Custom OpenCL Kernels

Dr. Charles Determan Jr. PhD*

October 18, 2017

1 Introduction

At the heart of portable GPU computing is the use of OpenCL kernel files. Essentially, these files contain routines/functions that are compiled for use on different devices (e.g. CPU, GPU, FPGA, etc.). One example is the SAXPY function

```
"__kernel void SAXPY(__global float* x, __global float* y, float a)
{
    const int i = get_global_id(0);

    y [i] += a * x [i];
}
"
```

The exact definitions and specifications of OpenCL kernels is beyond the scope of this vignette and the interested user is recommended to consult additional resource.

Now, to compile a function to use the OpenCL kernel requires the definition of OpenCL contexts, buffers, defining global/local sizes, enqueuing operations, and managing data memory copies. The intent here is to make the use of a custom OpenCL kernel as seamless as possible within the *gpuR* package.

*cdetermanjr@gmail.com

2 Demo

In order to create the dynamic functions to leverage the OpenCL kernels, the user should be able to say what the purpose of each argument is. Taking the SAXPY example above, the user must know which arguments are device objects (i.e. OpenCL buffers) and which aren't. In this case, the first two arguments I will denote as `vclVector` objects and the third as a scalar argument. Likewise, they are denoted by the objects intent, which kernel they will be passed to (only relevant when more than one kernel function), and the corresponding argument name in the OpenCL kernel function. The setup call would look like the following.

```
cl_args <- setup_opencl(objects = c("vclVector", "vclVector", "scalar"),
                        intents = c("IN", "OUT", "IN"),
                        queues = list("SAXPY", "SAXPY", "SAXPY"),
                        kernel_maps = c("x", "y", "a"))
```

Once the argument definitions are setup, the kernel file and argument definitions can be passed to the `custom_opencl` function which also takes one additional argument. OpenCL is a typed language and thefore the user must denote the precision.

```
custom_opencl("saxpy.cl", cl_args, "float")
```

You will notice that there is no assignment set during this call. The internal function is compiled and exported to the current global environment with the name of the kernel file (prior to the extension).

The function can then be called as a normal function on the respective objects.

```
a <- rnorm(16)
b <- rnorm(16)
gpuA <- vclVector(a, type = "float")
gpuB <- vclVector(b, type = "float")
scalar <- 2

# apply custom function
# equivalent to - scalar*a + b
saxpy(gpuA, gpuB, scalar)
```