuSapply.getOption()
```

Details

There are a few options that are allowed to be changed, they are: `sapplyOption.debugCode`: Replace the compiled GPU code with your customized code, this option is useful when you want to debug the compiled code, or when you want to customize the compiled code.

`sapplyOption.compileEveryTime`: Specify whether you want the compiler to compile the R code everytime.

Value

An options class

Examples

```
opt=gpuSapply.getOption()
```

`gpu_cast_float` *Internal usage only, the package export this function only for the other package to access.*

Description

Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.
Internal usage only, the package export this function only for the other package to access.

Usage

```
gpu_cast_float(x)

gpu_cast_double(x)

gpu_cast_uint(x)

gpu_cast_int(x)

gpu_cast_long(x)

gpu_cast_ulong(x)

isgreater(x, y)

## S3 method for class 'extCode'
extractVars(x)

extractVars(x)

## Default S3 method:
extractVars(x)

## S3 method for class 'expression'
extractVars(x)

## S3 method for class 'varInfo'
extractVars(x)
```

Arguments

x	Internal usage only
y	Internal usage only

Value

A double type data

A vector of variables

Examples

```
gpu_cast_float(10)
#Just to make biocCheck happy with that.
```

---

kernel.getOption *Get the openCL compilation options*

Description

Get the openCL compilation options

Usage

```
kernel.getOption()
```

Details

#' verbose turn the verbose mode on and off.

kernelOption$localThreadNum controls the local thread number in each group, the local thread number should be a divisor of the argument .globalThreadNum. If it is set to auto, the suggested number of local thread number will be obtained from openCL API and reduced to a divisor of .globalThreadNum.

kernelOption$localThreadNumMacro specifys whether the local thread number should be inserted into the code as an macro. If it is TRUE, the macro cl_local_thread_num will be defined. It is useful when you want to dynamically allocate the memory \(Mostly local memory\) according to the local thread number

kernelOption$signature This is for internal usage only, please do not change it

kernelOption$flag The openCL compiler flag.

kernelOption$autoType Determine whether the type of kernel arguments should be defined as an macro, see the .kernel document for detail

Value

A list of available options

Examples

```
opt=kernel.getOption()
opt
```

`Matrix` *Create a matrix*

Description

The function create a matrix, it is only useful in the openCL functions. it can also be called in R, but its argument may or may not take any effect.

Usage

```
Matrix(
  nrow = 1,
  ncol = 1,
  precision = GPUVar$default_float,
  constDef = FALSE,
  shared = FALSE,
  location = "global"
)
```

Arguments

`nrow`, `ncol`	The matrix dimension.
`precision`	The variable type, please refer to `gpuMagic.getAvailableType()` to see the available data type.
`constDef`	Specify if the variable can be redefined. The package will automatically update the variable definition when it is needed, if you do not need this feature, you can manually turn the feature off. It is useful in some special cases such as turning off the auto update to do the integer division (By default, the package will convert the variable to the default float type before doing the division).
`shared`	If the matrix is shared by all the workers in a work group. Do not use it if you don't know its meaning.
`location`	The physical memory location of the matrix, it can be either 'global' or 'local'. Do not use it if you don't know its meaning.

Value

a matrix initialize with 0.

Examples

```
#Create a 10-by-10 matrix
A=Matrix(10,10)
```

`ncol,gpuMatrix-method` *gpuMatrix class*

Description

gpuMatrix class

Usage

```
## S4 method for signature 'gpuMatrix'
ncol(x)
```

Arguments

`x`	an gpuMatrix object

Details

`gpuMatrix()`: Create a matrix in an openCL device

`gpuEmptMatrix()`: Create an empty matrix without initialization in an openCL device

`upload()`: The function will automatically be called when an gpuMatrix object is created. It is only needed when you want to update value of the matrix.

`download()`: Get the data from the device. You should explicitly call it when you want to collect the data from the device.

`nrow()`,`ncol()`: return the number of rows or columns present in `x`

`dim()`: Retrieve the dimension of an gpuMatrix object

`length()`: Get the length of an gpuMatrix object.

'getSize()': Get the matrix size in byte

Value

`gpuMatrix()`: A gpuMatrix object

`gpuEmptMatrix()`: A gpuMatrix object

Examples

```
n=10
m=20
A=matrix(runif(n*m),n,m)
#Create a 64 bit floating point GPU matrix
A_dev=gpuMatrix(A,'double')

#Create an empty matrix
B_dev=gpuEmptMatrix(row=n,col=m)
```

print.options *Print the available options in a pretty format*

Description

Print the available options in a pretty format

Usage

```
## S3 method for class 'options'
print(x, ...)

## S3 method for class 'plainText'
print(x, ...)

## S3 method for class 'deviceList'
print(x, ...)

## S3 method for class 'varInfo'
print(x, simplify = TRUE, printDef = FALSE, ...)
```

Arguments

x	an options object.
...	just for making the package checking happy.
simplify	Specify whether only the important properties should be printed
printDef	Whether the variable definition should be printed(version=0)

Value

No return value, the result will be printed in the console

Examples

```
opt=gpuMagic.getOptions()
print(opt)
```

return_nocpy *No copy method*

Description

Doing some opration without copying memory

Usage

```
return_nocpy(x)

t_nocpy(x)
```

Arguments

x	an object

Details

`return_nocpy`: The usage of the `return_nocpy` is same as `return`. This feature is for openCL code only, if it is called in R, the function `return()` will be called instead

`t_nocpy`: The function transposes `x` without allocating the memory. It only works for the openCL code, if it is called in R, the function `t()` will be called instead

Value

`return_nocpy`: No return value

`t_nocpy`: the transpose of `x`

Examples

```
x=matrix(0)
#return_nocpy(x)
#x=t_nocpy(x)
```

Scalar *Create a scalar variable*

Description

The function will create a scalar variable, it is only useful in the openCL functions. It can also be called in R, but its argument will not take any effect.

Usage

```
Scalar(precision = GPUVar$default_float, constDef = FALSE)
```

Arguments

precision	The variable type, please refer to `gpuMagic.getAvailableType()` to see the available data type.
constDef	Specify if the variable can be redefined. The package will automatically update the variable definition when it is needed, if you do not need this feature, you can manually turn the feature off. It is useful in some special cases such as turning off the auto update to do the integer division (By default, the package will convert the variable to the default float type before doing the division).

Value

a variable initialize with 0.

Examples

```
a=Scalar(precision='double',constDef=FALSE)
```

subRef *Get a reference of the subset of a matrix*

Description

The function will get a reference of the matrix subset. This is a 0-copy method, which means any change in the reference variable will cause the change in the original matrix. The function is useful when the GPU memory is limited or you do not want to create a copy the data. DO NOT call this function in R, this is for openCL code only(eg. gpuSapply).

Usage

```
subRef(variable, i = "", j = "")
```

Arguments

variable	the matrix you want to subset
i	the index of a vector or the row index of a matrix
j	(Optional) The column index of a matrix

Details

The package implement this function purely using the code. it will not actually be called on device side. For example, if we have the following code:

```
#Alternative of B=A[ind]
B=subRef(A,ind)
a=B[2]
```

In the compilation stage, the code will be changed to

```
a=A[ind[2]]
```

The variable B does not exist in the code after the compilation and therefore no memory is allocated for it.

Value

A reference to the subset of a matrix

Warning

Since this feature is implemented like a macro, so it is possible to change the value of `ind` after the matrix B is created and before you modify the matrix B. In such case, it may cause an unexpected error. It is a good practice to keep the `ind` same while using the subset reference.

Examples

```
#create data
ind=1:10
A=matrix(0,100,100)
#Use the one-index subsetting, create a vector of length 10
B=subRef(A,ind)
#Subsetting the matrix A,create a 10-by-10 matrix
C=subRef(A,ind,ind)
#row subsetting
D=subRef(A,ind,)
#column subsetting
E=subRef(A,,ind)
```

[,gpuMatrix,ANY,ANY,missing-method

extract/set parts of the data in gpuMatrix object

Description

extract/set parts of the data in gpuMatrix object

Usage

```
## S4 method for signature 'gpuMatrix,ANY,ANY,missing'
x[i = NA, j = NA, ..., drop = TRUE]

## S4 replacement method for signature 'gpuMatrix,ANY,ANY,numeric'
x[i, j, ...] <- value
```

Arguments

x	an gpuMatrix object
i, j	indices specifying elements to extract or replace. The index j can be missing or empty.
...	This argument is only for compatibility. It does not have any effect.
drop	For matrices and arrays. If TRUE the result is coerced to the lowest possible dimension.
value	The value you want to set

Value

A matrix subset

no return value

Index