Package 'OpenCL'

March 9, 2020

Version 0.2-1

Title Interface allowing R to use OpenCL

Author Simon Urbanek <Simon.Urbanek@r-project.org>, Aaron Puchert <aaronpuchert@alice-dsl.net>

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

Depends R (>= 2.0.0)

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

License BSD_3_clause + file LICENSE

SystemRequirements OpenCL library

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

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-03-09 08:34:17 UTC

R topics documented:

clBuffer *Create and handle OpenCL buffers*

Description

OpenCL buffers are just like numeric or integer vectors that reside on the GPU and can directly be accessed by kernels. Both non-scalar arguments to `oclRun` and its return type are OpenCL buffers.

Just like vectors in R, OpenCL buffers have a mode, which is (as of now) one of "double" or "numeric" (corresponds to `double` in OpenCL C), "single" (`float`) or "integer" (`int`).

The constructor `clBuffer` takes a context as created by `oclContext`, a length and a mode argument.

The conversion function `as.clBuffer` creates an OpenCL buffer of the same length and mode as the argument and copies the data. Conversely, `as.double` (= `as.numeric`) and `as.integer` read a buffer and coerce the result as vector the appropriate mode.

With `is.clBuffer` one can check if an object is an OpenCL buffer.

The methods `length.clBuffer` and `print.clBuffer` retrieve the length and print the contents, respectively.

Basic access to the data is available via `[...]`. As of now, only an empty selection is supported (which selects all elements), i.e. you can only select `buf[]`.

Usage

```
clBuffer(context, length, mode = c("numeric", "single", "double", "integer"))
as.clBuffer(vector, context)
is.clBuffer(any)
## S3 method for class 'clBuffer'
as.double(x, ...)
## S3 method for class 'clBuffer'
as.integer(x, ...)
## S3 method for class 'clBuffer'
print(x, ...)
## S3 method for class 'clBuffer'
length(x)
## S3 method for class 'clBuffer'
x[indices]
## S3 replacement method for class 'clBuffer'
x[indices] <- value
```

Arguments

`context`	OpenCL context as created by `oclContext`
`length`	Length of the required buffer
`mode`	Mode of the buffer, can be one of "numeric", "clFloat", "integer"
`vector`	Numeric or integer vector or `clFloat` object
`any`	Arbitrary object

x	OpenCL buffer object (clBuffer)
indices	Indices to access the buffer, must be omitted (as of now)
value	New values
...	Arguments passed to subsequent methods

Author(s)

Aaron Puchert

See Also

oclContext, oclRun

Examples

```
library(OpenCL)
ctx<-oclContext()

buf<-clBuffer(ctx, 16, "numeric")
# Do not write buf<-..., as this replaces buf with a vector.
buf[]<-sqrt(1:16)
buf

intbuf<-as.clBuffer(1:16, ctx)
print(intbuf)

length(buf)
as.numeric(buf)
buf[]

## clBuffer is the required argument and return type of oclRun.
## See oclRun() examples.
```

---

oclContext *Create an OpenCL context for a given device.*

Description

OpenCL contexts host kernels and buffers for the device they are hosted on. They also have an attached command queue, which allows out-of-order execution of all operations. Once you have a context, you can create a kernel in the context with oclSimpleKernel.

Usage

```
oclContext(device = "default", precision = c("best", "single", "double"))
```

Arguments

device	Device object as obtained from oclDevices or a type as in oclDevices. In this case, a suitable device of the given type will be selected automatically.
precision	Default precision of the context. This is the precision that will be chosen by default for numeric buffers and kernels with numeric output mode.

Value

An OpenCL context.

Author(s)

Aaron Puchert

See Also

oclDevices, oclSimpleKernel

Examples

```
library(OpenCL)
platform <- oclPlatforms()[[1]]
device <- oclDevices(platform)[[1]]
ctx <- oclContext(device)
print(ctx)
```

---

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 = "all")
```

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(p[[1]], "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.

Usage

```
oclRun(kernel, size, ..., dim = size)
```

Arguments

kernel	Kernel object as obtained from oclSimpleKernel
size	Length of the output vector
...	Additional arguments passed to the kernel
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.

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 unsigned int second argument denoting the result length. All other arguments are assumed to be read-only and will be filled according to the ... values. These can either be OpenCL buffers as generated by clBuffer for pointer arguments, or scalar values (vectors of length one) for scalar arguments. Only integer (int), and numeric (double or float) scalars and OpenCL buffers are supported as kernel arguments. The caller is responsible for matching the argument types according to the kernel in a way similar to .C and .Call.

Value

The resulting buffer of length size.

Author(s)

Simon Urbanek, Aaron Puchert

See Also

oclSimpleKernel, clBuffer

Examples

```
library(OpenCL)
ctx = oclContext(precision="single")

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

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

## does the device support double-precision?
if (any("cl_khr_fp64" == oclInfo(attributes(ctx)$device)$exts)) {
  k.dnorm <- oclSimpleKernel(ctx, "dnorm", code, "double")
  f <- function(x, mu=0, sigma=1) {
    buf <- clBuffer(ctx, length(x), "double")
    buf[] <- x
    as.numeric(oclRun(k.dnorm, length(x), buf, 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

Creates a kernel object by compiling the supplied code. The kernel can then be used in `oclRun`.

Usage

```
oclSimpleKernel(context, name, code,
    output.mode = c("numeric", "single", "double", "integer"))
```

Arguments

`context`	Context (as created by `oclContext`) to compile the kernel in.
`name`	Name of the kernel function - must match the name used in the supplied code.
`code`	Character vector containing the code. The code will be concatenated (as-is, no newlines are added!) by the engine.
`output.mode`	Mode of the output argument of the kernel, as in `clBuffer`. This can be one of "single", "double", "integer", or "numeric". The default value "numeric" maps to the default precision of the context. The kernel code may use a type `numeric` that is typedef'd to the given precision, i.e. either `float` or `double`. The OpenCL extension `cl_khr_fp64` will be enabled automatically in the second case, so you don't have to add the pragma yourself.

Details

`oclSimpleKernel` builds the program specified by `code` and 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*` or `__global float*` for the output and the second argument must be `const unsigned int` for the length of the output vector. Additional numeric scalar arguments are assumed to have the same mode as the output, i.e. if the output shall have "double" precision, then numeric scalar arguments are assumed to be double values, similarly for single-precision. 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, Aaron Puchert

See Also

`oclContext`, `oclRun`

Index