Accelerating ggplot2 Based Projection On R-map Using NVIDIA GPU

Supervisor: **Dr. Jia Uddin**

Md. Zahidul Islam 12201051

Md. Tausif Elahi 12201036

Department of Computer Science and Engineering,

BRAC University.

Submitted on: 17th August 2016

DECLARATION

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole or in part, has been previously submitted for any degree.

Signature of Supervisor

Dr. Jia Uddin

Signature of Author

Md. Zahidul Islam

Md. Tausif Elahi

ACKNOWLEDGEMENTS

All thanks to Almighty ALLAH, the creator and the owner of this universe, the most merciful, beneficent and the most gracious, who provided us guidance, strength and abilities to complete this research.

We are especially thankful to Dr. Jia Uddin, our thesis supervisor, for his help, guidance and support in completion of my project. We also thankful to the BRAC University Faculty Staffs of the Computer and Communication Engineering, who have been a light of guidance for us in the whole study period at BRAC University, particularly in building our base in education and enhancing our knowledge.

Finally, we would like to express our sincere gratefulness to our beloved parents, brothers and sisters for their love and care. We are grateful to all of our friends who helped us directly or indirectly to complete our thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

With an increasing amount of user and data demands for fast data processing, the optimization of database operations continues to be a challenging work. A common optimization technique is to leverage parallel hardware architectures. With the introduction of general-purpose GPU computing, massively parallel hardware has become available within commodity hardware. To efficiently exploit this technology, we introduce the method of speculative query processing. Moreover, as the dataset grows increasingly larger, multiple-thread spatial query sometimes cannot meet the performance requirement. The concept of GPU-accelerated parallel computing turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU.

R is a free software environment for graphics and statistical computing that provides a programming language and built-in libraries of mathematics operations for data analysis, statistics, machine learning and much more. R programs tend to process large amounts of data, and often have significant independent data and task parallelism. Therefore, R applications stand to benefit from GPU acceleration. This way, R users can benefit from R’s high-level, user-friendly interface while achieving high performance. Thus focusing on accelerating R computations using CUDA libraries by calling our own parallel algorithms written in CUDA from R and profiling GPU-accelerated R applications using the CUDA Profiler.

CHAPTER01

INTRODUCTION

1.1 Motivation

In this thesis, our work mainly focuses on enhancing graphical data processing rate by accelerating R computations using CUDA libraries. We are going to use the idea of parallel computing, which is built on the working methodology of General-purpose computing on graphics processing units. R has a fantastic mechanism for creating data structures. R's data structures include matrices, vectors, arrays, data frames and lists [7]. R's extensible object system includes objects for regression models, time-series and geo-spatial coordinates. The use of multiple graphics cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip [3]. In this thesis, we will introduce the computation model of R with GPU acceleration, focusing on accelerating R computations using CUDA libraries by calling our own parallel algorithms written in CUDA from R; and profiling GPU-accelerated R applications using the CUDA Profiler.

R can handle graphical data and process and map those data quite efficiently when compared to other languages. But if the dataset is large enough the computation speed of R may become quite a bit slow. Our main focus is to amplify the performance of R while handling bulky graphical data, thus we merged with the concept of parallel computing. The platform for operation is CUDA; CUDA is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

1.2 Contribution Summary

- The first approach is to use existing GPU-accelerated R packages listed under High Performance and Parallel Computing with R using packages from CRAN (the

Comprehensive R Archive Network).. Examples include gputools and cudaBayesreg. These packages are very easy to install and use.

- The second approach is to use the GPU through CUDA directly. While the CUDA ecosystem provides many ways to accelerate applications, R cannot directly call CUDA libraries or launch CUDA kernel functions. To solve this problem, we need to build an interface to bridge R and CUDA [17]. For productivity, we should also encapsulate these interface functions in the R environment, so that the technical changes between the CPU and GPU are transparent to the R user.

1.3 Thesis Orientation

- Chapter 02 includes the necessary background information regarding the proposed approaches of parallel computing using CUDA and R.
- In chapter 03 we will review some further techniques and methods for optimizing CUDA and amplification of data processing.
- Chapter 04 is based on how map projection is done by accelerating ggplot2 based projection on R
- Chapter 05we accelerates R Using CUDA Libraries, analyzes performance and demonstrates the experimental results and comparison.
- Chapter 06 states the resources used and problems faced during our thesis work.
- Chapter 07 concludes the thesis and overview on future research directions.

CHAPTER 02
BACKGROUND INFORMATION

2.1 GPU Computing: The Revolution

With CUDA, we can send C, C++ and FORTRAN code straight to GPU, so no assembly language required. Developers at companies such as Adobe, Math Works, ANSYS, Autodesk, and Wolfram Research are waking that sleeping giant – the GPU -- to do general-purpose scientific and engineering computing across a range of platforms.

Using high-level languages, GPU-accelerated applications run the sequential part of their workload on the CPU – which is optimized for single-threaded performance – while accelerating parallel processing on the GPU [1]. This is called "GPU computing."

GPU computing is possible because today's GPU does much more than render graphics: It sizzles with a teraflop of floating point performance and crunches application tasks designed for anything from finance to medicine.

CUDA is widely deployed through thousands of applications and published research papers and supported by an installed base of over 375 million CUDA-enabled GPUs in notebooks, workstations, compute clusters and supercomputers [4].

2.2 Tools and Training

Today, the CUDA ecosystem is growing rapidly as more and more companies provide world-class tools, services and solutions. If you want to write your own code, the easiest way to harness the performance of GPUs is with the CUDA Toolkit, which provides a comprehensive development environment for C and C++ developers [4].

The CUDA Toolkit includes a compiler, math libraries and tools for debugging and optimizing the performance of your applications. You'll also find code samples, programming guides, user manuals, API references and other documentation to help you get started.

2.3 CUDA Programming Model

A kernel is executed by a grid of thread blocks. A thread block is a batch of threads that can cooperate with each other by sharing data through shared memory and by synchronizing their execution. Threads from different blocks cannot cooperate. Kernels are launched in grids, one kernel executes at a time. A block executes on one Streaming Multiprocessor(SM) and does not migrate. Several blocks can reside concurrently on one SM. Threads and blocks have IDs so each thread can decide what data to work on so it simplifies memory addressing when processing multi-dimensional data [2].

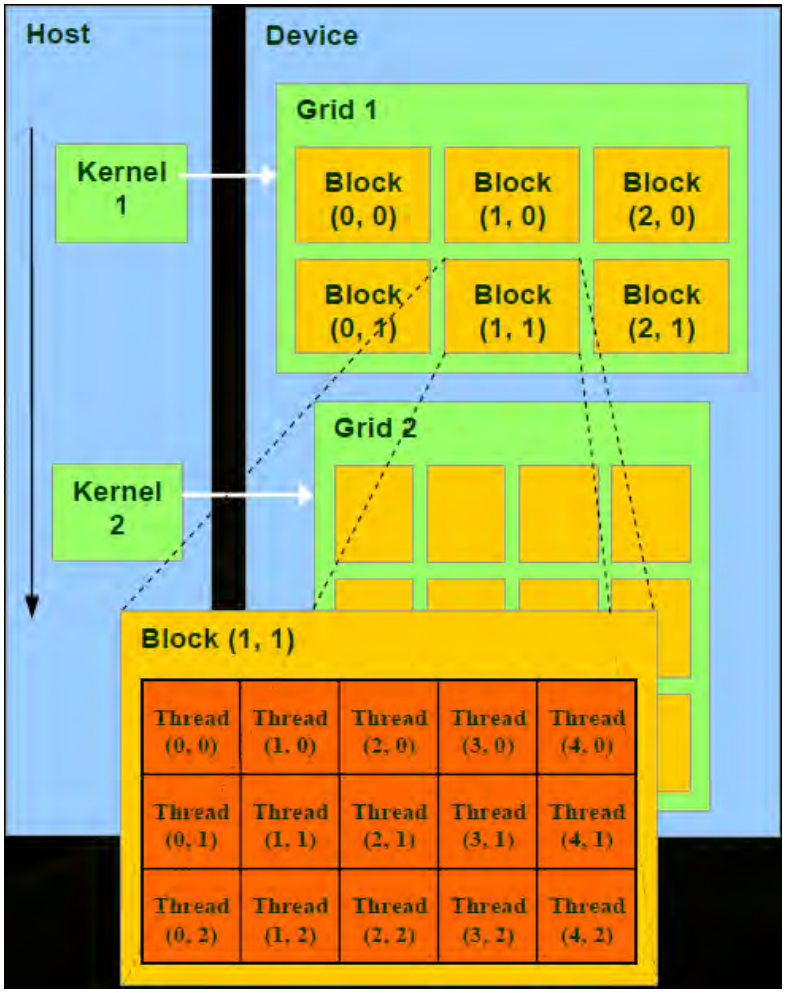

Figure 2.1 : CUDA Programming Model [2].

2.4 How GPUs Accelerate Applications

GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. The concept of GPU-accelerated parallel computing turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations [16]. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU.

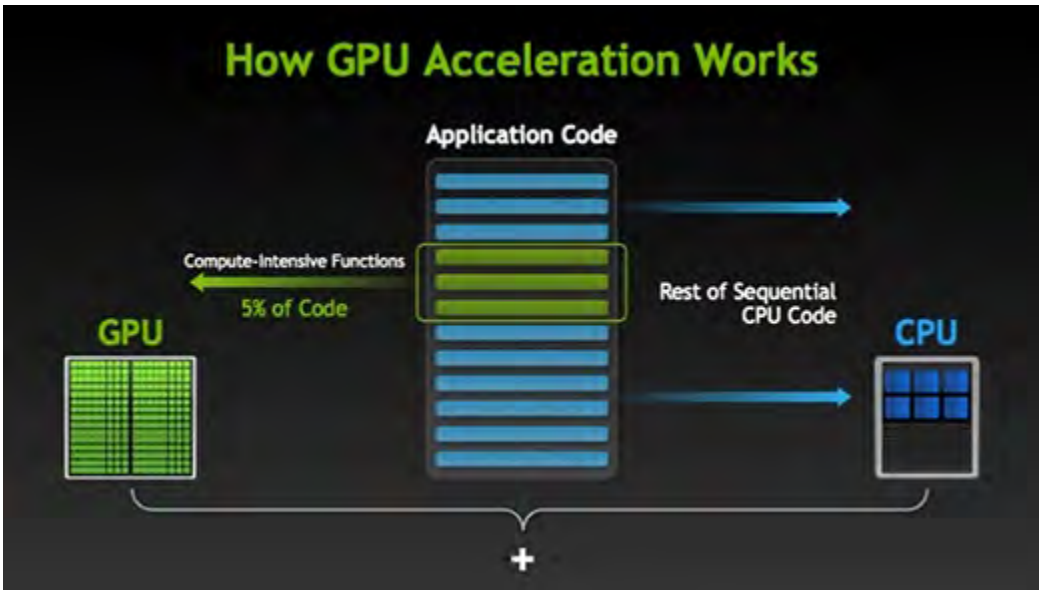

Figure 2.2 : CUDA Work Distribution [16].

2.5 R programming language

R is a programming language and software environment for graphics and statistical computing supported by the R Foundation for Statistical Computing. This R language is widely used among data miners and statisticians for developing statistical software and data analysis [5]. R is a GNU package. The source code for the R software environment is written primarily in C, FORTRAN, and R. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available. One of R's strengths

is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed [7]. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

2.5.1 The R environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The term “environment” is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software [7].

2.5.2 A Moveable Feast

Flexibility and power abound in R. For instance, it is easy to call C and C++ functionality from R. R does not insist that everything is done in its language, so we can mix tools — picking the best tool for each particular task [8]. The pieces of code that are written in the R language are always available to the user, so a minor change to the task usually requires only a minor change to the code — a change that can be carried out in a minor amount of time.

CHAPTER 03

OPTIMIZING CUDA

3.1 Matrix Transpose

The code we wish to optimize is a transpose of a matrix of single precision values that operates out-of-place, i.e. the input and output are separate arrays in memory. For simplicity of presentation, we'll consider only square matrices whose dimensions are integral multiples of 32 on a side. It consists of several kernels as well as host code to perform typical tasks such as allocation and data transfers between host and device, launches and timing of several kernels as well as validation of their results, and de-allocation of host and device memory [18].

In addition to performing several different matrix transposes, we run simple matrix copy kernels because copy performance indicates the performance that we would like the matrix transpose to achieve. For both matrix copy and transpose, the relevant performance metric is effective bandwidth, calculated in GB/s by dividing twice the size in GB of the matrix (once for loading the matrix and once for storing) by time in seconds of execution.

3.2 Simple Matrix Copy

Let's start by looking at the matrix copy kernel

```
__global__ void copy(float *odata, const float *idata)
{
int x = blockIdx.x * TILE_DIM + threadIdx.x;
int y = blockIdx.y * TILE_DIM + threadIdx.y;
int width = gridDim.x * TILE_DIM;

for (int j = 0; j < TILE_DIM; j+= BLOCK_ROWS)
odata[(y+j)*width + x] = idata[(y+j)*width + x];
}
```

Each thread copies four elements of the matrix in a loop at the end of this routine because the number of threads in a block is smaller by a factor of four (TILE_DIM/BLOCK_ROWS) than the number of elements in a tile. Note also that TILE_DIM must be used in the calculation of the matrix index y rather than BLOCK_ROWS or block Dim.y. The loop iterates over the second dimension and not the first so that contiguous threads load and store contiguous data, and all reads from idata and writes to odata are coalesced.

```
{
int x = blockIdx.x * TILE_DIM + threadIdx.x;
int y = blockIdx.y * TILE_DIM + threadIdx.y;
int width = gridDim.x * TILE_DIM;

for (int j = 0; j < TILE_DIM; j+= BLOCK_ROWS)
odata[(y+j)*width + x] = idata[(y+j)*width + x];
}
```

3.3 Naive Matrix Transpose

Our first transpose kernel looks very similar to the copy kernel. The only difference is that the indices for odata are swapped.

```
__global__ void transposeNaive(float*odata,constfloat*idata)
{
int x = blockIdx.x * TILE_DIM + threadIdx.x;
int y = blockIdx.y * TILE_DIM + threadIdx.y;
int width = gridDim.x * TILE_DIM;

for(int j =0; j < TILE_DIM; j+= BLOCK_ROWS)
odata[x*width +(y+j)]= idata[(y+j)*width + x];
}
```

The transposeNaive reads from idata are coalesced as in the copy kernel, but for our 1024×1024 test matrix the writes to odata have a stride of 1024 elements or 4096 bytes between contiguous thread [19]. This puts us well into the asymptote of the stride memory access plot from our global memory coalescing post, and we expect the performance of this kernel to suffer accordingly. The results of the copy and transposeNaive kernels bear this out. The transposeNaive kernel achieves only a fraction of the effective bandwidth of the copy kernel. Because this kernel does very little other than copying, we would like to get closer to copy throughput.

3.4 Coalesced Transpose Via Shared Memory

The remedy for the poor transpose performance is to use shared memory to avoid the large strides through global memory. The Figure 3.1depicts how shared memory is used in the transpose.

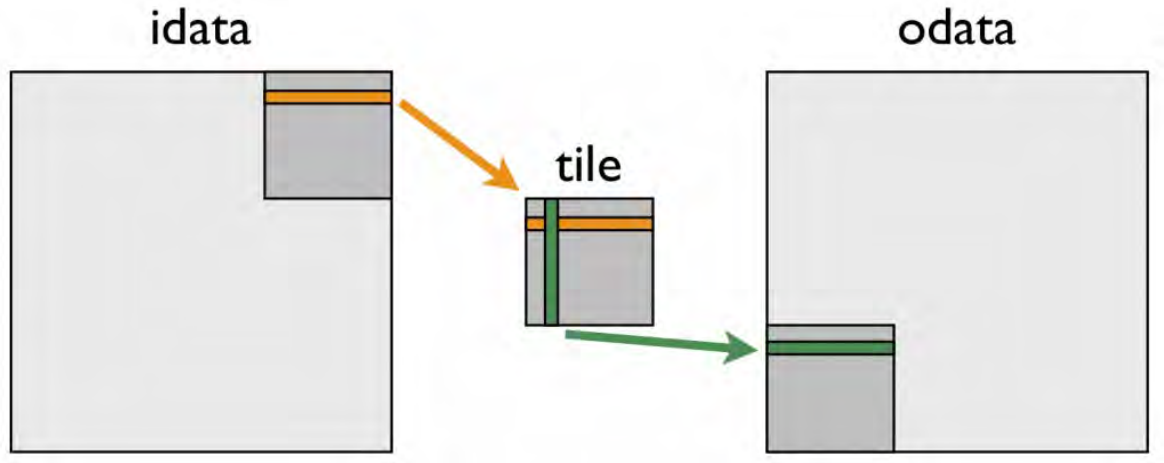

Figure 3.1: The following kernel performing the “tiled” transpose [18].

```
__global__ void transposeCoalesced(float *odata, const float
*idata)
{
__shared__ float tile[TILE_DIM][TILE_DIM];

int x = blockIdx.x * TILE_DIM + threadIdx.x;
int y = blockIdx.y * TILE_DIM + threadIdx.y;
int width = gridDim.x * TILE_DIM;

for (int j = 0; j < TILE_DIM; j += BLOCK_ROWS)
tile[threadIdx.y+j][threadIdx.x] = idata[(y+j)*width + x];

__syncthreads();

x = blockIdx.y * TILE_DIM + threadIdx.x;  // transpose block
offset
y = blockIdx.x * TILE_DIM + threadIdx.y;

for (int j = 0; j < TILE_DIM; j += BLOCK_ROWS)
odata[(y+j)*width + x] = tile[threadIdx.x][threadIdx.y + j];
}
```

In the first do loop, a warp of threads reads contiguous data from idata into rows of the shared memory tile [18]. After recalculating the array indices, a column of the shared memory tile is written to contiguous addresses in odata. Because threads write different data to odata than they read from idata, we must use a block-wise barrier synchronization __syncthreads(). This approach gives us a nice speed up, as shown in this updated effective bandwidth table

The transposeCoalesced results are an improvement over the transposeNaive case, but they are still far from the performance of the copy kernel. We might guess that the cause of the performance gap is the overhead associated with using shared memory and the required synchronization barrier __syncthreads(). We can easily test this using the following copy kernel that uses shared memory [18].

```
__global__ void copySharedMem(float *odata, const float *idata)
{
__shared__ float tile[TILE_DIM * TILE_DIM];

int x = blockIdx.x * TILE_DIM + threadIdx.x;
int y = blockIdx.y * TILE_DIM + threadIdx.y;
int width = gridDim.x * TILE_DIM;

for (int j = 0; j < TILE_DIM; j += BLOCK_ROWS)
tile[(threadIdx.y+j)*TILE_DIM + threadIdx.x] = idata[(y+j)*width + x];

__syncthreads();

for (int j = 0; j < TILE_DIM; j += BLOCK_ROWS)
odata[(y+j)*width + x] = tile[(threadIdx.y+j)*TILE_DIM + threadIdx.x];
}
```

Note that the synchthreads() call is technically not needed in this case, because the operations for an element are performed by the same thread, but we include it here to mimic the transpose behavior. The second line of the table below shows that the problem is not the use of shared memory or the barrier synchronization.

CHAPTER 04

ACCELERATING ggplot2 BASED PROJECTION ON R

4.1 The GPU-Accelerated R Software Stack

R is free and open source software, and the users of R continuously upgrade packages and plugging from the R's own library. There Is always arduous knowledge from R, but there is another software ArcGIS learning curve is just more complex than R. If we measure side by side of measuring ArcGIS and R, then I have to tell this R language are more flexible and simplest command on R. because I have a bitter experience on ArcGIS. We learn barely the glazed the surface of the power of that software. On R we can write a good contained annotated data script and then than after while we just assume the mapping on R is easy to understand with various data set. By regarding of thesis, mapping on R is quite simple and insightful since many functions are more similar commands as plotting to another type of diagram. On our thesis we improvise the graphical part step by step on R [9]. At first we make on simple mapping then after that we use some core library from R and then some initial step for improvising the mapping. so far, we can say that R is not the easiest way of doing maps, but it is convenient and it allows the user the full control of any type of mapping that how the map actually is [9]. There are tons of different types of creating various maps by using R. In our research we are using R version 3.3.1 and it's a studio of R.

On this part of thesis we are going to make a simple R mapping and after that using some tools from library we are using Tmap library and ggplot2 library from the R library for the improvising the graphical representation of mapping [10].

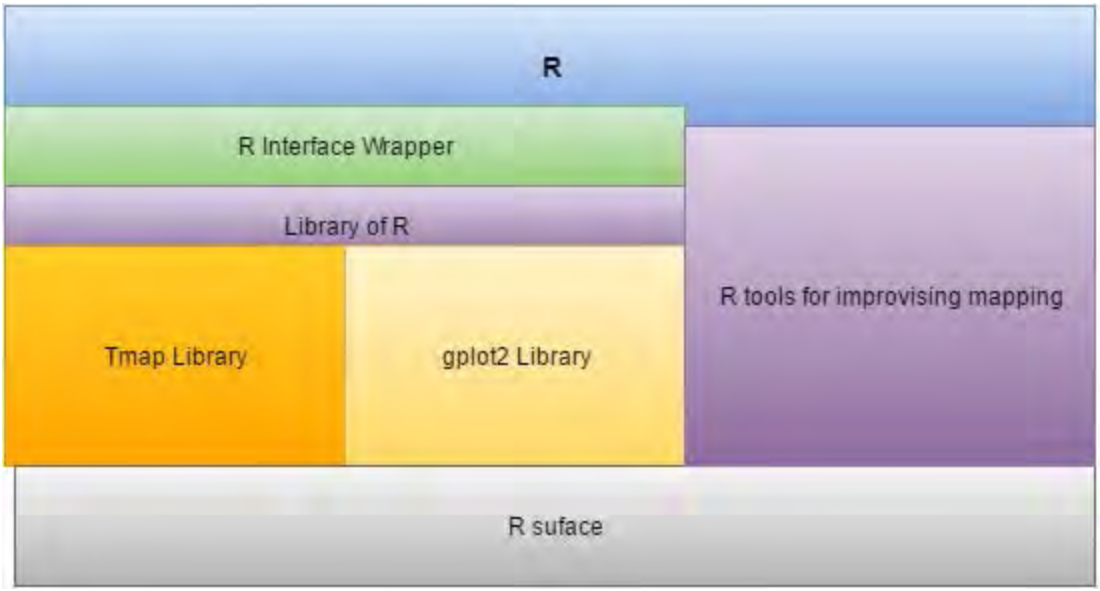

Figure 4.1: The R + GPU software stack.

4.2 Simple Mapping

To start, we will make a blank map none of our any data information. By plotting the mapping we need to packages “maps” this library contains the function we will use this command and another one is map data which function has some basic world map data query. Then with one line of code we create a simple map. Suppose the map is for Canada

Figure 4.2: Country mapping using R.

```
library(maps)

library(mapdata)

map("worldHires","Canada”, xlim=c(-141,-53), ylim=c(40,85),
col="gray90”, fill=TRUE)
```

The function “map()” which is working here. We tell it that the plot the map of Canada from the database of the library command “worldHires”. This is the similar like other plotting or R, we can be determined the range of the map what we want to see, and we can also measure from the axis which is works in the map x-axis longitude and y-axis latitude, it is called GPS coordinates in terms of lat. After that we specify the color what we want and we choose here the color of gray. So this map easy and here we can extended our map by changing the xlim and ylim command. And this command works as zoom part of Canada mapping.

4.3 Plotting GPS data and shapefiles

Previous map we are not add any data of our own. On this part of implementation we are adding our own data to a map. Most of the time the map representation is very simple plotting. We need to .csv file of all your data which consisting the minimum two columns: the one is latitude and one for the longitude. The GPS point transfer to decimal degrees because R language can’t read the degree minute and second.

Our data for the species range is contained which is shapefile. This shapefile contains a layer of dataset that can be form of polygons, lines etc. even A lot of GIS data is available online by searching some site mostly on Github [11]. To read our data from shepfile, we will use the command of “maptools” functions, and it will depending our dataset. Such as polygon data use ‘readshape Poly’, point data uses ‘readshapepoints’ [11]. After reading the command from shapefile, to our map on the function plot()’.we use the point based function on our two column so longitude and latitude will be added

```
library(maps)

library(mapdata)

library(maptools)  #for shapefiles

library(scales)  #for transparency

pcontorta <- readShapePoly("pinucont.shp")   #layer of data for
species range

samps <- read.csv("FieldSamples.csv")   #my data for sampling sites,
contains a column of "lat" and a column of "lon" with GPS points in
decimal degrees

map("worldHires","Canada", xlim=c(-140,-110),ylim=c(48,64),
col="gray90", fill=TRUE)  #plot the region of Canada I want

map("worldHires","usa", xlim=c(-140,-110),ylim=c(48,64), col="gray95",
fill=TRUE, add=TRUE)  #add the adjacent parts of the US; can't forget
my homeland

plot(pcontorta, add=TRUE, xlim=c(-140,-110),ylim=c(48,64),
col=alpha("darkgreen", 0.6), border=FALSE)  #plot the species range

points(samps$lon, samps$lat, pch=19, col="red", cex=0.5)  #plot my
sample sites
```

The details from the code that how it process, when we creating the map, everything goes under in the order from the code. We plotter the parts of Canada. And it extends from the first mapping. This is where the ‘Scale’ library uses. It provides the color(transparent) [9]. On this code we use the command “alpha(‘darkgreen’,’0.6’)”. This command indicates that the color has to be 40% transparent and the rest of the part should opaque. On this code if you notice that we include every line “add=true” because the data are being added to the existing plot, and there will be no map. This is so important because data are plotted on the layer and this thing should be cover but plotting the layer down on the line. At last, we plotted on the GPS point. We add the “boc()”, adding text with “text()” and also clarifying the GPS point as the x-axis and y-axis by adding the command “map.scale()” [9].

Figure 4.3: Projection map using mapscale method.

4.4 Projected Maps

The last map which is using our own data which indicates the map of Canada and the side of America, so it was a large scale map. And now we are implementing the more implementation of projectile mapping on R. A globe is the most accurate representation of the Earth's surface. When we can't determine on the globe therefore we will use maps projections. The last map was unprotected so we are using right now the first map we are doing to be projected. If we consider at the top that there was a island then it appears to be much more smaller than it actually is. And even if we plot at the bottom of the scale of the map the maps looks like larger. So we are planning and providing our best knowledge it still not possible plotting maps on scale. But we are draw a projected map on the grid lines on our map.

The package we are using is called "mapproj". It allows that the creation of the projected maps as well as it also contains many types of projection that can be chosen what you want to your mapping. We use the that called "gilbert" projection

```
library(mapproj)

map(database= "world", ylim=c(45,90), xlim=c(-160,-50), col="grey80",
fill=TRUE, projection="gilbert", orientation= c(90,0,225))

lon <- c(-72, -66, -107, -154)  #fake longitude vector

lat <- c(81.7, 64.6, 68.3, 60)  #fake latitude vector

coord <- mapproject(lon, lat, proj="gilbert", orientation=c(90, 0,
225))  #convert points to projected lat/long

points(coord, pch=20, cex=1.2, col="red")  #plot converted points
```

The package what we use called “gilbert” projection which is requires an parameter which is telling that R where North pole is located and the R also take the order where it should be located. And it also plots in the correct place. Once we plotted our map we want to add the point where it should be but it is not working properly. So overcome of this trouble we use our knowledge and get a way of it. At first, we converted our point into our projection what we chosen. And then it plotted on our map. And it dine on our last two lines. From the vectors fake points that created the two lines. The functions ‘mapproject()’ takes the vector of latitude and also the longitude the converted point ‘coord’ after that we can plot our map in normal process. The another function we use which is called ‘grid.lines()’ can be then be used to add grid lines [10].

Figure 4.4: Projection map using mapproj**()** command.

4.5 More extension from R library

4.5.1 Population Map on Projection Map

We covered the projected mapping as well as how we manipulating our data to the map. There are many other function for mapping or other useful plug-ins tools in conjunction of create mapping on the R. symbols and color which is easily implemented on R by using ‘cex’ or ‘col’. A map is plotted the same way as we done before. Like, the ‘layout()’ package. We can also pie chart on our map the command will be ‘plotrix’ package and using the package called “add.pie()”. Not only this package you can also create output easily by using .pdf by using this function on the code “pdf()” and “dev.off()” [11].

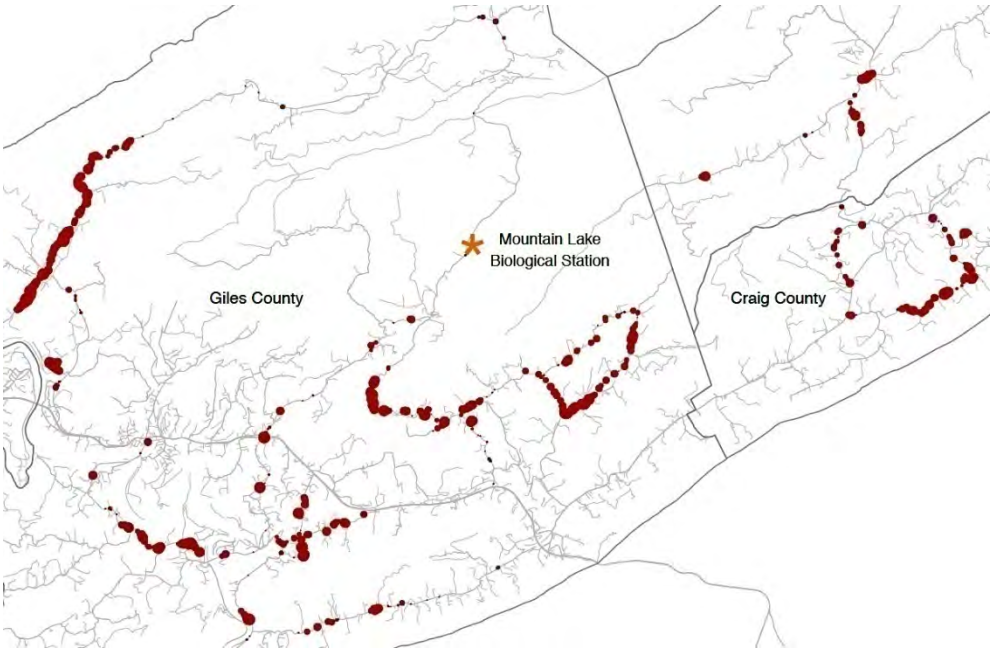

Figure 4.5: Displaying population sizes with graduated point sizes**(**cex=data$pposize) [11].

4.5.2 Pie Chart on projection Map

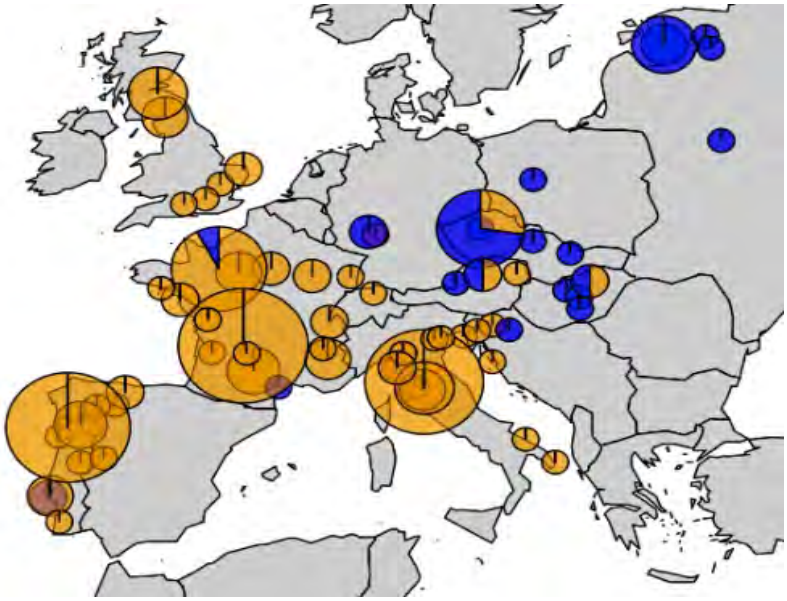

Figure 4.6: Plot pie chart on a map using “add.pie()” in the mapplots’ package [11].

The above diagram it shows that the pie chart on the map. It use by the add.pie(z=c(east,west),x=lon, y=lat, radius=sqrt(tot), col=c(alpha(“orange”,0.6), alpha(“blue”,0.6)),lebels=””) [11].

z indicates the portions of the pie charts which is covered by each type which is given, x and y coordinates for the given point, and also radius to designated the size of circle of the pie chart [12].

To plot all points, we run a loop which will through the data which we given at a time on each pie chart, there are most efficient methods on R but we found it easy [12].

4.6 Improvising the map with ggplot2

On our last diagram on R we want to projectile our map but we cannot do it properly because it is not fitted on the point on scale so we converting into the point that diagram we noticed that the projectile could not point out properly. Regarding this issue now we are using the most powerful package on R which is called ggplot2. Particularly when we don't have the on the fly projection we use the coordinate system mismatch. Like, we use points and after that we use the polygons by themselves. At first we check it on ggmap package.

Read in the point and polygon data:

By reading the dataset, our point data which is in a comma-separated file including with latitude and longitude. Our polygon are a shapefile of Canada .

```
library(rgdal)
library(ggplot2)
#read in point data(tabular data)
mapdata <-read.csv("mapdata.csv", stingAsFactors=FALSE)
head(mapdata)
##       id buildarea latitude longtitude
## 1  10023    12.820    40.76    -73.99
## 1  10027    22.092    40.76    -73.99
## 1 10030B    26.081    40.76    -73.99
## 1  10023    25.597    40.76    -73.98
## 1  10072    3.8775    40.76    -73.82
## 1  10225    16.416    40.76    -73.97


   # the sapefile of canada countries/brought. careful about how you define
the path
   # and layer. i always find this odd
   countries<-readOGR("nybb.shp", layer="nybb")
## OGR data source with driver: ESRI Shapefile
## Source: "nybb.shp" , layer : "nybb"
## with 5 features and 6 fields
## Feature type: wkbMUltiPolygon with 2 dimensions
```

Now we are trying to use the map using the gplot.

```
#map the coutries

ggplot() + geom_polygon(data=counties, aes(x=long, y=lat, group))
```

After using this code we get a map of Canada/

Figure 4.7: ggplot2 package using for a sample map.

Now we are using to point on the map on latitude and longitude by using the ggpolt() package.

```
#map the points

ggplot() + geom_point(data=mapdata, aes(x=longitude, y=latitude),
color="red")
```

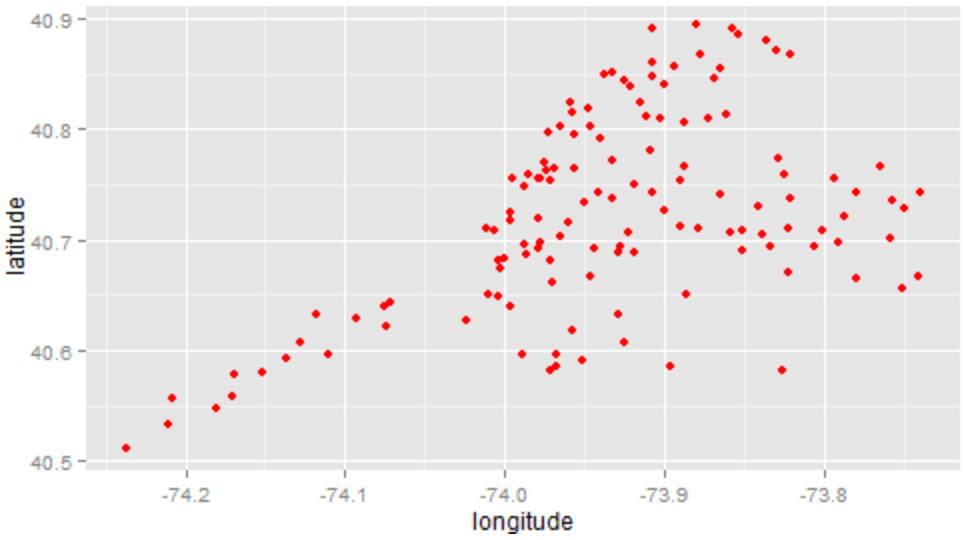

Figure 4.8: Projection using ggplot2 package.

Now we are trying to combine this two for projection.

```
#map bothpolys and points together

ggplot()+geom_polygon(data=countries, aes(x=long, y=lat, group-group))
+

geom_point(data=mapdata, aes(x=longitude, y=latitude), color="red")
```

After combining this two dataset mapping and pointing for the projection and the result makes it horrible because the point and the map that we trying to match by x and y coordinate (x=longitude and y=latitude).

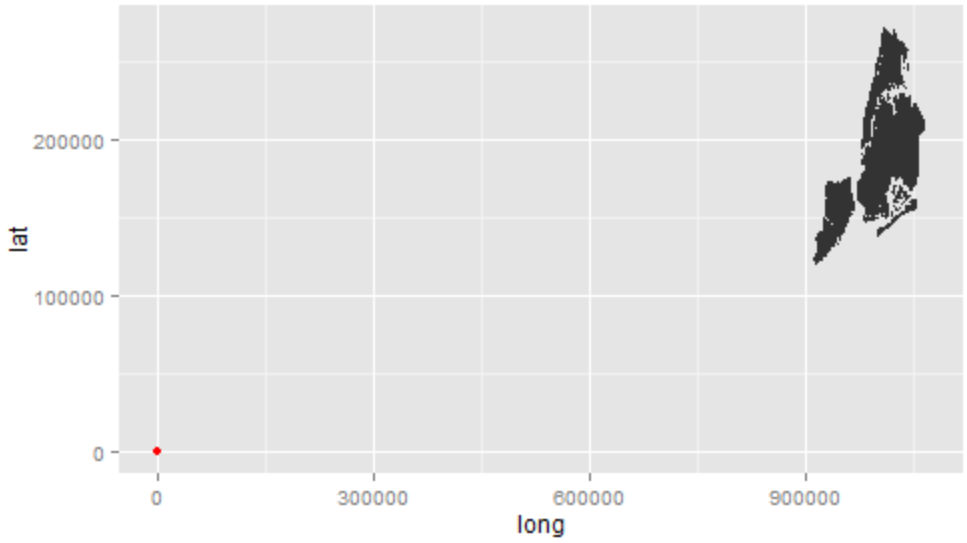

Figure 4.9: Diagram the map by ggplot2 after Marge Ploys and points.

We are trying to get the consistent projection but output shows the inconsistent projections. So by using ggplot we find out that there is no possible way to draw consistent projections.

4.7 Make the projections consistent

For country mapping projection we are using proj4sharing. It included the existing projection system for the layers.[13]

```
proj4string(counties)
```

```
## [1] "+proj=lcc +lat_1=40.66666666666666 +lat_2=41.03333333333333
+lat_0=40.16666666666666 +lon_0=-74 +x_0=300000 +y_0=0 +datum=NAD83
+units=us-ft +no_defs +ellps=GRS80 +towgs84=0,0,0"
```

The projectile format which is very much clear. To get the more reliable format you can also use the .prj file which is come on the shapefile. For more understanding the nice formatting shape file is here [13].

```
PROJCS["NAD_1983_StatePlane_New_York_Long_Island_FIPS_3104_Feet",
    GEOGCS["GCS_North_American_1983",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS_1980",6378137,298.257222101]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.0174532925199433295]],
    PROJECTION["Lambert_Conformal_Conic_2SP"],
    PARAMETER["False_Easting",984249.9999999999],
    PARAMETER["False_Northing",0],
    PARAMETER["Central_Meridian",-74],
    PARAMETER["Standard_Parallel_1",40.66666666666666],
    PARAMETER["Standard_Parallel_2",41.03333333333333],
    PARAMETER["Latitude_Of_Origin",40.16666666666666],
    UNIT["Foot_US",0.30480060960121924],
    AUTHORITY["EPSG","102718"]]
```

Figure 4.10: Using parameter on shapefile [13].

This is a projection for Canada – a island. But looking at the tabular point data file we see the latitude and longitude coordinates not the x and y coordinates it clearly shows the dataset is not projected. We need to project the data point to match the polygon dataset. For doing this we use the command “sptrasform()” package. But before that we need to convert the points data frame into the class SpatialPointFrame [14].

```
class(mapdata)
## [1] "data.frame"
coordinates(mapdata)<-~longitude+latitude
class(mapdata)
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
# does it have a projection/coordinate system assigned?
proj4string(mapdata) # nope
## [1] NA
# we know that the coordinate system is NAD83 so we can manually
# tell R what the coordinate system is
proj4string(mapdata)<-CRS("+proj=longlat +datum=NAD83")
# now we can use the spTransform function to project. We will project
# the mapdata and for coordinate reference system (CRS) we will
# assign the projection from counties
mapdata<-spTransform(mapdata, CRS(proj4string(counties)))
# double check that they match
identical(proj4string(mapdata),proj4string(counties))
## [1] TRUE
```

After converting this by above command we need to connect this to the map

```
# ggplot can't deal with a SpatialPointsDataFrame so we can convert
back to a data.frame

mapdata<-data.frame(mapdata)

# we're not dealing with lat/long but with x/y

# this is not necessary but for clarity change variable names

names(mapdata)[names(mapdata)=="longitude"]<-"x"

names(mapdata)[names(mapdata)=="latitude"]<-"y"

# now create the map

ggplot() +geom_polygon(data=counties, aes(x=long, y=lat,
group=group))+  geom_point(data=mapdata, aes(x=x, y=y), color="red")
```

After connecting the code with the map we can see that the projection is matched but it still hazy because of some missing tools.

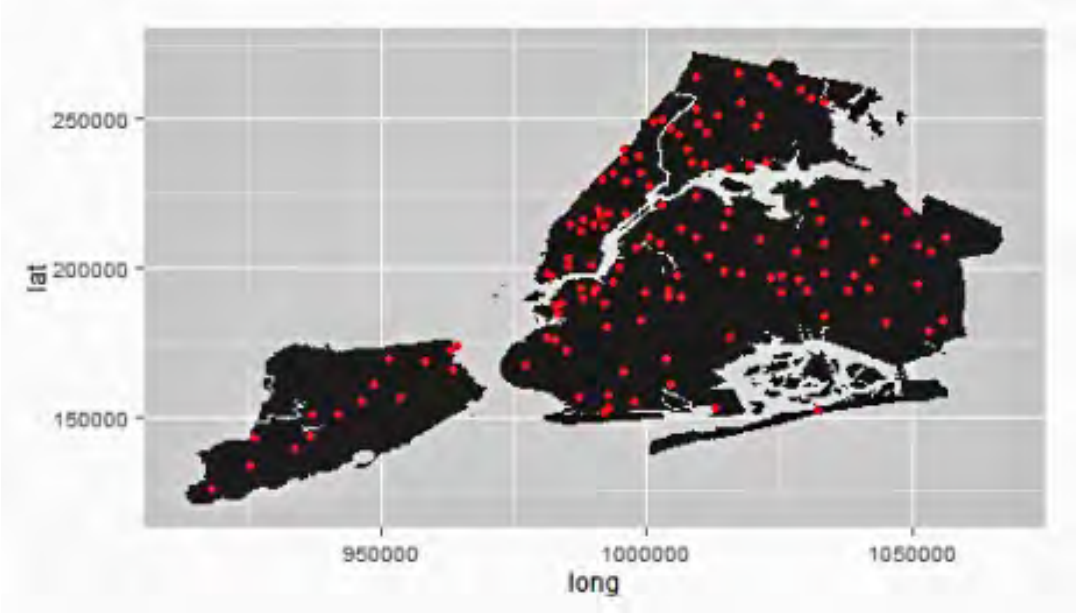

Figure 4.11: Diagram the map after projection and connecting the code.

So the clearer to the map and little bit nicer we all some color gradient and clean up title [14].

```
ggplot() +

    geom_polygon(data=counties, aes(x=long, y=lat, group=group),
fill="grey40",

        colour="grey90", alpha=1)+

    labs(x="", y="", title="Building Area Within 1000m")+ #labels

    theme(axis.ticks.y = element_blank(),axis.text.y =
element_blank(), # get rid of x ticks/text

          axis.ticks.x = element_blank(),axis.text.x =
element_blank(), # get rid of y ticks/text

          plot.title = element_text(lineheight=.8, face="bold",
vjust=1))+ # make title bold and add space

    geom_point(aes(x=x, y=y, color=buildarea), data=mapdata, alpha=1,
size=3, color="grey20")+# to get outline

    geom_point(aes(x=x, y=y, color=buildarea), data=mapdata, alpha=1,
size=2)+

    scale_colour_gradientn("Building\narea (sq-km)",

        colours=c( "#f9f3c2","#660000"))+ # change color scale

    coord_equal(ratio=1) # square plot to avoid the distortion
```

After adding the color gradient the maps looks better than before and we successfully done with our projection on mapping using **ggpolt2**.[14]

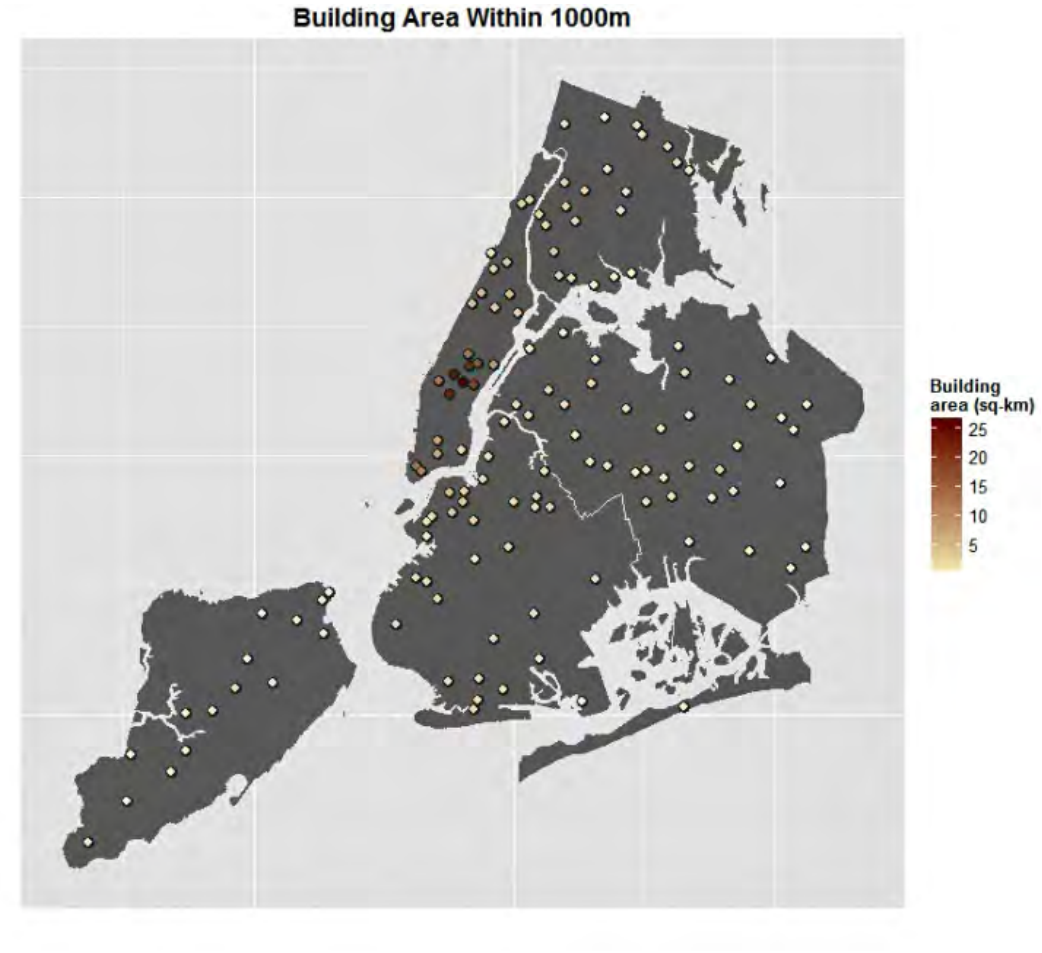

Figure 4.12: After using color code on ggplot2 package.

CHAPTER 05
ACCELERATING R USING CUDA LIBRARIES

5.1 Interfacing with CUDA

On the previous part of our thesis we are improvising the graphical point of view on r mapping. On this section we are using CUDA with r to make the r code mapping with in more efficient way. Although the usage of the hardware has been very limited in the R programming language, it's possible to connect. The way of connect with the programto the GPU can be either OpenCL or CUDA. It is very complex because working with low level language is not easy as high level language but the complex GPU code is more accessible for the R programmer. This idea is behind the gpuRpackage. The aspect of behind gpuR:

1. Applicable on NVIDA CUDA
2. CUDA code to easily incorporate with existing R algorithms.
3. The functions will allow objects to persist on GPU.

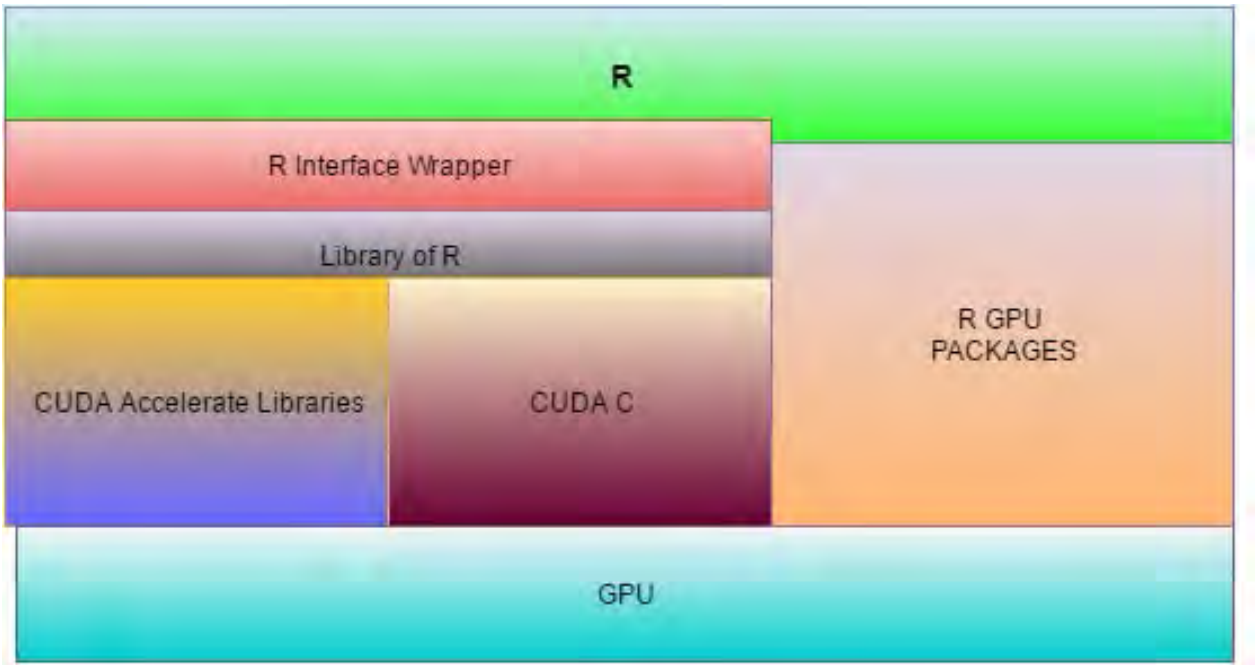

Figure 5.1: The process of R interfaces connect with NVIDA CUDA & implementing existing R GPU packages.

5.2 Application

The gpuR package was for bring the capability of any R programmer/user to computing GPU with a GPU device. there are some package to using GPU for example gputools, cudaBayesreg, gmatrixetc.but all the tools that for R are limited to NVIDIA GPUs. On the contrary, OpenCL would allow all the programmer to get the benefit from the CUDA. ThegpuR package using ViennaCL linear algebra library which can auto-turned OpenCL kernels. The header use in backend the package of RViennaCL. It also access a CUDA backend with NVIDIA GPU's for future improvement [14].

The another way to connect with GPU is Rth. Which is written by Drew Schmidt and also from us, a set of templates to generate CUDA code. And this code is usable on for two different kinds of parallel platforms. Which are GPU and multicore [14].

Like, we computed the distance between rows between matrix. At first, R's built in dist() function, and then using gpuDist() from the package of gputools. For example, for 1000X1000 matrix, the GPU version take 0.258s.

5.3 Writing CUDA code

On our thesis, we are writing a code on CUDA compiler to connect the R package for connecting. This code treat is on C code.

This CUDA code is looking complicated but there are only a few main points which will help you to understand the code.

Most of the part of the code, notice in the comment at the code, it consists of allocating space/memory for our dataset on the GPU and after that moving the data back and forth between the host memories and device memories. Like as,

```
//allocate space for device matrix
cudaMalloc((void**)&dm,msize);
//copy host matrix to device matrix
cudaMemcpy(dm,hm,msize,cudamemcpyHostToDevice);
```

Each SM which is broken into several blocks, of also size which is prompted by the programmer. All the thread within a block will execute in the step which is called lockstep. The programmers also determined the grid size which is amount of block number.

The programmer can also write the Kernel, which is designed by _global_. Every thread runs the kernel, and every thread will measured by their ID number from its block number and also the number of thread within the block

```
total number of thread =
// number of blocks * block size
int totth = gridDim.x * BLOCKSIZE,
//my thread number
me= blockIdx.x * blockDim.x +
    threadIDx.x;
```

The kernel run in our case the call which is procpairs(). The programmer not only supply argument but also determines the block and grid size.

```
//OK, ready to launch Kernal,
  // so configure grid
  dim3 dimGrid (nblk,1);
  dim3 dimBlock (BLOCKSIZE,1,1);
 //launch the kernal
 procpairs<<<dimGrid, dimBlock>>>(dm,dlot,n);
```

Fig 5.5 Define grid size through the block size

According to the shared-memory programming, one of them should avoid the race conditions where multiple threads might be step to each other while writing to shared variables. Let, the instance which has current is 9 and threads is 3 and 6 both want to add 1 to the same time. They both read x as a 9 but they will both write 10. GPU can't handle and multicore too. So there we used CUDA's atomicAdd() to safely update of our total.

5.4 Compiling and Running

In a view of fact we are running two different aspects of hardware platforms at a time. In a C compiling and running also CUDA code when require. For example, on our machine CUDA is in the directory /usr/local/cuda/. This is extension of our library path [13].

```
export CUDA_HOME-/usr/local/cuda/
export path = ($CUDA_HOME/bin $path)
export LD_LIBRARY_PATH=$CUDA_HOME/lib64
```

CUDA has its own compile, nvcc which is invoked on our source file. We want CUDA machine code for hardware version and also included the system file from R studio [13].

The result from this process 2.1s for CUDA code VS about 31.6 s for straight R code. this process is little bit complex CUDA code here is not the optimizer but GPU is definitely a lot faster for this application.

5.5 Testing Performance

After complete and running my code we test the performance between the CPU and the GPU performance

5.5.1 GPU performance

Our desirable image size of our map is 640*640 pixels. So we have to determine the right amount of block and thread number of each block. And we test different block number with different GPU for our desired output.

Another way to get the most reliable output to use the command

Make blocks setup = SETUP

Make block setup = fargo

and a file called fargo.blocks inside setups/fargo is created and is filled with this information, which represents the best block size for each kernel [19].

Here are the output of different block and thread of each block:

```
library(gputools)
N <- 1e3
m <- matrix(sample(100, size = N*N, replace = T), nrow = N)
system.time(gpuDist(m))
//block number, N = 256
//thread number,N = 1024
##    user  system elapsed
##   0.640   0.172   0.812
```

(a)

```
library(gputools)
N <- 1e3
m <- matrix(sample(100, size = N*N, replace = T), nrow = N)
system.time(gpuDist(m))
//block number, N = 534
//thread number,N = 768
##    user  system elapsed
##   0.640   0.1697   0.809
```

(b)

```
library(gputools)
N <- 1e3
m <- matrix(sample(100, size = N*N, replace = T), nrow = N)
system.time(gpuDist(m))
//block number, N = 1024
//thread number,N = 256
##    user  system elapsed
##   0.640   0.1692   0.809
```

(c)

```
library(gputools)
N <- 1e3
m <- matrix(sample(100, size = N*N, replace = T), nrow = N)
system.time(gpuDist(m))
//block number, N = 2048
//thread number,N = 128
##    user  system elapsed
##   0.640   0.1723   0.812
```

(d)

(a) The output of block number 256 and thread number 1024

(b) The output of block number 534 and thread number 768

(c) The output of block number 1024 and thread number 256

(d) The output of block number 2048 and thread number 128

Table 5.5.1 The table of performance on CUDA GPU (different block and measured) and CPU performance for R map (640 * 640 pixel).

Device	Contain	Elapsed Time(user + system) Measured on second(unit)
CUDA GPU	(a)Block number = 256 tread number on each block= 1024	0.812
	(b)Block number =534 tread number on each block = 768	0.8097
	(c)Block number = 1024 tread number on each block = 256	0.8094
	(d)Block number = 2048 tread number on each block = 128	.8123
CPU performance	Using gplot2 package and also help of GPUtools library on R	4.874

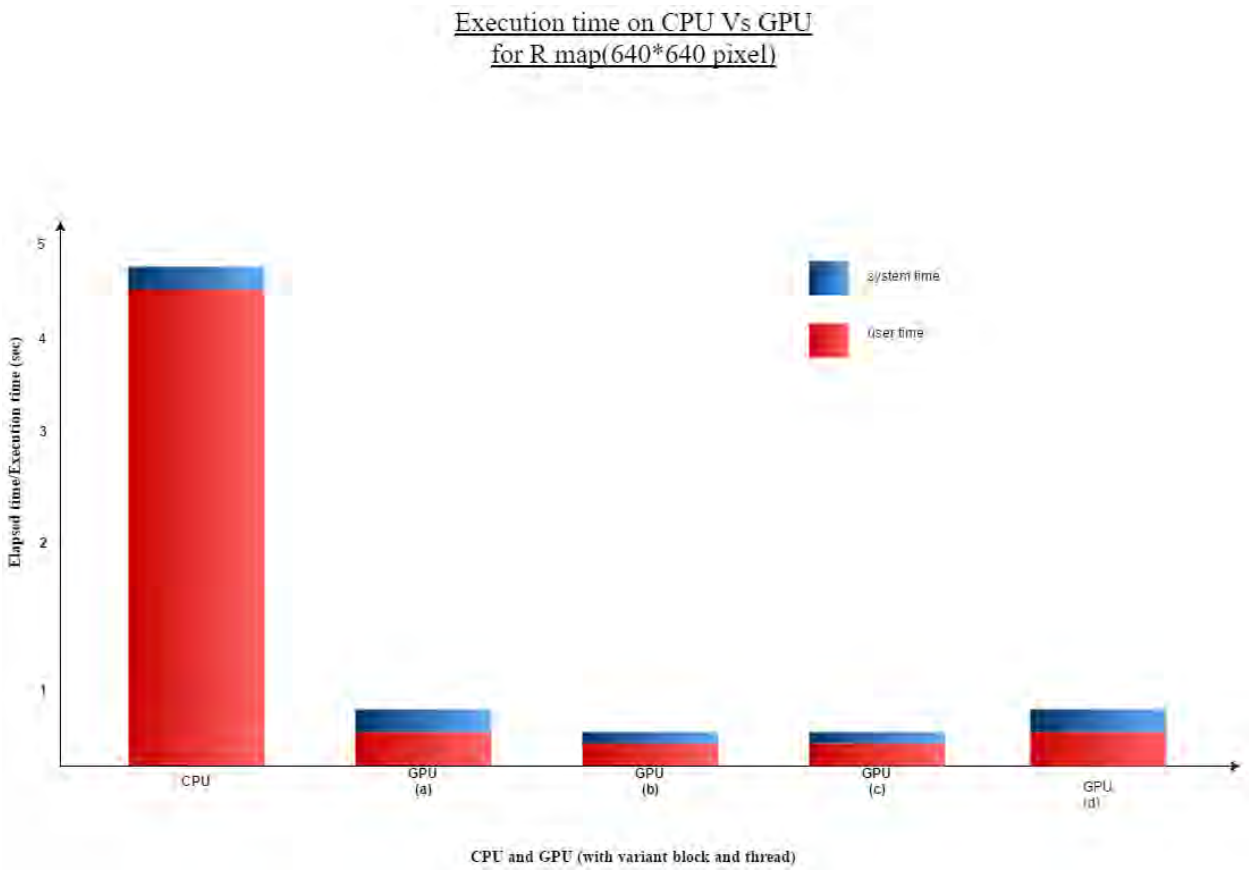

Figure5.2 : The chart of the performance of CPU and GPU performance for R-map (640*640 pixel).

5.6 CUDA library

In plenty of cases, such as our thesis example, one can attend very fast speedup with CUDA. But there is no optimization on the code. If anyone wants more faster as well as more optimization then CUDA will be the encounter because it is the most tweak able platforms any every their upgrades works as a phenomenal working [13].

For direct implementation of any code (if/else) and or any array access ones can do multi-core programming but it will be more complex than you never thought but the GPUs have what amount your programmed run your GPU can handle and manages easily [13]. The number of box size as well as the size which can very good handle in CUDA and there is no counterpart in multi-core programming.

But we found two problems. One is the machine efficiency as well as the human efficiency and second one is NVIDIA GPU series gradually involving code should work for the future.

CHAPTER 6

CHALLENGES & RESOURCES

6.1 Challenges

The problem is challenging first on the GPU architecture. Programming with GPU is not easy since none of us are more familiar with programming on GPU environment. We two only know about the programming in terms of CPU environment. The architecture of GPU is different from CPU, which is more challenging to have proper utilization of all the available resources.

Furthermore, R is totally a new language for us. It is majorly used by Statisticians rather than Computer Engineers, so getting help regarding coding was very few. Also, so tutorials, source codes, and methods for C are very few and scarce. Lastly the codes and methods that we found from various resources are mainly for statistical data analysis, so we had to take fragments of those codes and convert them into algorithms for image mapping.

The most toughest part was interfacing our R libraries with CUDA compilers. While the CUDA ecosystem provides many ways to accelerate applications, R cannot directly call CUDA libraries or launch CUDA kernel functions. To solve this problem, we need to build an interface to bridge R and CUDA . Thus focusing on accelerating R computations using CUDA libraries by calling our own parallel algorithms written in CUDA from R and profiling GPU-accelerated R applications using the CUDA Profiler.

6.2 Resources

For this project, we need the compiler of CUDA and I'll use the machine which has NVIDIA 480 GPU. Specifically, the GPU I will use is NVIDIA GeForce GTX 840M graphics processor is a collection of 15 multiprocessors, with 64 streaming processors each.

CHAPTER 7

CONCLUSION

7.1 Concluding Remarks

GPUs have evolved to the point where many real-world applications are easily implemented on them and run significantly faster than on multi-core systems. Future computing architectures will be hybrid systems with parallel-core GPUs working in tandem with multi-core CPUs. Our main motive in this Thesis is accelerating R libraries using CUDA platform by parallel computing with GPU. We introduced the computation model of R with GPU acceleration, focusing on accelerating R computations using CUDA libraries by calling our own parallel algorithms written in CUDA from R; and profiling GPU-accelerated R applications using the CUDA Profiler.

GPU computing is possible because today's GPU does much more than render graphics: It sizzles with a teraflop of floating point performance and crunches application tasks designed for anything from finance to medicine. The concept of GPU-accelerated parallel computing turns the massive computational power of a modern graphics accelerator's shader pipeline into general-purpose computing power, as opposed to being hard wired solely to do graphical operations. In certain applications requiring massive vector operations, this can yield several orders of magnitude higher performance than a conventional CPU. CUDA is widely deployed through thousands of applications and published research papers and supported by an installed base of over 375 million CUDA-enabled GPUs in notebooks, workstations, compute clusters and supercomputers.

In our Thesis we worked with only mapping, but collaborating R and GPU, geomapping geocoding, statistical analysis, and data plotting is also possible and is majorly used around the world. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available.

7.2 Future works

The potential future directions for research based on the results presented in this thesis can be characterized into the following sections.

7.2.1 Geocoding

Geocoding is the process of transforming a description of a location—such as a pair of coordinates, an address, or a name of a place—to a location on the earth's surface. You can geocode by entering one location description at a time or by providing many of them at once in a table. The resulting locations are output as geographic features with attributes, which can be used for mapping or spatial analysis.

7.2.2 Statistical Data Analysis

R's data structures include vectors, matrices, arrays, data frames and lists. R's extensible object system includes objects for like, regression models, time-series and geo-spatial coordinates. Strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages

REFERENCES

[1] What is GPU Computing? | High-Performance Computing | NVIDIA | NVIDIA. [online]. http://www.nvidia.com/object/what-is-gpu-computing.html, Retrieved April 18, 2016.

[2] D Kirk. “NVIDIA CUDA software and GPU parallel computing architecture”. ISMM. Vol. 7, pp. 103-14. 2007.

[3] CT Yang, CL Huang, CF L in, Hybrid CUDA, “OpenMP, and MPI parallel programming on multicore GPU clusters”, Computer Physics Communications, vol. 182, no. 1, pp. 266-269, 2011.

[4] J Nickolls, I Buck, , M Garland, & K Skadron. “Scalable parallel programming with CUDA”. Queue, Vol. 6, no. 2, pp. 40-53. 2008.

[5] J Fox & R Andersen. (January 2005). Department of Sociology, McMaster University. “Using the R Statistical Computing Environment to Teach Social Statistics Courses”. Retrieved 2006-08-03.

[6] Parallel Programming and Computing Platform | CUDA | NVIDIA | NVIDIA. [online]. http://www.nvidia.com/object/cuda_home_new.html, Retrieved April 18, 2016.

[7] What is R? [online]. https://www.r-project.org/about.html. Retrieved August 09, 2016

[8] Why use the R Language? - Burns Statistics. [online]. http://www.burns-stat.com/documents/tutorials/why-use-the-r-language/, Retrieved August 09, 2016.

[9] Hijmans, R Fri May 8, 08:21:11 CEST 2009.“Define projection using Raster package” [R-sig-Geo].

[10] D Mcllroy & R Brownrigg. “Packed for R, transition to plan 9 codebase”. Map Projection. August, 2005.

[11] “Making Maps with R”. [online]. 2012. Web. 15 Aug. 2016.

[12] P Zhao. “CUDA integration. R for Deep Learning (III): CUDA and MultiGPUs Acceleration”. May 8. 2016.

[13] [online]. https://devtalk.nvidia.com/default/board/57/. Retrieved August 15, 2016.

[14] Parallel Programming with GPUs and R. [online]. https://www.r-bloggers.com/parallel-programming-with-gpus-and-r/, Retrieved August 15, 2016.

[15] MC Borja. “Data Manipulation in R”. Journal of the Royal Statistical Society: Series A (Statistics in Society), volume. 172, no. 3, pp. 699-699. 2009.

[16] What Is GPU Computing? | High-Performance Computing | NVIDIA | NVIDIA. “What Is GPU Computing?” | High-Performance Computing | NVIDIA | NVIDIA. [online]. http://www.nvidia.com/object/what-is-gpu-computing.html. August 15, 2016.

[17] Accelerate R Applications with CUDA. [online]. (2014, August 04). https://devblogs.nvidia.com/parallelforall/accelerate-r-applications-cuda/, Retrieved August 15, 2016.

[18] G Ruetsch & P Micikevicius. “Optimizing matrix transpose in CUDA”. Nvidia CUDA SDK Application Note, 18. 2009.

[19] Improving CUDA Performance.[online]. http://fargo.in2p3.fr/manuals/html/cuda_perf.html, Retrieved August 15, 2016.

APPENDIX

Installation Process

before starting mapping we have to install the R package tools which is called R-studio. The version is 3.1.1.

Download: at first we have to download the free full version software from the website of R. it's open source software.

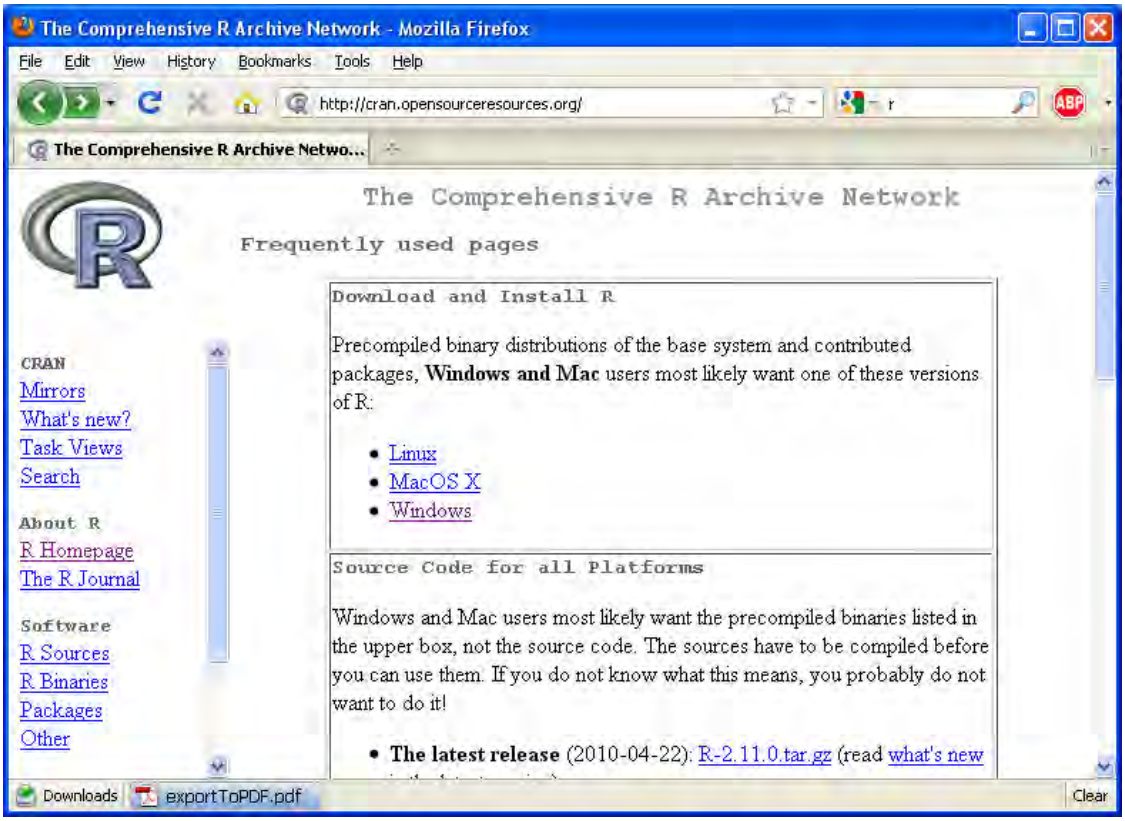

Figure: The website of R studio.

After going on the website then selecting the latest version of R. because without latest version some package and library will missing may be because of that some function could not work.

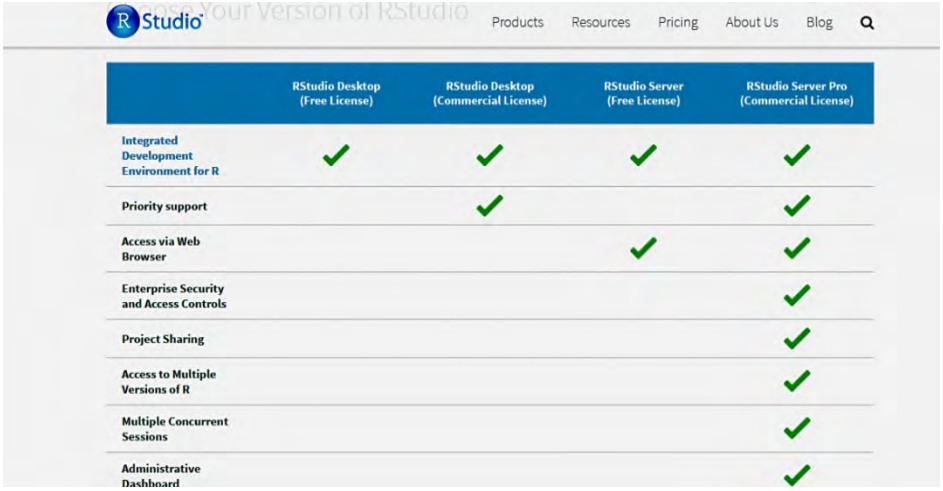

Figure: Choosing the latest version of R studio with packages.

After that we have to install properly by selecting the package we need.

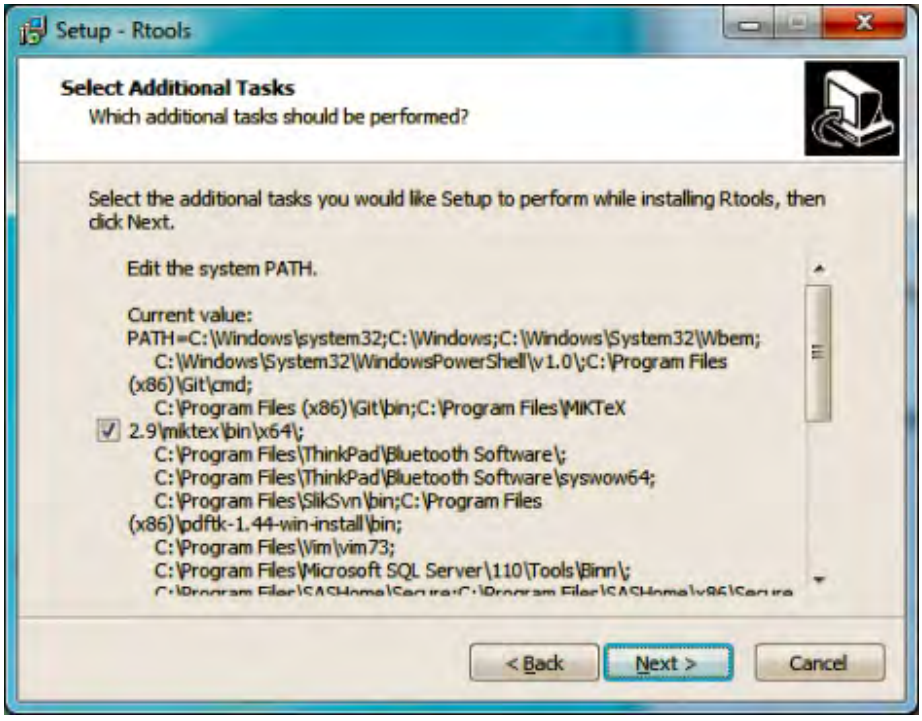

Figure: Installation on the right path.