R2GPU: A Very Simple R Interface for General Purpose Computing on Nvidia GPUs

Julio Olaya, Xueyuan Cao, Stan Pounds

Department of Biostatistics, St. Jude Children's Research Hospital, Memphis, TN, USA

St. Jude Children's Research Hospital
ALSAC • Danny Thomas, Founder
Finding cures. Saving children.

Finding cures. Saving children.

Abstract

We have developed the *R2GPU* suite of R packages (*R2GPUbase*, *R2GPUmath*, *R2GPUprob*) to provide a very simple R interface to general purpose computing on NVIDIA graphical processing units (GPUs). The *R2GPU* packages define the infrastructure to initiate interactions between an R session and a GPU device, transmit data between an R session and a GPU device, and transmit instructions from R to a GPU device. This infrastructure provides two distinct practical advantages to the R programmer:

(1) most functions for the GPU are syntactically identical to standard R, thereby reducing the burden of learning a new set of R commands; and

(2) the results of a GPU calculation may remain on the GPU for further processing by the GPU, thereby greatly reducing the overhead of using the GPU to maximize the speed-up for real applications.

In this way, the *R2GPU* packages provide R programmers the ability to easily expand existing packages to perform computationally intensive tasks on the GPU.

Background

We have entered a big data era in science, business, and government. Big data are common in many applications . Computing time for data processing and analysis can be extensive. For example, biomedical genomics researchers often collect millions of data values for each of many patients. Computational analysis of this data can take hours, days, or weeks.

Graphical processing units (GPUs) are an inexpensive platform for parallel computing. In principle, GPUs could be used for many big data applications. However, programming for GPUs is very technical and difficult. A simple interface to GPU computing in a popular language such as R would make GPU computing power more readily available to more software developers.

Objectives

1. **Use standard R syntax for GPU computing.**
2. **Maximize speed-up for real applications.**

Methods

1. **Define an R object class `CP` that points from the CPU to data on the GPU.**
2. **Define `toGPU(x)` as a function that transmits the data in `x` from the CPU to the GPU and returns a `CP` object.**
3. **Use a standard R interface to define methods for `CP` objects.**
4. **Define `toGPU(x)` as a function that operates on the `CP` object class to bring data back to the CPU from the GPU.**

Illustrative Example

```
# Rank and z-transform each row or column of a matrix
library(R2GPUmath)
prep.mtx=function(X,      # a numeric matrix
                  dm,     # what to operate on (1=rows, 2=colums)
                  gpu=F) # indicates whether to use a GPU

{                                 # begin function
  if (dm==2) X=t(X)                 # transpose to operate on columns
  n=ncol(X)                         # get number of columns

  r=NULL                            # initialize r
  if (!gpu) r=apply(X,1,rank)     # rank each row on CPU
  else                              # rank each row on GPU
  {
    x=toGPU(X)                        # move data to GPU
    x=t(x)                            # transpose for GPU rank routine
    r=rank(x)                         # use GPU rank routine
  }
  r=t(r)                            # transpose the matrix of ranks
  m=rowMeans(r)                     # find mean of ranks for each row
  s=rowSums((r-m)^2)/(n-1)          # find MSE of ranks for each row
  s=sqrt(s)                         # find RMSE of ranks for each row
  z=(r-m)/s                         # z-transform ranks for each row
  if (dm==2) z=t(z)                 # transpose back if needed
  return(z)                         # return result (to GPU if on GPU)
}                                 # end function

m=100000; n=1000; Y=matrix(rnorm(m*n),m,n);          # example matrix
cpu.prep=prep.mtx(Y,1,F); gpu.prep=prep.mtx(Y,1,T); # run both ways
```

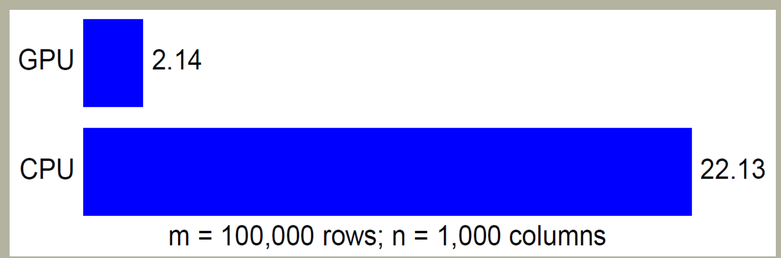

Figure 1. Computing time in seconds for the illustrative example shown above. The GPU achieved a 10.3 fold speed-up.

Conclusions and Discussion

1. **The R2GPU packages empower R programmers to achieve massive speed-ups (10-100 fold) on a GPU while leaving most of their CPU code unchanged.**
2. **GPUs are a much more affordable parallel computing resource than traditional clusters. A K40 GPU costs roughly $3,500.**
3. **Future work will make the GPU available for more standard R functions.**

Example Application

We used the *R2GPU* packages to develop a GPU implementation of the projection onto the most interesting statistical evidence (PROMISE; Pounds et al 2009; PMID 9528086) statistical method. PROMISE is an integrated data analysis method that identifies genes that show the most significant pattern of biologically meaningful associations with multiple pharmacological and clinical endpoint variables. PROMISE is a computationally intense method based on permutation testing. We developed an implementation that could use the CPU or GPU. The GPU was 125 times faster than the CPU. The vast majority of the code lines were identical for the GPU and CPU implementations (Figure 3).

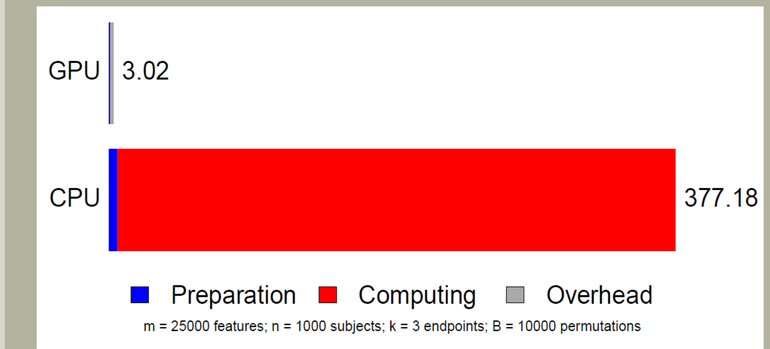

Figure 2. Computing time in seconds for a PROMISE analysis. The total completion time for the CPU is 377.18 seconds while that for the GPU is only 3.02 seconds (a 125-fold speed-up). The computing time for the CPU is 371.72 seconds while that for the GPU is only 0.01 seconds (a 33,792 fold speed-up).

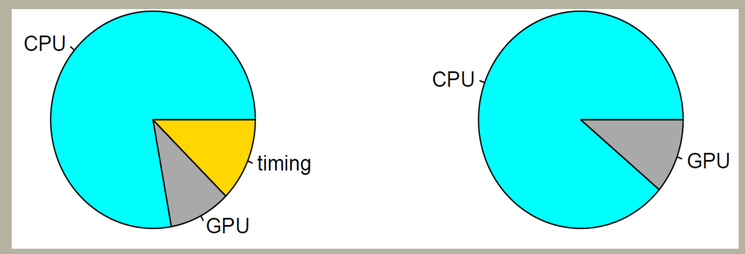

Figure 3. Percentage of code that is identical to the CPU implementation (light blue), used for benchmark timing (gold) and unique to the GPU implementation (gray). The left panel shows the overall percentages; the right panel excludes the code for timing.

Technical Details

Hardware :
Dell cluster with two nodes.
Each node has 12 CPU cores (2.4 GHz Intel ® Xeon).
Eight NVIDIA K40m Tesla GPUs.
Each K40m has 11.25 GB Global Memory.

Software:
R Linux Version 3.1.2
C++
CUDA v6.5 runtime APIs, computer capabilities of 3.5.

Acknowledgements: St. Jude donors, ALSAC, the PBTC Foundation, and Dr. James Boyett