Package 'OpenCL'

January 31, 2019

Version 0.1-3.1

Title Interface Allowing R to Use OpenCL

Author Simon Urbanek <Simon.Urbanek@r-project.org>

Maintainer Simon Urbanek <Simon.Urbanek@r-project.org>

Depends R (>= 2.0.0)

Description Provides an interface to OpenCL, allowing R
to leverage computing power of GPUs and other HPC accelerator
devices.

License BSD

SystemRequirements OpenCL library

URL http://www.rforge.net/OpenCL/

Repository CRAN

Date/Publication 2019-01-31 17:27:45 UTC

NeedsCompilation yes

R topics documented:

clFloat *clFloat represents a single-precision vector that can be used with OpenCL.*

Description

clFloat is a constructor of a single-precision vector - an object of the class "clFloat".

The purpose of this type is to serve as a pass-though for OpenCL calls that involve single precision vectors such that they do not need to be converted to/from double precision and thus incurring the corresponding penalty. This type is not designed to be a full numeric vector replacement and thus opeartion such as arithmetics are intentionally not implemented.

The current internal representation is a raw vector of the machine-specific representation of the C float type but this may change in the future so no code should rely on it. It also implies that serialization is only compatible for machines of the same endianness.

as.clFloat coerces objects into single-precision vectors essentially by calling clFloat(as.numeric(x)).

is.clFloat returns TRUE the x is an object of the class clFloat

Some most basic methods such as length or print are implemented as well as basic coersion methods.

Usage

```
clFloat(x)
as.clFloat(x)
is.clFloat(x)
## S3 method for class 'clFloat'
as.double(x, ...)
## S3 method for class 'clFloat'
as.integer(x, ...)
## S3 method for class 'clFloat'
as.character(x, ...)
## S3 method for class 'clFloat'
print(x, ...)
## S3 method for class 'clFloat'
length(x)
## S3 replacement method for class 'clFloat'
length(x) <- value
## S3 method for class 'clFloat'
x[...]
## S3 replacement method for class 'clFloat'
x[...] <- value
```

Arguments

x	object
...	arguments passed to subsequent methods
value	new length or values

Author(s)

Simon Urbanek

See Also

oclRun

Examples

```
x <- clFloat(1:10/2)
print(x)
length(x)
as.double(x)
as.character(x)
as.integer(x)
is.clFloat(x)
identical(x, as.clFloat(as.numeric(x)))
x[1:5]
x[1] <- 0

## clFloat is the return type of oclRun(..., native.result=TRUE)
## for single-precision kernels. It can also be used instead of
## numeric vectors in such kernels to avoid conversions.
## See oclRun() examples.
```

---

oclDevices *Get a list of OpenCL devices.*

Description

oclDevices retrieves a list of OpenCL devices for the given platform.

Usage

```
oclDevices(platform = oclPlatforms()[[1]], type = "default")
```

Arguments

platform	OpenCL platform (see oclPlatforms)
type	Desired device type, character vector of length one. Valid values are "cpu", "gpu", "accelerator", "all", "default". Partial matches are allowed.

Value

List of devices. May be empty.

Author(s)

Simon Urbanek

See Also

oclPlatforms

Examples

```
p <- oclPlatforms()
if (length(p)) print(oclDevices, "all")
```

---

oclInfo *Retrieve information about an OpenCL object.*

Description

Some OpenCL obejcts have information tokens associated with them. For example the device obejct has a name, vendor, list of extensions etc. oclInfo returns a list of such properties for the given object.

Usage

```
oclInfo(item)
## S3 method for class 'clDeviceID'
oclInfo(item)
## S3 method for class 'clPlatformID'
oclInfo(item)
## S3 method for class 'list'
oclInfo(item)
```

Arguments

item	object to retrieve information properties from

Value

List of properties. The properties vary by object type. Some common properties are "name", "vendor", "version", "profile" and "exts".

Author(s)

Simon Urbanek

Examples

```
p <- oclPlatforms()
if (length(p)) {
  print(oclInfo(p[[1]]))
  d <- oclDevices(p[[1]])
  if (length(d)) print(oclInfo(d))
}
```

oclPlatforms *Retrieve available OpenCL platforms.*

Description

`oclPlatforms` retrieves all available OpenCL platforms.

Usage

```
oclPlatforms()
```

Value

List of available OpenCL platforms.

Author(s)

Simon Urbanek

See Also

`oclDevices`

Examples

```
print(oclPlatforms())
```

oclRun *Run a kernel using OpenCL.*

Description

`oclRun` is used to execute code that has been compiled for OpenCL.

`oclResult` collects results from an asynchronous `oclRun` call.

Usage

```
oclRun(kernel, size, ..., native.result = FALSE, wait = TRUE, dim = size)
oclResult(context, wait = TRUE)
```

Arguments

kernel	kernel object as obtained from oclSimpleKernel
size	length of the output vector
...	additional arguments passed to the kernel
native.result	logical scalar, if TRUE then the result from a single-precision kernel is not converted to double-precision but returned as a clFloat object.
wait	logical scalar, if TRUE then oclRun waits for the operation to finish and returs the result. Otherwise the kernel is only enqueued, so it will be run in parallel to R and have to be collected later with oclResult.
dim	numeric vector describing the global work dimensions, i.e., the index range that the kernel will be run on. The kernel can use get_global_id(n) to obtain the (n + 1)-th dimension index and get_global_size(n) to get the dimension. OpenCL standard supports only up to three dimensions, you can use use index vectors as arguments if more dimensions are required. Note that dim is not necessarily the dimension of the result although it can be.
context	context object that was returned by oclRun(..., wait = FALSE) call.

Details

oclRun pushes kernel arguments, executes the kernel and retrieves the result. The kernel is expected to have either __global double * or __global float * type (write-only) as the first argument which will be used for the result and const int second argument denoting the result length. All other arguments are assumed to be read-only and will be filled accoding to the ... values. Scalar values (vectors of length one) are passed as constants, vectors are passed as global objects. Only numeric (int*, double*), clFloat (float*) and logical (int*) vectors are supported as kernel arguments. Numeric (double-precision) vectors are converted to single-precision automatically when using single-precision kernel. The caller is responsible for matching the argument types according to the kernel in a way similar to .C and .Call.

oclResult retrieves the result of a previous operation that was enqueued using oclRun(..., wait = FALSE). If oclResult(..., wait = FALSE) is used then NULL is returned in case the result is not ready yet. Note that results can be collected only once and the context object becomes invalid after a successful call to oclResult since all associated OpenCL objects are released.

Value

oclRun: for wait = TRUE is the result of the operation, a numeric vector of the length size. Otherwise oclRun returns a call context object that can be used by oclResult to retrieve the result.

oclResult: Result of the previously started operation or NULL if wait=FALSE and the operation has not completed yet.

Author(s)

Simon Urbanek

See Also

oclSimpleKernel, clFloat

Examples

```
library(OpenCL)
p = oclPlatforms()
d = oclDevices(p[[1]])

code = c("
__kernel void dnorm(
  __global float* output,
 const unsigned int count,
  __global float* input,
 const float mu, const float sigma)
{
  int i = get_global_id(0);
  if(i < count)
      output[i] = exp(-0.5f * ((input[i] - mu) / sigma) * ((input[i] - mu) / sigma))
      / (sigma * sqrt( 2 * 3.14159265358979323846264338327950288 ) );
};")
k.dnorm <- oclSimpleKernel(d[[1]], "dnorm", code, "single")
f <- function(x, mu=0, sigma=1, ...)
  oclRun(k.dnorm, length(x), x, mu, sigma, ...)

## expect differences since the above uses single-precision but
## it should be close enough
f(1:10/2) - dnorm(1:10/2)

## this is optional - use floats instead of regular numeric vectors
x <- clFloat(1:10/2)
f(x, native.result=TRUE)

## does the device support double-precision?
if (any(grepl("cl_khr_fp64", oclInfo(d[[1]])$exts))) {
code = c("#pragma OPENCL EXTENSION cl_khr_fp64 : enable
__kernel void dnorm(
  __global double* output,
 const unsigned int count,
  __global double* input,
 const double mu, const double sigma)
{
  int i = get_global_id(0);
  if(i < count)
      output[i] = exp(-0.5f * ((input[i] - mu) / sigma) * ((input[i] - mu) / sigma))
      / (sigma * sqrt( 2 * 3.14159265358979323846264338327950288 ) );
};")
k.dnorm <- oclSimpleKernel(d[[1]], "dnorm", code, "double")
f <- function(x, mu=0, sigma=1)
  oclRun(k.dnorm, length(x), x, mu, sigma)

## probably not identical, but close...
f(1:10/2) - dnorm(1:10/2)
} else cat("\nSorry, your device doesn't support double-precision\n")

## Note that in practice you can use precision="best" in the first
```

```
## example which will pick "double" on devices that support it and
## "single" elsewhere
```

`oclSimpleKernel` *Create and compile OpenCL kernel code.*

Description

`oclSimpleKernel` creates a kernel object by compiling the supplied code. The kernel can them be used in `oclRun`.

Usage

```
oclSimpleKernel(device, name, code, precision = c("single", "double", "best"))
```

Arguments

`device`	Device (element of the list returned by `oclDevices()`) to compile the kernel on.
`name`	Name of the kernel function - must match the name used in the supplied code.
`code`	character vector containg the code. The code will be concatenated (as-is, no newlines are added!) by the engine.
`precision`	precision of all floating-point arguments in the kernel. Note that R uses only double-precision floating point representation, so single-precision computation requires temporary conversion of all input and output values and thus has significant overhead. However, not all devices support double-precision computation. If "best" is used then the kernel code is expected to use single-precision but it will be automatically augmented to double-precision (by replacing words "float" with "double" in the code and adding the `cl_khr_fp64` pragma) if supported by the device.

Details

`oclSimpleKernel` creates a new OpenCL context, then creates and builds the program specified by code and finally creates a kernel from the program.

The kernel built by this function is simple in that it can have exactly one vector output and arbitrarily many inputs. The first argument of the kernel must be `__global double*` for the output and the second argument must be `const int` for the length of the output vector. All additional arguments are optional. See `oclRun` for an example of a simple kernel.

Note that building a kernel can take substantial amount of time (depending on the OpenCL implementation) so it is generally a good idea to compile a kernel once and re-use it many times.

Value

Kernel object that can be used by `oclRun`.

Author(s)

Simon Urbanek

See Also

oclDevices, oclRun

Index