Description of the *affyPara* package: Parallelized preprocessing methods for Affymetrix Oligonucleotide Arrays

Markus Schmidberger [*†] Ulrich Mansmann[*]

October 28, 2009

Contents

1 Abstract **3**

2 Changes to previous Versions **3**

3 Introduction **3**
3.1 Requirements 4
3.2 Loading the package 4
3.3 Starting and stopping cluster 5
3.4 Inputdata: CEL Files or AffyBatch 5

4 Function Description **6**
4.1 Background Correction 6
4.1.1 Use Background Correction Para 7
4.2 Normalization 7
4.2.1 Use Quantile Normalization Para 8
4.3 Summarization 8
4.3.1 Use Summarization Para 9

[*]Division of Biometrics and Bioinformatics, IBE, University of Munich, 81377 Munich, Germany

[†]Package maintainer, Email: `schmidb@ibe.med.uni-muenchen.de`

4.4 Complete Preprocessing 9
4.4.1 Use Preprocessing Para 10
4.4.2 Use RMA Para 10
4.4.3 Use vsnRMA Para 11
4.5 Quality Control and Assessment 11
4.6 Distributing Data 11

5 Results and Discuccion **12**
5.1 Test for Accuracy 12
5.2 Speedup 13

1 Abstract

The *affyPara* package is part of the Bioconductor[1] [1] project. The package extends the *affy* package. The *affy* package is meant to be an extensible, interactive environment for data analysis and exploration of Affymetrix oligonucleotide array probe level data. For more details see the *affy* vignettes or [2].

The *affyPara* package contains parallelized preprocessing methods for high-density oligonucleotide microarray data. Partition of data could be done on arrays and therefore parallelization of algorithms gets intuitive possible. The partition of data and distribution to several nodes solves the main memory problems – caused by the `AffyBatch` object – and accelerates the methods [5].

This document was created using R version 2.10.0 and versions 1.24.0, 0.3-3 and 3.14.0 of the packages *affy*, *snow* and *vsn* respectively.

2 Changes to previous Versions

For major changes see the NEWS file in the source code of the package or use the function `readNEWS()`.

3 Introduction

The functions in the *affyPara* package have the same functionality and a very similar user-interface as the functions in the *affy* package. For a detailed function and method description see the *affy* vignettes and help files. The *affyPara* package contains parallelized preprocessing methods for high-density oligonucleotide microarray data.

The package is designed for large numbers of microarray data and solves the main memory problems caused by the `AffyBatch` object at only one workstation or processor. Partition of data could be done on arrays and therefore parallelization of algorithms gets intuitive possible. It is very difficult to define a concrete limit for a large number of data, because this strongly depends on the computer system (architecture, main memory, operating system). In general a computer cluster and the *affyPara* package should be

[1] `http://www.bioconductor.org/`

used when working with more than 150 microarrays. The partition of data and distribution to several nodes solves the main memory problems (at one workstation) and accelerates the methods. Parallelization of existing preprocessing methods produces, in view of machine accuracy, the same results as serialized methods.

3.1 Requirements

The *affyPara* package requires the *affy*, *snow* and *vsn* package. From the *affy* and *vsn* packages several subfunctions for preprocessing will be used. The *snow* package [4] will be used as interface to a communication mechanism for parallel computing. In the *snow* package four low level interfaces have been implemented, one using PVM via the *rpvm* package by Li and Rossini, one using MPI via the *Rmpi* [8] package by Hao Yu, one using NetWorkSpaces via the *NWS* package by Revolution Computing and one using raw sockets that may be useful if PVM and MPI are not available. For a comparison and review of existing parallel computing techniques with the R language see [6].

For more details concerning the *snow* package, see the help files or the webpage `http://www.cs.uiowa.edu/~luke/R/cluster/cluster.html`.

3.2 Loading the package

First of all you have to load the package and the depending packages.

```
> library(affyPara)
```

For demonstration we use the small `AffyBatch` object '`Dilution`' from the *affydata* package:

```
> library(affydata)
> data(Dilution)
> Dilution

AffyBatch object
size of arrays=640x640 features (35221 kb)
cdf=HG_U95Av2 (12625 affyids)
number of samples=4
number of genes=12625
annotation=hgu95av2
notes=
```

3.3 Starting and stopping cluster

After loading the libraries the computer cluster has to be initialized. Starting a workstation cluster is the only step in using a computer cluster that depends explicitly on the underlying communication mechanism. A cluster is started by calling the `makeCluster()` function, but the details of the call depend on the type of cluster. PVM, MPI or NWS cluster may also need some preliminary preparations to start the systems. For some examples see the webpage `http://www.cs.uiowa.edu/~luke/R/cluster/cluster.html`.

To start a cluster you should use

```
> cl <- makeCluster(4, type = "SOCK")
```

with the first parameter '4' for the number of spawend slaves and a parameter (type) for the used communication mechanism. In this example we use raw socket connections. As default in the *snow* a cluster object 'cl' will be created.

To stop a cluster you should use

```
> stopCluster(cl)
```

The *affyPara* package masks (overwrites) the functions `makeCluster()` and `stopCluster()` from the *snow* package. The new functions save the cluster object (cl) in the namespace environment. Therefore you do not have to deal with the 'cl' object in the function calls of the *affyPara* functions. But it is still possible, using the parameter `cluster=cl`.

Socket clusters should stop automatically, when the process – that created them – terminates. However, it is still a good idea to call `stopCluster()`.

For more details see the *snow* package, the R package for your communication mechanism (*Rmpi*, *Rpvm*, *NWS*), and the implementation of your communication mechanism.

3.4 Inputdata: CEL Files or AffyBatch

Before running any kind of proprocessing, the probe level data (CEL files) have to be handled. As suggested in the *affy* package an object of class `AffyBatch` can be created:

- Create a directory.
- Move all the relevant CEL files to that directory.

- Make sure your working directory contains the CEL files (`getwd()`, `setwd()`).
- Then read in the data:

```
> AffyBatch <- ReadAffy()
```

This `AffyBatch` object can be used to do preprocessing (with functions from the *affyPara* and *affy* package) on the data. Depending on the size of the dataset and on the memory available at the computer system, you might experience errors like 'Cannot allocate vector ...'.

The idea of the *affyPara* package is, that all probe level data will never be needed at one place (computer) at the same time. Therefore it is much more efficient and memory friendly to distribute the CEL files to the local disc of the slave computers or to a shared memory system (e.g. samba device). To build only small `AffyBatch` objects at the slaves, do preprocessing at the slaves and rebuild the results (`AffyBatch` or `ExpressionSet` object) at the master node. This process is implemented in the functions from the *affyPara* package.

4 Function Description

4.1 Background Correction

Background correction (BGC) methods are used to adjust intensities observed by means of image analysis to give an accurate measurement of specific hybridization. Therefore BGC is essential, since part of the measured probe intensities are due to non-specific hybridization and the noise in the optical detection system.

In the *affyPara* package the same BGC methods as in the *affy* package are available. To list the background correction methods – built into the package – the function `bgcorrect.method()` can be used:

```
> bgcorrect.methods()

[1] "bg.correct" "mas"        "none"       "rma"
```

4.1.1 Use Background Correction Para

The function `bgCorrectPara()` needs an input data object (Dilution) and the background correction method (method="rma") as input parameters.

```
> affyBatchBGC <- bgCorrectPara(Dilution, method = "rma", verbose = TRUE)

Partition of object 3.11 sec DONE
Object Distribution: 2 2
Initialize AffyBatches at slaves 3.851 sec DONE
BGC on Slaves 5.313 sec DONE
Rebuild AffyBatch 1.505 sec DONE
```

If you do not want to use an `AffyBatch` object as input data, you can directly give the CEL files and a vector of the CEL files location respectively to the function `bgCorrectPara()`:

```
> files <- list.celfiles(full.names = TRUE)
> affyBatchGBC <- bgCorrectPara(files, method = "rma", cluster = cl)
```

For this method all CEL files have to be available at every node. This could be achieved for example using a a shared memory system. If you want to distribute the CEL files to the slaves, see Chapter 4.6.

Additionally this example demonstrates how to use an extra cluster object (`cluster=cl`).

4.2 Normalization

Normalization methods make measurements from different arrays comparable. Multi-chip methods have proved to perform very well. We parallelized the following methods

`contrast` -> `normalizeAffyBatchConstantPara`

`invariantset` -> `normalizeAffyBatchInvariantsetPara`

`loess` -> `normalizeAffyBatchLoessPara`

`quantile` -> `normalizeAffyBatchQuantilesPara`

`vsn2` -> `vsnPara`

available from the *affy* package in the function `normalize()` and the *vsn* package.

The parallelized normalization functions need an input data object and the coresponding normalization parameters as input paramters.

4.2.1 Use Quantile Normalization Para

The function `normalizeAffyBatchQuantilesPara()` needs an input data object (Dilution) and quantil normalization parameters as input parameters (type = "pmonly").

```
> affyBatchNORM <- normalizeAffyBatchQuantilesPara(Dilution, type = "pmonly",
+     verbose = TRUE)

Partition of object 0.909 sec DONE
Object Distribution: 2 2
Initialize AffyBatches at slaves 0.827 sec DONE
PM normalization 4.049 sec DONE
Rebuild AffyBatch 1.004 sec DONE
```

If you do not want to use an `AffyBatch` object as input data, you can directly give the CEL files and a vector of the CEL files location respectively to the function `normalizeAffyBatchQuantilesPara()`:

```
> files <- list.celfiles(full.names = TRUE)
> affyBatchNORM <- normalizeAffyBatchQuantilesPara(files, type = "pmonly")
```

For this method all CEL files have to be available from a shared memory system. If you want to distribute the CEL files to the slaves, see Chapter 4.6.

4.3 Summarization

Summarization is the final step in preprocessing raw data. It combines the multiple probe intensities for each probeset to produce expression values. These values will be stored in the class called `ExpressionSet`. Compared to the `AffyBatch` class, the `ExpressionSet` requires much less main memory, because there are no more multiple data. Therefore the complete preprocessing functions in the *affyPara* package are very effizient, because no complete `AffyBatch` object has to be build, see Chapter 4.4.

The parallelized summarization functions need an input data object and the coresponding summarization parameters as input paramters. To see the summarization methods and PM correct methods that are built into the package the function `express.summary.stat.methods()` and `pmcorrect.methods()` can be used:

```
> express.summary.stat.methods()
[1] "avgdiff"           "liwong"            "mas"
[4] "medianpolish"      "playerout"         "medianpolish_orig"
[7] "liwong_orig"       "farms_orig"        "playerout_orig"
> pmcorrect.methods()
[1] "mas"        "methods"    "pmonly"     "subtractmm"
```

4.3.1 Use Summarization Para

The function `computeExprSetPara()` needs an input data object (Dilution) and the summarization parameters as input parameters (pmcorrect.method = "pmonly", summary.method = "avgdiff").

```
> esset <- computeExprSetPara(Dilution, pmcorrect.method = "pmonly",
+     summary.method = "avgdiff")
```

If you do not want to use an `AffyBatch` object as input data, you can directly give the CEL files and a vector of the CEL files location respectively to the function `computeExprSetPara()`:

```
> files <- list.celfiles(full.names = TRUE)
> esset <- normalizeAffyBatchQuantilesPara(files, pmcorrect.method = "pmonly",
+     summary.method = "avgdiff")
```

For this method all CEL files have to be available from a shared memory system. If you want to distribute the CEL files to the slaves, see Chapter 4.6.

4.4 Complete Preprocessing

By combining the background correction, normalization and summarization methods to one single method for preprocessing an efficient method can be obtained. For parallelization, the combination has the big advantage of reducing the exchange of data between master and slaves. Moreover, at no point a complete `AffyBatch` object needs to be built, and the time-consuming rebuilding of the AffyBatch objects is no longer necessary.

4.4.1 Use Preprocessing Para

It is important to note that not every preprocessing method can be combined together. For more details see the vignettes in the *affy* package.

The function `preproPara()` needs an input data object (Dilution) and the parameters for BGC, normalization and summarization as input parameters.

```
> esset <- preproPara(Dilution, bgcorrect = TRUE, bgcorrect.method = "rma",
+     normalize = TRUE, normalize.method = "quantil", pmcorrect.method = "pmonly
+     summary.method = "avgdiff")
```

The function works very similar to the `expresso()` function from the *affy* package. It is not very reasonable to have an `AffyBatch` object as input data object for this function. Because therefore you have to create a complete `AffyBatch` object. (very memory intensive). It is much better to use a vector of CEL files as input data object. And at no point a complete `AffyBatch` object needs to be built:

```
> files <- list.celfiles(full.names = TRUE)
> esset <- preproPara(files, bgcorrect = TRUE, bgcorrect.method = "rma",
+     normalize = TRUE, normalize.method = "quantil", pmcorrect.method = "pmonly
+     summary.method = "avgdiff")
```

For this method all CEL files have to be available from a shared memory system. If you want to distribute the CEL files to the slaves, see Chapter 4.6.

4.4.2 Use RMA Para

RMA is a famous [3] complete preprocessing method. This function converts an `AffyBatch` object into an `ExpressionSet` object using the robust multi-array average (RMA) expression measure. There exists a function `justRMA()` in the *affy* package, which reads CEL files and computes an expression measure without using an `AffyBatch` object.

The parallelized version of `rma()` is called `rmaPara()` and is a 'simple' wrapper function for the function `preproPara()`.

```
> esset <- rmaPara(Dilution)
```

It is not very reasonable to have an `AffyBatch` object as input data object for this function. Because therefore you have to create a complete `AffyBatch` object (very memory intensive).

It is much better to use a vector of CEL files as input data object. And at no point a complete `AffyBatch` object needs to be built:

```
> files <- list.celfiles(full.names = TRUE)
> esset <- rmaPara(files)
```

For this method all CEL files have to be available from a shared memory system. If you want to distribute the CEL files to the slaves, see Chapter 4.6.

4.4.3 Use vsnRMA Para

An other famous preprocessing method is `vsnrma()`. This uses varianze stabilization normalization for normalization and rma for summarization. The parallelized version is called `vsnrmaPara()`.

4.5 Quality Control and Assessment

Quality control and assessment gets very difficult for a huge number of microarrays data. It takes a lot of computation time and it requires a lot of main memory. In addition most methods for quality control are graphical tools and using more than 200 arrays no more information can be readout of the figures.

Therefore some optimized functions for quality control were implemented in parallel:

```
> boxplotPara(Dilution)
> MAplotPara(Dilution)
```

These functions create output tables with the quality assessment for all arrays and create optimized graphics (`plot=TRUE`) for huge numbers of arrays.

4.6 Distributing Data

At a workstation cluster the CEL files could be available by a shared memory system. At a workstation cluster, this is often done by a samba device. But this could be the bottle neck for communication traffic. For distributed memory systems, the function `distributeFiles()` for (hierarchically) distributing files from the master to a special directory (e.g. '/tmp/') at all slaves was designed. R or the faster network protocols SCP or RCP can be used for the process of distributing.

```
> path <- "tmp/CELfiles"
> files <- list.files(path, full.names = TRUE)
> distList <- distributeFiles(CELfiles, protocol = "RCP")
> eset <- rmaPara(distList$CELfiles)
```

With the paramteter `hierarchicallyDist` hierarchically distribution could be used. If `hierarchicallyDist = TRUE` data will be hierarchically distributed to all slaves. If `hierarchicallyDist = FALSE` at every slave only a part of data is available. This function and the corresponding input data object (`distList$CELfiles`) could be used for every parallelized preprocessing method in the *affyPara* package.

There is also a function to remove distributed files:

```
> removeDistributedFiles("/usr1/tmp/CELfiles")
```

5 Results and Discuccion

This article proposes the new package called *affyPara* for parallelized preprocessing of high-density oligonucleotide microarrays. Parallelization of existing preprocessing methods produces, in view of machine accuracy, the same results as serialized methods. The partition of data and distribution to several nodes solves the main memory problems and accelerates the methods.

5.1 Test for Accuracy

In view of machine accuracy, the parallelized functions produce same results as serialized methods. To compare results from different functions you can use the functions `identical()` or `all.equal()` from the *base* package.

```
> affybatch1 <- bg.correct(Dilution, method = "rma")
> affybatch2 <- bgCorrectPara(Dilution, method = "rma")
> identical(exprs(affybatch1), exprs(affybatch2))

[1] TRUE

> all.equal(exprs(affybatch1), exprs(affybatch2))

[1] TRUE
```

Attention: If you directly compare the `AffyBatch` or `ExpressionSet` objects there are some warnings or not similar results. This is being caused by different values of the 'Title' and 'notes' slots in experimentData. Using the function `exprs()` to get the expression data equal results – in view of machine accuracy – can be proven.

Attention for loess normalization: In loess normalization a random sub sample will be created. For generating the same results the random generator has to be reset for every run:

```
> set.seed(1234)
> affybatch1 <- normalize.AffyBatch.loess(Dilution)
> set.seed(1234)
> affybatch2 <- normalizeAffyBatchLoessPara(Dilution, verbose = TRUE)
> identical(exprs(affybatch1), exprs(affybatch2))
```

5.2 Speedup

In order to illustrate by how much the parallel algorithms are faster than the corresponding sequential algorithms, Figure 1 shows the speedup for the parallelized preprocessing methods for 50, 100 and 200 CEL files. An average speedup of up to the factor 10 for 200 arrays or more may be achieved.

The computation time for parallel algorithms is compared to the original serial code. It is well known that parts of the original code are not very well implemented. Therefore an increased speedup (super-linear) could be achieved for low numbers of processors, the outliers are mostly generated by unbalanced data distribution. For example 200 microarrays can not be equally distributed to 23 nodes, there are some computers who have to calculate with one more array. Furthermore foreign network traffic in the workstation cluster at the IBE is a reason for outliers. After a special number of processors (depending on number of arrays and method) the plots for all parallelized function get a flat. This means, by using more processors no more speedup could be achieved. Therefore for example for 200 microrrays circa 10 nodes will be enough.

The cluster at the Department for Medical Information, Biometrics and Epidemiology (IBE, University of Munich) consists of 32 personal computers with 8 GB main memory and two dual core Intel Xeon DP 5150 processors. Using this cluster, about 16.000 (32 nodes $\cdot$ approximately 500 CEL files) microarrays of the type HGU-133A can be preprocessed using the function

`preproPara()`. By expanding the cluster, the number of microarrays can be increased to any given number.

The *affyPara* package is tested in different hardware environments:

Computer Cluster: IBE, Linux-Cluster(LRZ, Munich, Germany), HLRB2(LRZ, Munich, Germany), Hoppy (FHCRC, Seattle, WA, USA)

Multicore: IBE, HLRB2(LRZ, Munich, Germany), lamprey (FHCRC, Seattle, WA, USA)

Thanks a lot to the institutes for providing access to their computer ressources.

```
> stopCluster()
```

References

[1] Robert C. Gentleman, Vincent J. Carey, Douglas M. Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, Laurent Gautier, Yongchao Ge, Jeff Gentry, Kurt Hornik, Torsten Hothorn, Wolfgang Huber, Stefano Iacus, Rafael Irizarry, Friedrich Leisch, Cheng Li, Martin Maechler, Anthony J. Rossini, Gunther Sawitzki, Colin Smith, Gordon Smyth, Luke Tierney, Jean Y. H. Yang, and Jianhua Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[2] R. Irizarry, L. Gautier, and L. Cope. An r package for analyses of affymetrix oligonucleotide arrays. In G. Parmigiani, E.S. Garrett, R.A. Irizarry, and S.L. Zeger, editors, *The Analysis of Gene Expression Data: Methods and Software*. Springer, New York, 2002.

[3] Rafael A Irizarry, Bridget Hobbs, Francois Collin, Yasmin D Beazer-Barclay, Kristen J Antonellis, Uwe Scherf, and Terence P Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, Apr 2003.

[4] Anthony Rossini. Simple parallel statistical computing in r. *UW Biostatistics Working Paper Series*, 193, 2003.

[5] Markus Schmidberger and Ulrich Mansmann. Parallelized preprocessing algorithms for high-density oligonucleotide array data. In *22th International Parallel and Distributed Processing Symposium (IPDPS 2008)*, 2008.

[6] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, and Ulrich Mansmann. State-of-the-art in parallel computing with r. 47, 1 2009.

[7] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[8] Hao Yu. *The Rmpi Package*. Department of Statistical and Actuarial Sciences; University of Western Ontario, London, Ontario N6A 5B7, 04 2004.

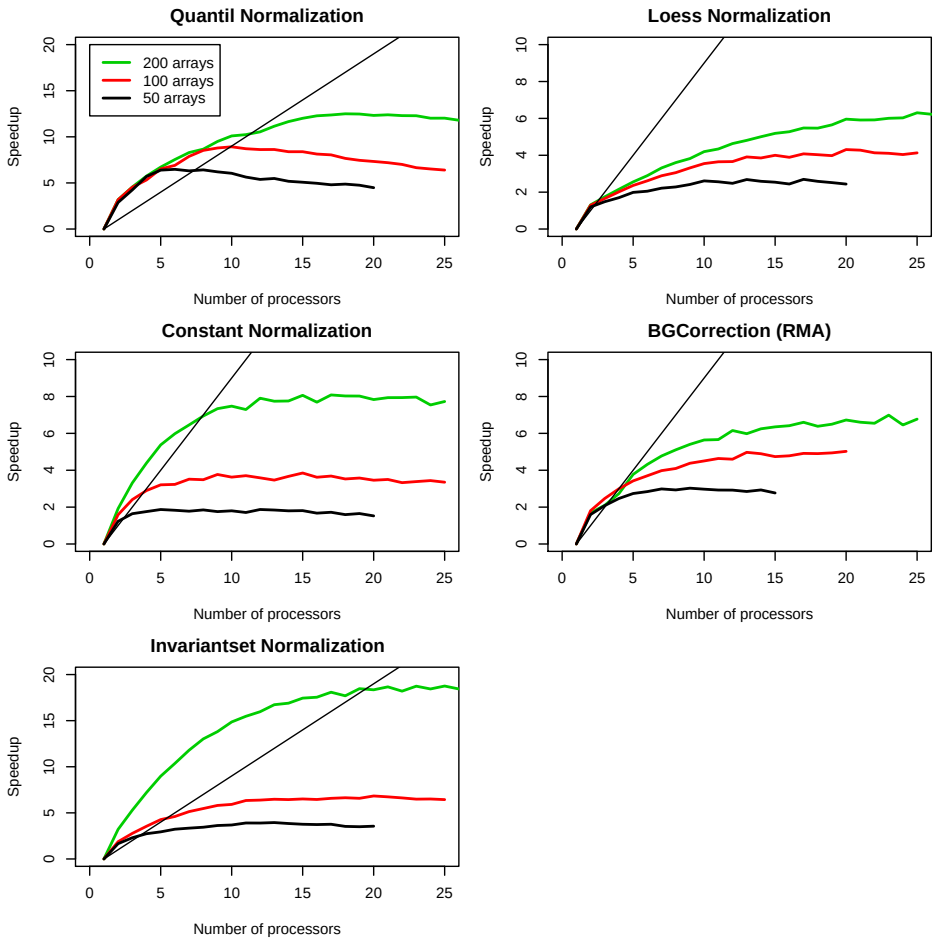

Figure 1: Speedup for the parallelized preprocessing methods for 200, 100 and 50 microarrays.