A Short Introduction to the *gpuR* Package

Dr. Charles Determan Jr. PhD*

March 7, 2016

1 Introduction

GPUs (Graphic Processing Units) were originally developed to perform graphic rendering and commonly referred to in the comuting gaming world. These devices are also able to be applied to numerical operations in parallel. Although there are a few different vendors, the two primary competitors are AMD and NVIDIA.

NVIDIA GPUs depend upon the proprietary CUDA framework whereas AMD GPUs utilize the open source OpenCL framework. The upside of OpenCL is that 'kernels' are able to used on any GPU whereas CUDA kernels can only be used on NVIDIA GPUs. It is worth noting, however, that CUDA tends to edge out OpenCL performance likely a result of the highly specific framework to the NVIDIA GPUs.

That said, programming either framework is often difficult for programmers unaccustomed to working with such a low-level interface. Creating bindings for high-level programming languages (such as R) make using GPUs much more accessible to a broader audience.

Several R packages have been developed including *gputools*, *cudaBayesreg*, *HiPLARM*, *HiPLARb*, and *gmatrix*. However, all of the packages depend upon a CUDA backend and therefore restrict the using to only using NVIDIA GPUs. The novelty of this package is the use of the ViennaCL library (http://viennacl.sourceforge.net/) which has been coveniently been repackaged in the *RViennaCL* package to be used in other R packages. This allows the user to leverage the auto-tuned OpenCL kernels of the ViennaCL library on any GPU. It also allows for a CUDA backend for those who in fact have a NVIDIA GPU for improved performance, which will be provided in a companion *gpuRcuda* package.

Of the aforementioned packages, most contain a very limited set of functions available to the R user within the packages. The most extensive being the *gmatrix* package which contains most linear algebra operations. All of the packages (with the exception of *gmatrix*) don't store the data on the GPU. As such, there is the overhead of transferring data back and forth between the device and host. Similar to *gmatrix*, this package utilizes S4 classes to store an external pointer to the data on the GPU which mirror the base `matrix` and `vector` classes. However, given the interactive nature of R programming and

*cdetermanjr@gmail.com

the limited RAM available on GPU's this package provides intermediate classes that remove the object from GPU RAM to allow objects to be stored on the CPU but still utilize the GPU as needed.

2 Install

Install *gpuR* using

```
# Stable version
install.packages("gpuR")
```

If a user is interested in using the current development version they should install directly from my github account.

```
# Dev version
devtools::install_github("cdeterman/gpuR", ref = "develop")

# Note this may require install of the RViennaCL from my github
# if updates have been made
#devtools::install_github("cdeterman/RViennaCL")
```

3 Getting Help

If you find an error/bug while using this package I would like to know about it. Likewise, if there is a method you think would be useful for others feel free to request it's addition (or provide a contribution yourself to be added). Please submit the all such reports and requests on my github Issues.

4 Basic Use with *gpuMatrix*

gpuR has most basic linear algebra operations. The user simply needs to create a *gpuMatrix* object and the GPU methods will be used. Here is a minimal example demonstrating typical matrix multiplication.

```
library("gpuR")

# verify you have valid GPUs
detectGPUs()

# create gpuMatrix and multiply
set.seed(123)
gpuA <- gpuMatrix(rnorm(16), nrow=4, ncol=4)
gpuB <- gpuA %*% gpuA
```

Most linear algebra methods have been created to be executed for the *gpuMatrix* and *gpuVector* objects. These methods include basic Aithmetic functions `%*%`, `+`, `-`, `*`, `/`, `t`, `crossprod`, `tcrossprod`, `colMeans`, `colSums`, `rowMean`, and `rowSums`. Math functions include `sin`, `asin`, `sinh`, `cos`, `acos`, `cosh`, `tan`, `atan`, `tanh`, `exp`, `log`, `log10`, `exp`, `abs`, `max`, and `min`. Additional operations include some linear algebra routines such as `cov` (Pearson Covariance) and `eigen`. A few 'distance' routines have also been added with the `dist` and `distance` (for pairwise) functions. These currently include 'Euclidean' and 'SqEuclidean' methods.

The objects may also be created specifying data type including int, float, and double. Float type was included to provide a smaller memory footprint and also increased throughput if the increased accuracy of double is not required.

Both the *gpuMatrix* and *vclMatrix* objects return a pointer to the data. Given that working with GPU's implies that you are working with larger datasets, this prevents R from making unneccessary copies. However, this does require the user to exercise caution as any change made to a 'copy' (e.g. `gpuB <- gpuA`) will result in changes to the original object and all others pointing to it as well.

5 *vclMatrix* Class

The *vclMatrix* class was created to allow the user to put data directly on the GPU once and not need to continually push data back and forth between the host and device. Therefore, if multiple processes are to be applied to a given matrix, there will be significant savings by using *vclMatrix* objects. It is important to remember though, different GPU's have different amounts of RAM. The interactive nature of R often has many objects existing simultaenously where you may exceed your GPU's RAM. As such, the *gpuMatrix* class is provided.

6 'block' & 'slice' object

Sometimes it is useful to only refer to a subset of a matrix and apply some operations. This has been accomplished by providing the `block` and `slice` methods for matrix and vector classes respectively. The resulting object is a child class of the parent object (e.g. `gpuMatrixBlock` from `gpuMatrix`). As such, nearly all methods defined for the parent objects are also defined for the child class.

```
# create gpuMatrix
set.seed(123)
gpuA <- gpuMatrix(rnorm(16), nrow=4, ncol=4)

# create block omitting the 1st row
gpuB <- block(gpuA,
              rowStart = 2L, rowEnd = 4L,
              colStart = 1L, colEnd = 4L)
```

The resulting object is a reference to the parent objects elements to avoid memory duplication. As such, any changes to the 'block' or 'slice' object will alter the parent elements. If the user wishes to make these objects distinct, they should use the `deepcopy` function.

7 GPU Memory Management

This package has been written to utilize the capabilities provided by *Rcpp*. As such, the pointers referenced in the objects herein are handled by the 'XPtr' object. Once the object is deleted in the R session **and the garbage collector has run** the GPU memory will be freed.

It is important for the user to respect the rate at which the R garbage collector is called. It has been shown that when some of these functions are called repeatedly in a loop the GPU processing goes too rapidly for the garbage collector to be called to release the memory resulting in the GPU becoming filled. Explicitly calling `gc()` within loops following `rm` of the temporary object should alleviate problems such as these.

8 Contributing Back

Given the broad use for this package (i.e. many different GPUs) it would be useful if you could report success cases. A curated list of tested platforms would be valuable for future users. Please see report your Operating System, OpenCL Platform (can be seen with `platformInfo`), and your GPU(s) also on my gitter Tested GPUs page.

9 Crowd-source Testing

As with the above statements, there a many architecture possibilities that this package 'should' run on. That said, it is impossible for me to have access to all the different hardware. As such, I am hoping that users can continually test this package as well.

1. Test current release The simplest approach is to test the current CRAN version. Simply download the tar file from CRAN archive for *gpuR*. Then you can just run:

```
R CMD check gpuR_version.
```

NOTE - you will likely receive 3 NOTES (checking repository dependencies, package size, and pandoc). You also will likely see a WARNING referring to the Date field being old. YOu can safely ignore these results. Any error should be reported to my github Issues. Please include the OS, OpenCL Platform, & GPU(s). Ideally you can at least report which test is failing if returned by `R CMD check`.

2. Test development version. This requires a little bit more effort on the users part. First, you need to to clone my development branch from my git repository.

```
git clone -b develop https://github.com/cdeterman/gpuR.git
```

Then you can enter the gpuR directory and run `devtools::test()`.

Ideally, users will try to at least diagnose at least which function the error is originating (or sequence of functions). It is important that if the user cannot solve the bug that it is reproducible so that I (or others) are able to try to approach the problem.