Handling big spatial (raster/image) data in R (mraster)

Babak Naimi

31/03/2018

Applicant: Babak Naimi, ETH Zurich, Switzerland (naimi.b@gmail.com)

Package: mraster

Introduction

Raster/Image datasets have been increasingly available (e.g., satellite images) in various domains/applications that can be usually considered as big datasets. A big dataset is refered to a dataset that cannot be fitted in a memory, therefore, needs a proper solution to deal with, otherwise, an error is returned indicating the problem. Several packages have been introduced in R to handle and work with raster/image datasets. Among the packages, the package raster is the one that is well-known and rich (in terms of functionalities and implementation), providing the raference classes for different raster data types (e.g., single or multi-bands/layers) that has been used by many users/developers. To process a big datasets, the raster package uses a solution reads the data partially (part of the data, also called a chunk of data), process the chunk, and then go to the next part/chunk, repeat the procedure until all the chunks are read and processed. Although this procedure works, but is not efficient means make the process much slower compared to when the whole dataset is loaded in memory.

As a more efficient solution (see the demo section), we offer to use/adapt the well-known method that maps the memory into the file, means each part of a data file can be accessed randomly without the need of loading the file into the memory. This project aims to provide a framework that offers the required classes and functionalities to use this solution to deal with big spatial raster datasets. This can be adapted for the other types of data but given that raster datasets are potentially big data, they are targeted at the first phase of this project.

The Problem

How can we handle and process a big raster/image dataset (e.g., satellite image) that don't fit in memory with R? How can we make the read/write of such datasets more efficient? How their visualisations can become more efficient? How a standard multi-level tile-mosaicing procedure can be adapted for local file-based raster data storage to optimise their visualisation and handling? How and whether the solution can be adapted to be used by parallelized processing of raster datasets?

The Plan

We will develop an R package (its implementation has already been started) to handle image/raster:

- provide accessing raster datasets without loading them into memory
- easily/efficiently subsetting and binding datasets
- efficiently reading/writing raster/image data
- designing an indexing system together with an efficient organisation of (multi-) file-based storages to optimise the speed of accessing different parts of a dataset
- design a tile-based mosaicing of images to handle multi-level access and maximise efficiency of visualisations
- test whether the solution can be adapted for parallel processing tasks (i.e., a single source of data will be accessed by multiple cores)

We will document the software and provide tutorials and reproducible data analysis examples.

We will also develop and discuss how the functions in the existing packages (e.g., raster) can be benefitted by using this new package.

We will publish the resulting products in an open access form, in a journal (or on a conference) more directed to the Earth observation community.

Timeline:

- Month 1-3: work out design
- Month 4-8: programming the R package, testing with a set of sample datasets
- Month 9-12: write tutorials, develop teaching material and reproducible examples
- Month 9-12: experiment in different platforms and for different processing pipelines

How Can The ISC Help:

I request 10,000 USD that all will be used to develop the R package.

Part of this funding may be used to hire a student assistant for one year (8500 UDS), or will compensate the salary of the applicant for the time that he dedicates to implement the package.

1500 USD will be used to present the results in a relevant conference (e.g., UseR! 2019).

Dissemination

The project has been already started on github, and will be continuing on the same repository, information will be shared and reactions and contributions invited through r-sig-geo, as well as StackOverflow and GIS StackExchange. The project will use an Apache 2.0 license for maximum dissemination. The work will be published in 2 blogs, announced on r-sig-geo, and intermediary results will be presented at UseR! 2019. The final result will be published in a paper submitted to the Journal of Statistical Software; this paper will be available before publication as a package vignette.

Proof of concept

In order to show the problem and as a proof of concept to show how efficient the solution, following I showed some tests based on the preliminary functions that I already implemented in the package.

Example 1:

Let's try a raster file that can be loaded into memory Different solutions are tested: * solution1: when data are loaded in memory * solution2: when data are in disk but will be handled by the method implemented in the raster package * solution3: This is the solution offered by this package, i.e., data are in disk and will be handled by mapping the memory to the file (mraster package)

```r
library(raster)

library(mraster)

library(microbenchmark)
```

```r
#######################
#   Example 1: This example uses a raster file that can be loaded in memory, but
# tests the performance using different solutions
# (i.e., either when the data are in memory or they read from disk)
```

```
# the raster file: 'test.gri'
fsize <- file.size('test1.gri')

fsize # file size in Byte
```

```
## [1] 64974000
```

```
mraster:::.change_unit(fsize,'B','M') # change Byte to MByte
```

```
## [1] 61.96404
```

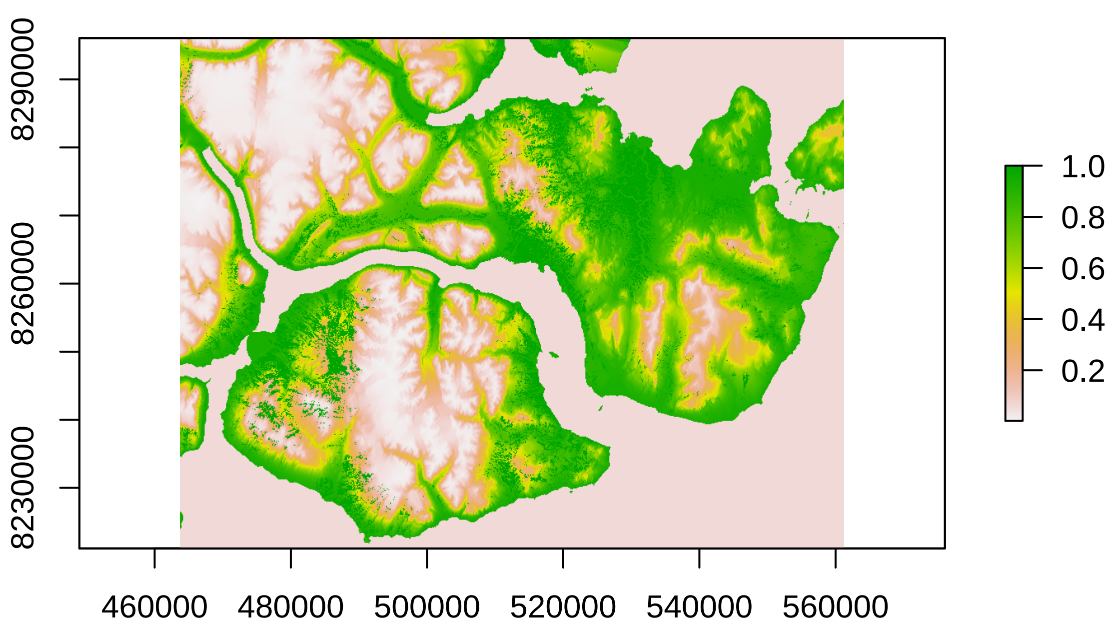
```
#--------------

# read the raster file using the raster package:
in.memory.using.raster <- raster('test1.gri')

in.disk.using.raster <- raster('test1.gri')

in.memory.using.raster <- readAll(in.memory.using.raster) # the raster values are loaded into memory

inMemory(in.memory.using.raster) # test whether the values are in memory or not
```

```
## [1] TRUE
```

```
inMemory(in.disk.using.raster) # the values are in disk
```

```
## [1] FALSE
```

```
plot(in.memory.using.raster) # plot of the image (using the package raster)
```




```
#-------------
# read the raster using the mraster package (using mapping the memory to the file solution):
map.to.file.using.mraster <- mraster('test1.gri')

#-----------------
# to get values at a cell(s), the cell number(s) can be specifies as a vector:
```

```r
# get the value at a cell (e.g., 100000) using the package raster:
in.memory.using.raster[100000]
```

```
##
## 0.05660474
```

```r
in.disk.using.raster[100000]
```

```
##
## 0.05660474
```

```r
# get the value at the same cell using the package mraster:
getRasterValues(map.to.file.using.mraster,100000)
```

```
##        pp2.1
## 1 0.05660474
```

```r
# let's test the performance of getting the values from a vector of cell (indx)

# here, we test and compare extracting of 100,000 records from the raster layer
# using the three solutions:

indx <- 4000000:4100000 # get 100,000 records
b <- microbenchmark(in.memory.using.raster[indx],in.disk.using.raster[indx],
                    getRasterValues(map.to.file.using.mraster,indx),times=10L)

b
```

```
## Unit: milliseconds
##                                              expr        min         lq
##                      in.memory.using.raster[indx] 243.274431 260.284514
##                        in.disk.using.raster[indx] 503.305859 532.744480
##  getRasterValues(map.to.file.using.mraster, indx)   4.575484   4.950232
##        mean     median         uq       max neval
##  279.892269 276.603487 294.430226 323.39981    10
##  648.184716 638.910851 705.333746 918.27502    10
##    8.585603   5.864578   6.503846  35.13607    10
```

```
## The solution offered in this proposal is 75.49671 times faster than when the raster object
##  is on the disk, accessed from the raster package
```

```
## The solution offered in this proposal is 32.60019 times faster than when the raster object
##  is loaded in the memory, accessed from the raster package
```

```r
# here, we test and compare extracting of 1,000,000 records (10 times more) from the raster layer
# using the three solutions:

indx <- 4000000:5000000 # get 1,000,000 records
b <- microbenchmark(in.memory.using.raster[indx],in.disk.using.raster[indx],getRasterValues(map.to.file

b
```

```
## Unit: milliseconds
##                                              expr        min         lq
##                      in.memory.using.raster[indx] 1492.90181 1549.27768
##                        in.disk.using.raster[indx] 1902.46656 2163.17652
##  getRasterValues(map.to.file.using.mraster, indx)   27.64566   29.74609
##        mean     median         uq        max neval
```

```
##  1688.19530 1596.21878 1801.82878 2120.95565    10
##  2238.76951 2243.24026 2399.21297 2655.05274    10
##    43.74737   40.27343   52.47357   81.67089    10
```

```
#---------
```

```
## The solution offered in this proposal is 51.17495 times faster than when the raster object
##  is on the disk, accessed from the raster package
```

```
## The solution offered in this proposal is 38.58964 times faster than when the raster object is
##  loaded in the memory, accessed from the raster package
```

Example 2:

In this example, we try a raster file that cannot be loaded in the memory Two solutions are tested: * solution1: when data are in disk and will be handled by the method implemented in the raster package * solution2: This is the solution offered by this package, i.e., data are in disk and will be handled by mapping the memory to the file (mraster package)

```
#######################
#   Example 2: This example uses a big raster file that cannot be loaded in memory

# the raster file: 'big '
fsize <- file.size('bigRaster.gri')

fsize # file size in Byte
```

```
## [1] 7715865600
```

```
mraster:::.change_unit(fsize,'B','M') # change Byte to MByte
```

```
## [1] 7358.423
```

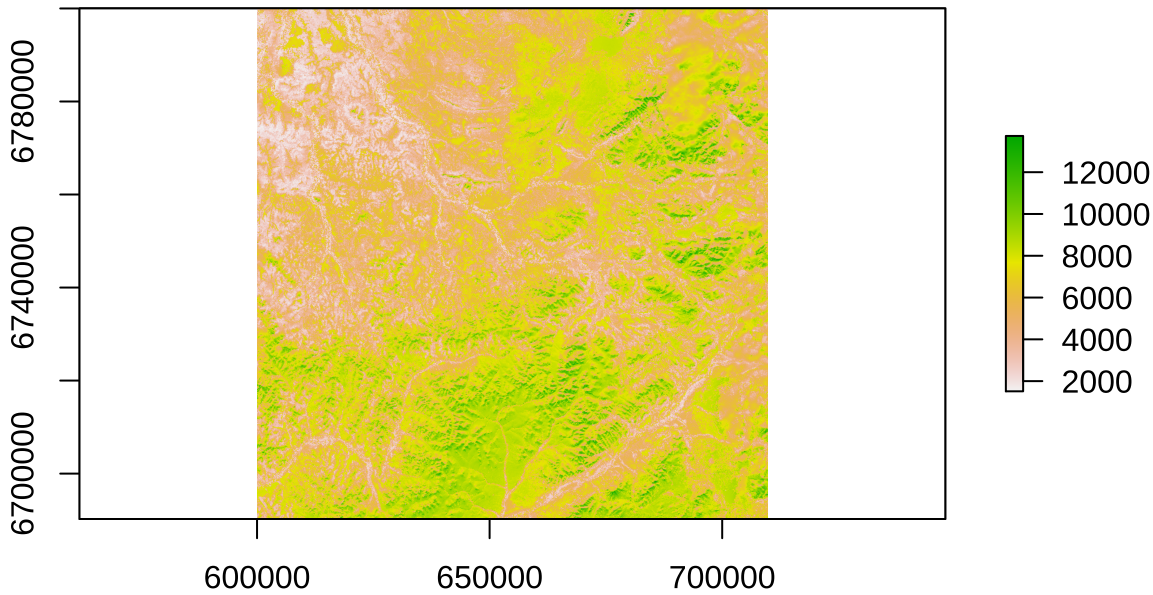
```
mraster:::.change_unit(fsize,'B','G') # the size is > 7 GigaBytes
```

```
## [1] 7.18596
```

```
#--------------

# read the raster file using the raster package:
in.disk.using.raster <- brick('bigRaster.gri') # it is a multi-band (multi-layer) image

inMemory(in.disk.using.raster) # the values are in disk
```

```
## [1] FALSE
```

```
plot(in.disk.using.raster[[1]]) # plot the first layer of the image (using the package raster)
```




```r
# read the raster using the mraster package (using mapping the memory to the file solution):
map.to.file.using.mraster <- mraster('bigRaster.gri')


# here, we test and compare extracting of 100,000 records from the raster layer
# using the two solutions:

indx <- 4000000:4100000 # get 100,000 records
b <- microbenchmark(in.disk.using.raster[indx],getRasterValues(map.to.file.using.mraster,indx),times=2L)

b
```

```
## Unit: milliseconds
##                                                expr         min          lq
##                          in.disk.using.raster[indx] 23447.07678 23447.07678
##  getRasterValues(map.to.file.using.mraster, indx)    36.78917    36.78917
##         mean      median         uq        max neval
##  23573.14200 23573.14200 23699.2072 23699.2072     2
##     77.49986    77.49986   118.2105   118.2105     2
```

```r
#---------
```

```
## The solution offered in this proposal is 304.1701 times faster than when the raster object
##  is on the disk, accessed from the raster package
```

```r
# here, we test and compare extracting of 1,000,000 records (10 times more) from the raster layers
# using the two solutions:

indx <- 4000000:5000000 # get 1,000,000 records
b <- microbenchmark(in.disk.using.raster[indx],getRasterValues(map.to.file.using.mraster,indx),times=2L)

b
```

```
## Unit: milliseconds
##                                                expr         min          lq
##                          in.disk.using.raster[indx] 235615.5747 235615.5747
##  getRasterValues(map.to.file.using.mraster, indx)    325.9104    325.9104
```

```
##         mean      median          uq         max neval
##  237281.9849 237281.9849 238948.3952 238948.3952     2
##     627.2329    627.2329    928.5553    928.5553     2
```

```
## The solution offered in this proposal is 378.2997 times faster than when the raster object
##  is on the disk, accessed from the raster package
```