Analysis of data from aCGH experiments using parallel computing and ff objects: long list of examples

Ramón Díaz-Uriarte

3-October-2014

1. Department of Biochemistry, Universidad Autonoma de Madrid Instituto de Investigaciones Biomedicas “Alberto Sols” (UAM-CSIC), Madrid (SPAIN).

rdiaz02@gmail.com

Contents

1 This vignette

2 Creating objects

3 The examples

  3.1 RAM objects and forking
  3.2 ff objects and cluster
  3.3 ff objects and forking
  3.4 Comparing output

4 Exercising the code for the load balancing options

5 Clean up actions

1 This vignette

We provide here example calls of all segmentation methods, with different options for methods, as well as different options for type of input object and clustering. This is provided here as both extended help and as a simple way of checking that all the functions can be run and yield the same results regardless of type of input and clustering.

2 Creating objects

We must ensure that we can run this vignette as stand alone. Thus, we load the package and create all necessary objects. This repeats work done in the main vignette.

We first try to move to the “/tmp” directory, if it exists. If it does not, the code will be executed in your current directory.

> try(setwd("~/tmp"))

> library(ADaCGH2)
> ## loading in-RAM objects
> data(inputEx)
> summary(inputEx)
### ID chromosome position L.1

- **Hs.101850**: 1 Min. : 1.000 Min. : 1180411 Min. : -1.07800
- **Hs.1019**: 1 1st Qu. : 1.000 1st Qu.: 36030889 1st Qu.: -0.22583
- **Hs.105460**: 1 Median : 2.000 Median : 70805790 Median : -0.01600
- **Hs.105656**: 1 Mean : 2.284 Mean : 92600349 Mean : -0.03548
- **Hs.105941**: 1 3rd Qu.: 3.000 3rd Qu.: 149843856 3rd Qu.: 0.16000
- **Hs.106674**: 1 Max. : 5.000 Max. : 243795357 Max. : 0.88300

### (Other) : 494 NA's : 5

### L.2 m4 m5 L3

- Min. : -0.795000 Min. : -0.1867 Min. : -4.67275 Min. : -13.273
- 1st Qu.: -0.139000 1st Qu.: 1.9790 1st Qu.: -0.02025 1st Qu.: 3.631
- Median : 0.007684 Median : 3.4504 Median : 1.60159 Median : 1.981
- 3rd Qu.: 0.134000 3rd Qu.: 5.8235 3rd Qu.: 3.04475 3rd Qu.: 4.110

### NA's : 15 NA's : 41 NA's : 9

### m6

- Min. : -0.7655
- 1st Qu.: -0.2260
- Median : -0.0440
- Mean : -0.0351
- 3rd Qu.: 0.1620
- Max. : 0.7750

### NA's : 203

---

```r
> head(inputEx)

<table>
<thead>
<tr>
<th>ID</th>
<th>chromosome</th>
<th>position</th>
<th>L.1</th>
<th>L.2</th>
<th>m4</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hs.212680</td>
<td>Hs.212680</td>
<td>1180411</td>
<td>NA</td>
<td>0.038</td>
<td>6.22625</td>
<td></td>
</tr>
<tr>
<td>Hs.129780</td>
<td>Hs.129780</td>
<td>1188042</td>
<td>NA</td>
<td>0.028</td>
<td>6.17425</td>
<td></td>
</tr>
<tr>
<td>Hs.42806</td>
<td>Hs.42806</td>
<td>1194444</td>
<td>NA</td>
<td>0.042</td>
<td>6.17425</td>
<td></td>
</tr>
<tr>
<td>Hs.76239</td>
<td>Hs.76239</td>
<td>1332537</td>
<td>NA</td>
<td>0.285</td>
<td>5.62425</td>
<td></td>
</tr>
<tr>
<td>Hs.40500</td>
<td>Hs.40500</td>
<td>2362211</td>
<td>NA</td>
<td>0.058</td>
<td>5.85125</td>
<td></td>
</tr>
<tr>
<td>Hs.449936</td>
<td>Hs.449936</td>
<td>2372287</td>
<td>0.294</td>
<td>-0.006</td>
<td>5.68525</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m5</th>
<th>L3</th>
<th>m6</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
```

---

```r
> cgh.dat <- inputEx[, -c(1, 2, 3)]
> chrom.dat <- as.integer(inputEx[, 2])
> pos.dat <- inputEx[, 3]
> ## choosing working dir for cluster
> originalDir <- getwd()
> if(!file.exists("ADaCGH2_vignette_tmp_dir"))
+  dir.create("ADaCGH2_vignette_tmp_dir")
> setwd("ADaCGH2_vignette_tmp_dir")
> ## creating ff objects
> fnameRdata <- list.files(path = system.file("data", package = "ADaCGH2"),
+  full.names = TRUE, pattern = "inputEx.RData")
```

---

2
> inputToADaCGH(ff.or.RAM = "ff", 
+ RDatafilename = fnameRdata)

... done reading; starting checks

... checking identical MidPos

... checking need to reorder inputData, data.frame version

... done with checks; starting writing

... done writing/saving probeNames

... done writing/saving chromData

... done writing/saving posData

... done writing/saving cghData

Calling gc at end

used (Mb) gc trigger (Mb) max used (Mb)
Ncells 1556933 83.2 2403845 128.4 1835812 98.1
Vcells 1548126 11.9 2481603 19.0 1922758 14.7

Files saved in current directory
/home/ramon/tmp/ADaCGH2_vignette_tmp_dir
with names :

> ## setting random number generator for forking
> RNGkind("L'Ecuyer-CMRG")
> ## initializing cluster and setting up random number generator
> number.of.nodes <- detectCores()
> cl2 <- parallel::makeCluster(number.of.nodes,"PSOCK")
> parallel::clusterSetRNGStream(cl2)
> parallel::setDefaultCluster(cl2)
> parallel::clusterEvalQ(NULL, library("ADaCGH2"))

[[1]]
[1] "ADaCGH2" "ff" "bit" "parallel" "methods" "stats"
[7] "graphics" "grDevices" "utils" "datasets" "base"

[[2]]
[1] "ADaCGH2" "ff" "bit" "parallel" "methods" "stats"
[7] "graphics" "grDevices" "utils" "datasets" "base"

[[3]]
[1] "ADaCGH2" "ff" "bit" "parallel" "methods" "stats"
[7] "graphics" "grDevices" "utils" "datasets" "base"

[[4]]
> ## verify we are using the right version of ADaCGH2
> parallel::clusterEvalQ(NULL,
+   library(help = ADaCGH2)$info[[1]][[2]])

[[1]]
[1] "Version: 2.5.2"

[[2]]
[1] "Version: 2.5.2"

[[3]]
[1] "Version: 2.5.2"

[[4]]
[1] "Version: 2.5.2"

[[5]]
[1] "Version: 2.5.2"
[[6]]
[1] "Version: 2.5.2"
[[7]]
[1] "Version: 2.5.2"
[[8]]
[1] "Version: 2.5.2"
[[9]]
[1] "Version: 2.5.2"
[[10]]
[1] "Version: 2.5.2"
[[11]]
[1] "Version: 2.5.2"
[[12]]
[1] "Version: 2.5.2"
[[13]]
[1] "Version: 2.5.2"
[[14]]
[1] "Version: 2.5.2"
[[15]]
[1] "Version: 2.5.2"
[[16]]
[1] "Version: 2.5.2"
[[17]]
[1] "Version: 2.5.2"
[[18]]
[1] "Version: 2.5.2"
[[19]]
[1] "Version: 2.5.2"
[[20]]
[1] "Version: 2.5.2"
[[21]]
[1] "Version: 2.5.2"
[[22]]
[1] "Version: 2.5.2"
Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2

Version: 2.5.2
> wdir <- getwd()
> parallel::clusterExport(NULL, "wdir")
> parallel::clusterEvalQ(NULL, setwd(wdir))

[[1]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[2]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[3]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[4]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[5]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[6]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[7]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[8]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[9]]
[[27]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[28]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[29]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[30]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[31]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[32]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[33]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[34]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[35]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[36]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[37]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[38]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[39]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[40]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[41]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[42]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[43]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"
[[44]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[45]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[46]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[47]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[48]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[49]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[50]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[51]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[52]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[53]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[54]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[55]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[56]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[57]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[58]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[59]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[60]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"

[[61]]
3 The examples

3.1 RAM objects and forking

```r
> cbs.mergel.RAM.fork <- pSegmentDNAcopy(cgh.dat, chrom.dat,
+       merging = "mergeLevels")
> cbs.mad.RAM.fork <- pSegmentDNAcopy(cgh.dat, chrom.dat, merging = "MAD")
> cbs.none.RAM.fork <- pSegmentDNAcopy(cgh.dat, chrom.dat, merging = "none")
> hmm.mergel.RAM.fork <- pSegmentHMM(cgh.dat, chrom.dat, merging = "mergeLevels")
> hmm.mad.RAM.fork <- pSegmentHMM(cgh.dat, chrom.dat, merging = "MAD")
> hs.mergel.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+       merging = "mergeLevels")
> hs.mad.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+       merging = "MAD")
> hs.none.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+       merging = "none")
> glad.RAM.fork <- pSegmentGLAD(cgh.dat, chrom.dat)
> biohmm.mergel.RAM.fork <- pSegmentBioHMM(cgh.dat,
+       chrom.dat,
+       pos.dat,
+       merging = "mergeLevels")
> biohmm.mad.RAM.fork <- pSegmentBioHMM(cgh.dat,
+       chrom.dat,
+       pos.dat,
+       merging = "MAD")
> biohmm.mad.bic.RAM.fork <- pSegmentBioHMM(cgh.dat,
+       chrom.dat,
+       pos.dat,
+       merging = "MAD",
+       aic.or.bic = "BIC")
> cghseg.mergel.RAM.fork <- pSegmentCGHseg(cgh.dat,
+       chrom.dat,
+       merging = "mergeLevels")
> cghseg.mad.RAM.fork <- pSegmentCGHseg(cgh.dat,
+       chrom.dat,
+       merging = "MAD")
> cghseg.none.RAM.fork <- pSegmentCGHseg(cgh.dat,
+       chrom.dat,
+       merging = "none")
```
3.2 \textit{ff} objects and cluster

Compared to the section 3.1, the main differences are that we explicitly set the typeParall argument to "cluster" (the default is "fork") and the change in the names of the input data (which now refer to the names of the RData objects that contain the \textit{ff} objects).

```r
> waves.mergel.RAM.fork <- pSegmentWavelets(cgh.dat, + chrom.dat, merging = "mergeLevels")
> waves.mad.RAM.fork <- pSegmentWavelets(cgh.dat, + chrom.dat, merging = "MAD")
> waves.none.RAM.fork <- pSegmentWavelets(cgh.dat, + chrom.dat, merging = "none")
>
> cbs.mergel.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData", + merging = "mergeLevels", + typeParall = "cluster")
> cbs.mad.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData", + merging = "MAD", + typeParall = "cluster")
> cbs.none.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData", + merging = "none", + typeParall = "cluster")
> hmm.mergel.ff.cluster <- pSegmentHMM("cghData.RData", "chromData.RData", + merging = "mergeLevels", + typeParall = "cluster")
> hmm.mad.ff.cluster <- pSegmentHMM("cghData.RData", "chromData.RData", + merging = "MAD", + typeParall = "cluster")
> hs.mergel.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData", + merging = "mergeLevels", + typeParall = "cluster")
> hs.mad.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData", + merging = "MAD", typeParall = "cluster")
> hs.none.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData", + merging = "none", typeParall = "cluster")
> glad.ff.cluster <- pSegmentGLAD("cghData.RData", "chromData.RData", + typeParall = "cluster")
> biohmm.mergel.ff.cluster <- pSegmentBioHMM("cghData.RData", + "chromData.RData", + "posData.RData", + merging = "mergeLevels", + typeParall = "cluster")
> biohmm.mad.ff.cluster <- pSegmentBioHMM("cghData.RData", + "chromData.RData", + "posData.RData", + merging = "MAD", + typeParall = "cluster")
> biohmm.mad.bic.ff.cluster <- pSegmentBioHMM("cghData.RData", + "chromData.RData", + "posData.RData", + merging = "MAD", + typeParall = "cluster")
```
3.3  **ff** objects and forking

The main difference with section 3.2 is the argument `typeParall`; we did not need to pass it explicitly (since the default is `fork`), but we will do for clarity.

```r
> cbs.mergel.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
  + merging = "mergeLevels",
  + typeParall = "fork")
> cbs.mad.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
  + merging = "MAD",
  + typeParall = "fork")
> cbs.none.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
  + merging = "none",
  + typeParall = "fork")
> hmm.mergel.ff.fork <- pSegmentHMM("cghData.RData", "chromData.RData",
  + merging = "mergeLevels",
  + typeParall = "fork")
> hmm.mad.ff.fork <- pSegmentHMM("cghData.RData", "chromData.RData",
  + merging = "MAD",
  + typeParall = "fork")
> hs.mergel.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
  + merging = "mergeLevels",
  + typeParall = "fork")
> hs.mad.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
  + merging = "MAD",
  + typeParall = "fork")
> hs.none.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
  + merging = "none",
  + typeParall = "fork")
> glad.ff.fork <- pSegmentGLAD("cghData.RData", "chromData.RData",
  + aic.or.bic = "BIC",
  + typeParall = "cluster")
```
> biohmm.mergel.ff.fork <- pSegmentBioHMM("cghData.RData",
+ "chromData.RData",
+ "posData.RData",
+ merging = "mergeLevels",
+ typeParall = "fork")
> biohmm.mad.ff.fork <- pSegmentBioHMM("cghData.RData",
+ "chromData.RData",
+ "posData.RData",
+ merging = "MAD",
+ typeParall = "fork")
> biohmm.mad.bic.ff.fork <- pSegmentBioHMM("cghData.RData",
+ "chromData.RData",
+ "posData.RData",
+ merging = "MAD",
+ aic.or.bic = "BIC",
+ typeParall = "fork")
> cghseg.mergel.ff.fork <- pSegmentCGHseg("cghData.RData",
+ "chromData.RData",
+ merging = "mergeLevels",
+ typeParall = "fork")
> cghseg.mad.ff.fork <- pSegmentCGHseg("cghData.RData",
+ "chromData.RData",
+ merging = "MAD",
+ typeParall = "fork")
> cghseg.none.ff.fork <- pSegmentCGHseg("cghData.RData",
+ "chromData.RData",
+ merging = "none", typeParall = "fork")
> waves.merge.ff.fork <- pSegmentWavelets("cghData.RData",
+ "chromData.RData",
+ merging = "mergeLevels",
+ typeParall = "fork")
> waves.mad.ff.fork <- pSegmentWavelets("cghData.RData",
+ "chromData.RData",
+ merging = "MAD",
+ typeParall = "fork")
> waves.none.ff.fork <- pSegmentWavelets("cghData.RData",
+ "chromData.RData",
+ merging = "none",
+ typeParall = "fork")

3.4 Comparing output

Here we verify that using different input and clustering methods does not change the results. Before carrying out the comparisons, however, we open the \texttt{ff} objects gently.

First, we will open the objects created above (same objects as were also created in the main vignette, in section “Carrying out segmentation and calling”). Instead of inserting many calls to each individual object, we open all available objects that match \texttt{ff.cluster}. To do that quickly we store the names of the objects

> ff.cluster.obj <- ls(pattern = "*.ff.cluster")
> tmpout <-
+ capture.output(
+ lapply(ff.cluster.obj, function(x) lapply(get(x), open))
+ )

We repeat that operation with the output from section 3.3:

> ff.fork.obj <- ls(pattern = "*.ff.fork")
> tmpout <-
+ capture.output(
+ lapply(ff.fork.obj, function(x) lapply(get(x), open))
+ )
>

And we create the list of results from the RAM and forking runs (no need for special
opening here, since these are not \texttt{ff} objects)

> RAM.fork.obj <- ls(pattern = "*.RAM.fork")

We can now compare the output. We want to compare the output from three different
methods, so we need to run three comparisons (this is what we did explicitly in the help for
\texttt{pSegment}). Since this is a very repetitive operation, we define a small utility function that
will return \texttt{TRUE} if both components (\texttt{outSmoothed} and \texttt{outState}) of all three objects are
identical. (Since the function will take as input not an actual object, but a name, we use
\texttt{get} inside the function.)

We use \texttt{all.equal} to compare the output from the smoothing, to allow for possible
numerical fuzz (that could result from differences in storage). When comparing the assigned
state, however, we check for exact identity.

> identical3 <- function(x, y, z) {
+ comp1 <- all.equal(get(x)$outSmoothed[, ], get(y)$outSmoothed[, ])
+ comp2 <- all.equal(get(y)$outSmoothed[, ], get(z)$outSmoothed[, ])
+ comp3 <- identical(get(x)$outState[, ], get(y)$outState[, ])
+ comp4 <- identical(get(y)$outState[, ], get(z)$outState[, ])
+ if (!all(isTRUE(comp1), isTRUE(comp2), comp3, comp4)) {
+ cat(paste("Comparing ", x, y, z, ", \"n",
+ "\n comp1 = ", paste(comp1, sep = " ", collapse = "\n "),
+ "\n comp2 = ", paste(comp2, sep = " ", collapse = "\n "),
+ "\n comp3 = ", paste(comp3, sep = " ", collapse = "\n "),
+ "\n comp4 = ", paste(comp4, sep = " ", collapse = "\n "),
+ "\n\n\n"))
+ return(FALSE)
+ } else {
+ TRUE
+ }
+ }

You should expect most (though not necessarily all) the comparisons to yield a \texttt{TRUE}. In
some cases, however, different runs of the same method might not yield the same results (e.g.,
CBS, HMM, etc). If you get non-identical results, you can try running those methods a few
times, to check for differences. You can also disable load balancing, and try using reproducible streams for the random number generators (see the vignette of package `parallel`).

Let’s check those results then:

```r
> mapply(identical3, RAM.fork.obj,
+       ff.fork.obj, ff.cluster.obj)
Comparing  cbs.mad.RAM.fork  cbs.mad.ff.fork  cbs.mad.ff.cluster
not equal: some info from comparisons.

  comp1 = TRUE
  comp2 = Component "m4": Mean relative difference: 0.07284734
  comp3 = TRUE
  comp4 = TRUE

Comparing  glad.RAM.fork  glad.ff.fork  glad.ff.cluster
not equal: some info from comparisons.

  comp1 = Component "m5": Mean relative difference: 0.1130491
          Component "L3": Mean relative difference: 0.6999325
  comp2 = Component "m5": Mean relative difference: 0.1051295
          Component "L3": Mean relative difference: 1.194193
  comp3 = FALSE
  comp4 = FALSE

Comparing  hmm.mad.RAM.fork  hmm.mad.ff.fork  hmm.mad.ff.cluster
not equal: some info from comparisons.

  comp1 = TRUE
  comp2 = Component "m5": Mean relative difference: 0.6976968
  comp3 = TRUE
  comp4 = TRUE

biohmm.mad.bic.RAM.fork  biohmm.mad.RAM.fork  biohmm.mergel.RAM.fork
 TRUE    TRUE    TRUE
  cbs.mad.RAM.fork  cbs.mergel.RAM.fork  cbs.none.RAM.fork
 FALSE    TRUE    TRUE
  cghseg.mad.RAM.fork  cghseg.mergel.RAM.fork  cghseg.none.RAM.fork
 TRUE    TRUE    TRUE
  glad.RAM.fork  hmm.mad.RAM.fork  hmm.mergel.RAM.fork
 FALSE    FALSE    TRUE
  hs.mad.RAM.fork  hs.mergel.RAM.fork  hs.none.RAM.fork
 TRUE    TRUE    TRUE
  waves.mad.RAM.fork  waves.mergel.RAM.fork  waves.none.RAM.fork
 TRUE    TRUE    TRUE

> (Of course, we depend on the lists of names of objects having the output from the same
method and option in the same position, which is the case in these examples).
4 Exercising the code for the load balancing options

This section simply exercises the load balancing options. We use Haar as it is the fastest method, and one unlikely to be affected by the order in which different columns are run (in contrast to, say, HMM), so we need not worry about random numbers here. (Note: sometimes, and only in some machines, the code that uses the cluster, not the forking, fails with a serialization error. I do not know the reason.)

```r
> hs.none.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+   merging = "none")
> hs.none.RAM.fork.lb <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+   merging = "none", loadBalance = TRUE)
> hs.none.RAM.fork.nlb <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+   merging = "none", loadBalance = FALSE)
> identical3("hs.none.RAM.fork", "hs.none.RAM.fork.lb", "hs.none.RAM.fork.nlb")

[1] TRUE
```

```r
> hs.none.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "cluster")
> hs.none.ff.cluster.lb <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "cluster",
+   loadBalance = TRUE)
> hs.none.ff.cluster.nlb <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "cluster",
+   loadBalance = FALSE)
> ## do not show all the opening ... messages
> tmpout <-
+   capture.output(
+       lapply("hs.none.ff.cluster", function(x) lapply(get(x), open))
+   )
> tmpout <-
+   capture.output(
+       lapply("hs.none.ff.cluster.lb", function(x) lapply(get(x), open))
+   )
> tmpout <-
+   capture.output(
+       lapply("hs.none.ff.cluster.nlb", function(x) lapply(get(x), open))
+   )
> identical3("hs.none.ff.cluster", "hs.none.ff.cluster.lb",
+   "hs.none.ff.cluster.nlb")

[1] TRUE
```

```r
> hs.none.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "fork")
> hs.none.ff.fork.lb <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "fork",
+   loadBalance = TRUE)
> hs.none.ff.fork.nlb <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+   merging = "none", typeParall = "fork",
+   loadBalance = FALSE)
> tmpout <-
```

22
capture.output(
+   lapply("hs.none.ff.fork", function(x) lapply(get(x), open))
+ )
> tmpout <-
+ capture.output(
+   lapply("hs.none.ff.fork.lb", function(x) lapply(get(x), open))
+ )
> tmpout <-
+ capture.output(
+   lapply("hs.none.ff.fork.nlb", function(x) lapply(get(x), open))
+ )
> identical3("hs.none.ff.fork", "hs.none.ff.fork.lb", "hs.none.ff.fork.nlb")

[1] TRUE

(There is no need to compare between ff.fork, ff.cluster, RAM.fork, as those were already shown to be identical.)

5 Clean up actions

These are not strictly necessary, but we will explicitly stop the cluster. In this vignette, we will not execute the code below to remove the directory we created or the objects, in case you want to check them out or play around with them, but the code is below. To make sure there are no file permission problems, we add code below to 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).

> parallel::stopCluster(cl2)

> ## This is the code to remove all the files we created
> ## and the temporary directory.
> ## We are not executing it!
>
> load("chromData.RData")
> load("posData.RData")
> load("cghData.RData")
> delete(cghData); rm(cghData)
> delete(posData); rm(posData)
> delete(chromData); rm(chromData)
> tmpout <-
+ capture.output(
+   lapply(ff.fork.obj, function(x) {
+     lapply(get(x), delete))})
> rm(list = ff.fork.obj)
> tmpout <-
+ capture.output(
+   lapply(ff.cluster.obj, function(x) {
+     lapply(get(x), delete))})
> rm(list = ff.cluster.obj)
> setwd(originalDir)
> print(getwd())
> Sys.sleep(3)
> unlink("ADaCGH2_vignette_tmp_dir", recursive = TRUE)
> Sys.sleep(3)