Analysis of data from aCGH experiments using parallel computing and ff objects

Ramón Díaz-Uriarte[1] and Daniel Rico[1]

March 31, 2012

1. Structural Computational Biology Group. Spanish National Cancer Center (CNIO), Madrid (SPAIN). rdiaz02@gmail.com, drico@cnio.es

Contents

1 Overview:

ADaCGH2 is a package for the analysis of CGH data. The main features of ADaCGH2 are:

- Parallelization of (several of) the main segmentation/calling algorithms currently available, allow efficient usage of computing clusters.
- Data access data using the *ff* package (http://cran.r-project.org/web/packages/ff/index.html), making it possible to analyze data from very large projects.

- Parallelization and *ff* are used simultaneously. WaviCGH Carro et al. (2010) (`http://wavi.bioinfo.cnio.es`), a web-server application for the analysis and visualization of array-CGH data that uses ADaCGH2, consitutes a clear demonstration of the usage of *ff* on a computing cluster with shared storage over NFS.

ADaCGH2 is a major re-write of our former package ADaCGH Díaz-Uriarte and Rueda (2007). We have improved the parallelization and, specially, changed completely the data handling routines. Thanks to the usage of the *ff* package, ADaCGH2 can now analyze data sets of more than four million probes in machines with no more than 2 GB of RAM.

2 A commented example

For both interactive and non-interactive executions we will often execute the following in sequence:

1. Convert the original data to *ff* objects
2. Initialize the computing cluster
3. Carry out segmentation and calling
4. Plot the results

We cover each in turn in the remaining of this section.

```
> library(ADaCGH2)
```

2.1 Choosing a working directory

This package reads and writes quite a few files to the hard drive. The easiest way to organize your work is to create a separated directory for each project. All plot files and ff data will be stored there. At the end of this example, we will remove this directory.

(Just in case, I check for its existence first.)

```
> if(!file.exists("ADaCGH2_vignette_tmp_dir"))
+   dir.create("ADaCGH2_vignette_tmp_dir")
> originalDir <- getwd()
> setwd("ADaCGH2_vignette_tmp_dir")
```

2.2 Convert original data to ff objects

The first two steps need not be carried out in that sequence. Conversion of original data does not use the computing cluster, but can be done after initialization of the cluster. The purpose of this step is to write down the ff files to disk, so they are available for the segmentation and ploting functions. (See below, section 4 for examples of using objects from Limma and snapCGH).

To allow the conversion to be carried out in data from previous sessions, the conversion takes as input the name of an RData.

```
> fname <- list.files(path = system.file("data", package = "ADaCGH2"),
+                     full.names = TRUE, pattern = "inputEx1")
> tableChromArray <- inputDataToADaCGHData(filename = fname)

 Calling gc before returning to track memory usage

          used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1515490 40.5    2403845 64.2  1835812 49.1
Vcells  947210  7.3    1757946 13.5  1377104 10.6

 Files saved in current directory
 /private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc/ADaCGH2_vignette_tmp_di
 chromData.RData, posData.RData, cghData.RData, probeNames.RData

> tableChromArray

   Index ArrayNum ArrayName ChromNum ChromName posInit posEnd
1      1        1       L.1        1         1       1    616
2      2        2       L.2        1         1       1    616
3      3        3        m4        1         1       1    616
4      4        1       L.1        2         2     617    882
5      5        2       L.2        2         2     617    882
6      6        3        m4        2         2     617    882
7      7        1       L.1        3         3     883   1113
8      8        2       L.2        3         3     883   1113
9      9        3        m4        3         3     883   1113
10    10        1       L.1        4         4    1114   1234
11    11        2       L.2        4         4    1114   1234
12    12        3        m4        4         4    1114   1234
13    13        1       L.1        5         5    1235   1425
14    14        2       L.2        5         5    1235   1425
15    15        3        m4        5         5    1235   1425
16    16        1       L.1        6         6    1426   1624
17    17        2       L.2        6         6    1426   1624
```

18	18	3	m4	6	6	1426	1624
19	19	1	L.1	7	7	1625	1825
20	20	2	L.2	7	7	1625	1825
21	21	3	m4	7	7	1625	1825
22	22	1	L.1	8	8	1826	1965
23	23	2	L.2	8	8	1826	1965
24	24	3	m4	8	8	1826	1965
25	25	1	L.1	9	9	1966	2286
26	26	2	L.2	9	9	1966	2286
27	27	3	m4	9	9	1966	2286
28	28	1	L.1	10	10	2287	2519
29	29	2	L.2	10	10	2287	2519
30	30	3	m4	10	10	2287	2519
31	31	1	L.1	11	11	2520	2880
32	32	2	L.2	11	11	2520	2880
33	33	3	m4	11	11	2520	2880
34	34	1	L.1	12	12	2881	3084
35	35	2	L.2	12	12	2881	3084
36	36	3	m4	12	12	2881	3084
37	37	1	L.1	13	13	3085	3172
38	38	2	L.2	13	13	3085	3172
39	39	3	m4	13	13	3085	3172
40	40	1	L.1	14	14	3173	3323
41	41	2	L.2	14	14	3173	3323
42	42	3	m4	14	14	3173	3323
43	43	1	L.1	15	15	3324	3446
44	44	2	L.2	15	15	3324	3446
45	45	3	m4	15	15	3324	3446
46	46	1	L.1	16	16	3447	3601
47	47	2	L.2	16	16	3447	3601
48	48	3	m4	16	16	3447	3601
49	49	1	L.1	17	17	3602	3837
50	50	2	L.2	17	17	3602	3837
51	51	3	m4	17	17	3602	3837
52	52	1	L.1	18	18	3838	3898
53	53	2	L.2	18	18	3838	3898
54	54	3	m4	18	18	3838	3898
55	55	1	L.1	19	19	3899	4141
56	56	2	L.2	19	19	3899	4141
57	57	3	m4	19	19	3899	4141
58	58	1	L.1	20	20	4142	4241
59	59	2	L.2	20	20	4142	4241
60	60	3	m4	20	20	4142	4241
61	61	1	L.1	21	21	4242	4284

```
62    62        2       L.2       21        21    4242   4284
63    63        3        m4       21        21    4242   4284
64    64        1       L.1       22        22    4285   4368
65    65        2       L.2       22        22    4285   4368
66    66        3        m4       22        22    4285   4368
```

The first command is used in this example to find the complete path of the example data set. The actual call to the function is the second expression. The return object is not used for anything further and can be deleted (or not saved).

With large data sets that can be the single step that consumes the most RAM, since we need to load the original data into R. Even if we remove the original data and call `gc()`, R might not return all of the memory to the operating system, and this might be inconvenient in multiuser environments and/or long running processes. Thus, it is possible to execute the above in a separate R process that is spawned exclusively just for the conversion. For instance, we could use the `multicore` package and do:

```
> require(multicore)
> parallel(inputDataToADaCGHData(filename = fname), silent = FALSE)
> tableChromArray <- collect()[[1]]
> if(inherits(tableChromArray, "try-error")) {
+   stop("ERROR in input data conversion")
+ }
>
```

That way, the `ff` are produced and stored locally in the hard drive, but the R process where the original data was loaded (and the conversion to `ff` carried out) dies after the conversion, freeing back the memory to the operating system.

2.3 Initialize computing cluster

Initialization is done with the function `snowfallInit`, a function which uses the package `snowfall` to initialize the cluster and set up the slave nodes. In the example we will use a sockect cluster, but MPI can also be used (our servers use LAM-MPI). In this example we will use 2 nodes.

```
> snowfallInit(universeSize = 2, typecluster = "SOCK")

R Version:  R version 2.15.0 RC (2012-03-22 r58802)

Library ADaCGH2 loaded.
```

2.4 Carry out segmentation and calling

Segmentation and calling are carried out with the `pSegment` functions. Here we show just one such example. Many more are available from the help.

```
> help(pSegment)

> hs_mad.out <- pSegmentHaarSeg("cghData.RData",
+                               "chromData.RData", merging = "MAD")
```

We can take a quick look at the output. We first open the `ff` objects (the output is a list of `ff` objects) and then call `summary` on the smoothed list

```
> lapply(hs_mad.out, open)

$outSmoothed
[1] TRUE

$outState
[1] TRUE

> summary(hs_mad.out[[1]][,])

      L.1                L.2                 m4
 Min.   :-1.49400   Min.   :-1.360000   Min.   :-1.542250
 1st Qu.:-0.07997   1st Qu.:-0.010616   1st Qu.:-0.063417
 Median : 0.01077   Median : 0.008625   Median :-0.016141
 Mean   : 0.00569   Mean   : 0.005947   Mean   : 0.002951
 3rd Qu.: 0.07764   3rd Qu.: 0.013814   3rd Qu.: 0.098133
 Max.   : 0.79718   Max.   : 0.632500   Max.   : 1.342250
```

2.5 Plot the results

Plotting produces PNG files for easier sharing over the Internet. The plotting function takes as main arguments the names of RData files. We first write to disk the (`ff`) object with the results, and then call the plotting function:

```
> save(hs_mad.out, file = "hs_mad.out.RData", compress = FALSE)
> pChromPlot(outRDataName = "hs_mad.out.RData",
+                             cghRDataName = "cghData.RData",
+                             chromRDataName = "chromData.RData",
+                             posRDataName = "posData.RData",
+                             probenamesRDataName = "probeNames.RData",
+                             imgheight = 350)
```

3 Using CGHregions

The CGHregions package van de Wiel (2009) is a BioConductor package that implements a well known method van de Wiel and van Wieringen (2007) for dimension reduction for aCGH data (see a review of common regions issues and methods in Rueda and Diaz-uriarte (2010)).

The `CGHregions` function accepts different type of input, among others a data frame. The function `outputToCGHregions` produces that data frame, ready to be used as input to CGHregions (for the next example, you will need to have the *CGHregions* package installed).

```
> forcghr <- outputToCGHregions(hs_mad.out)
> if(require(CGHregions)) {
+   regions1 <- CGHregions(forcghr)
+   regions1
+ }

[1] 1 0 7
[1] 2 0 7
[1] "Tuning on small data set finished...started with entire data set"
[1]  2.000000000  0.004996431 21.000000000  1.000000000
[1] "Average error threshold is satisfied for any distance value c"
[1] "c = 3, nr of regions: 22"
[1] "Finished with entire data set."
cghRegions (storageMode: lockedEnvironment)
assayData: 22 features, 3 samples
  element names: regions
protocolData: none
phenoData: none
featureData
  featureNames: 1 2 ... 22 (22 total)
  fvarLabels: Chromosome Start ... AveDist (5 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:

>
```

Please note that `outputToCGHregions` does NOT check if the calls are something that can be meaningfully passed to CGHregions. In particular, you probably do NOT want to use this function when `pSegment` has been called using `merging = "none"`.

4 Input data from Limma and snapCGH

Many aCGH studies use pre-processing steps similar to those of gene expression data. The `MAList` object, from *Limma* and `SegList` object, from *snapCGH*, are commonly used to store aCGH information. The following examples illustrate the usage of the function `inputDataToADaCGHData` to convert `MAList` and `SrgList` data into a format suitable for *ADaCGH2*.

We will start with objects produced by *snapCGH*. The following code is copied from the *snapCGH* vignette (pp. 2 and 3). Please check the original vignette for details. In summary, a set of array files are read, the data are normalized and, finally, averaged over clones. snapCGH uses limma for the initial import of data and, next, with the `read.clonesinfo` function adds additional information such as chromosome and position. The `MA` object created is of class `MAList`, but with added information (compared to a basic, original, limma MAList object). `MA2` is of type `SegList`.

```
> datadir <- system.file("testdata", package = "snapCGH")
> targets <- readTargets("targets.txt", path = datadir)
> RG1 <- read.maimages(targets$FileName, path = datadir, source = "genepix")

Read /Library/Frameworks/R.framework/Versions/2.15/Resources/library/snapCGH/testdat
Read /Library/Frameworks/R.framework/Versions/2.15/Resources/library/snapCGH/testdat

> RG1 <- read.clonesinfo("cloneinfo.txt", RG1, path = datadir) ## snapCGH-specific
> RG1$printer <- getLayout(RG1$genes)
> types <- readSpotTypes("SpotTypes.txt", path = datadir)
> RG1$genes$Status <- controlStatus(types, RG1)

Matching patterns for: ID Name Chr
Found 21 CTD-
Found 9 CTA-
Found 405 Chrom8
Found 66 Chrom3
Found 75 Chrom1
Setting attributes: values Color

> RG1$design <- c(-1, -1)
> RG2 <- backgroundCorrect(RG1, method = "minimum") ## class RGList
> MA <- normalizeWithinArrays(RG2, method = "median") ## class MAList
> class(MA)

[1] "MAList"
attr(,"package")
[1] "limma"

> MA2 <- processCGH(MA, method.of.averaging = mean, ID = "ID") ## class SegList
```

```
Averaging duplicated clones
Processing chromosome  1
Processing chromosome  2
Processing chromosome  3
Processing chromosome  4
Processing chromosome  5
Processing chromosome  6
Processing chromosome  7
Processing chromosome  8
Processing chromosome  9
Processing chromosome  10
Processing chromosome  11
Processing chromosome  12
Processing chromosome  13
Processing chromosome  14
Processing chromosome  15
Processing chromosome  16
Processing chromosome  17
Processing chromosome  18
Processing chromosome  19
Processing chromosome  20
Processing chromosome  21
Processing chromosome  22
```

```
> class(MA2)
```

```
[1] "SegList"
attr(,"package")
[1] "snapCGH"
```

```
>
>
```

All the information (intensity ratios and location) is available in the `MA` and `MA2` objects. We can directly convert them to *ADaCGH2* objects (we set `na.omit = TRUE` as the data contain missing values). The first call process the `MAList` and the second the `SegList`.

```
> tmp <- inputDataToADaCGHData(MAList = MA, na.omit = TRUE)
```

```
 We have identical MidPos!!!


 Calling gc before returning to track memory usage
```

```
          used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1643702 43.9    2564037 68.5  2190143 58.5
Vcells 1102147  8.5    1925843 14.7  1925786 14.7

 Files saved in current directory
 /private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc/ADaCGH2_vignette_tmp_di
 chromData.RData, posData.RData, cghData.RData, probeNames.RData

> tmp <- inputDataToADaCGHData(MAList = MA2, na.omit = TRUE,
+                              minNumPerChrom = 4)

 Calling gc before returning to track memory usage

          used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1644169 44.0    2564037 68.5  2190143 58.5
Vcells 1103004  8.5    1925843 14.7  1925786 14.7

 Files saved in current directory
 /private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc/ADaCGH2_vignette_tmp_di
 chromData.RData, posData.RData, cghData.RData, probeNames.RData

>
>
```

We need to change the argument to `minNumPerChrom` because, after the data processing step in `processCGH`, chromosome 21 has only four observations.

The original `MAList` as produced directly from *limma* do not have chromosome and position information. That is what the `read.clonesinfo` function from *snapCGH* did. To allow using objects directly from *limma* and incorporating position information, we will use an approach to directly mimicks that in *snapCGH*. If you use and `MAList` you can also provide a `cloneinfo` argument; this can be either the full path to a file with the format required by `read.clonesinfo` or, else, the name of an object with (at least) three columns, names `ID`, `Chr`, and `Position`.

We copy from the limma vignette (section 3.2, p.8), changing the names of objets by appending ".limma".

```
> targets.limma <- readTargets("targets.txt", path = datadir)
> RG.limma <- read.maimages(targets.limma, path = datadir, source="genepix")

Read /Library/Frameworks/R.framework/Versions/2.15/Resources/library/snapCGH/testdat
Read /Library/Frameworks/R.framework/Versions/2.15/Resources/library/snapCGH/testdat

> RG.limma <- backgroundCorrect(RG.limma, method="normexp", offset=50)
```

```
Array 1 corrected
Array 2 corrected
Array 1 corrected
Array 2 corrected
```

```
> MA.limma <- normalizeWithinArrays(RG.limma)
>
```

We can add the chromosomal and position information in two different ways. First, as in `read.clonesinfo` or, else, we can provide the name of a file (with the same format as required by `read.clonesinfo`). Note that `fclone` is a path (and, thus, a character vector).

```
> fclone <- list.files(path = system.file("testdata", package = "snapCGH"),
+                      full.names = TRUE, pattern = "cloneinfo.txt")
> fclone
```

```
[1] "/Library/Frameworks/R.framework/Versions/2.15/Resources/library/snapCGH/testdat
```

```
> tmp <- inputDataToADaCGHData(MAList = MA.limma,
+                              cloneinfo = fclone,
+                              na.omit = TRUE)
```

```
Assuming cloneinfo is a file (possibly with full path)
 We have identical MidPos!!!


 Calling gc before returning to track memory usage

         used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1645098 44.0    2564037 68.5  2190143 58.5
Vcells 1135473  8.7    1925843 14.7  1925786 14.7

 Files saved in current directory
 /private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc/ADaCGH2_vignette_tmp_di
 chromData.RData, posData.RData, cghData.RData, probeNames.RData
```

```
>
```

Alternatively, we can provide the name of an object with the additional information. For illustrative purposes, we can use here the columns of the `MA` object.

```
> acloneinfo <- MA$genes
> tmp <- inputDataToADaCGHData(MAList = MA.limma,
+                              cloneinfo = acloneinfo,
+                              na.omit = TRUE)
```

```
Assuming cloneinfo is an R data frame
 We have identical MidPos!!!


 Calling gc before returning to track memory usage

          used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1645057 44.0    2564037 68.5  2190143 58.5
Vcells 1125824  8.6    1925843 14.7  1925786 14.7

 Files saved in current directory
 /private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc/ADaCGH2_vignette_tmp_di
 chromData.RData, posData.RData, cghData.RData, probeNames.RData

>
>
```

5 Clean up actions

These are not strictly necessary, but we will explicitly stop the cluster and remove the directory we created. To make sure there are no file permission problems, we also explicitly delete some of the "ff" files and objects (and we wait a few seconds to allow pending I/O operations to happen before we delete the directory).

```
> sfStop()
> load("chromData.RData")
> load("posData.RData")
> load("cghData.RData")
> delete(cghData); rm(cghData)

[1] TRUE

> delete(posData); rm(posData)

[1] TRUE

> delete(chromData); rm(chromData)

[1] TRUE

> lapply(hs_mad.out, delete)

$outSmoothed
[1] TRUE
```

```
$outState
[1] TRUE

> rm(hs_mad.out)
> setwd(originalDir)
> print(getwd())

[1] "/private/tmp/RtmpmUhGQ3/Rbuild226551d28200/ADaCGH2/inst/doc"

> Sys.sleep(3)

> unlink("ADaCGH2_vignette_tmp_dir", recursive = TRUE)
> print(dir())

[1] "ADaCGH2.Rnw" "ADaCGH2.bib" "ADaCGH2.tex"

> Sys.sleep(3)
```

References

Carro, A., Rico, D., Rueda, O. M., Díaz-Uriarte, R., and Pisano, D. G. (2010). waviCGH: a web application for the analysis and visualization of genomic copy number alterations. *Nucleic acids research*, 38 Suppl:W182–7.

Díaz-Uriarte, R. and Rueda, O. M. (2007). ADaCGH: A parallelized web-based application and R package for the analysis of aCGH data. *PloS one*, 2(1):e737.

Rueda, O. M. and Diaz-uriarte, R. (2010). Finding Recurrent Copy Number Alteration Regions : A Review of Methods. *Current Bioinformatics*, 5:1–17.

van de Wiel, M. a. and van Wieringen, W. N. (2007). CGHregions: Dimension Reduction for Array CGH Data with Minimal Information Loss. *Cancer informatics*, 3(0):55–63.

van de Wiel, S. V. . M. (2009). *CGHregions: Dimension Reduction for Array CGH Data with Minimal Information Loss.* R package version 1.7.1.