Description of the \textit{affyPara} package:  
Parallelized preprocessing methods for  
Affymetrix Oligonucleotide Arrays  

Markus Schmidberger    Ulrich Mansmann

April 30, 2008

Contents

1 Abstract 3

2 Changes to previous Versions 3
  2.1 Version 0.99.2 3
  2.2 Version 0.99.0 3
  2.3 Version 0.3.0 3
  2.4 Version 0.2.0 4
  2.5 Version 0.1.0 4

3 Introduction 4
  3.1 Requirements 4
  3.2 Loading of package 5
  3.3 Starting and stopping cluster 5
  3.4 Inputdata: CEL Files or AffyBatch 5
  3.5 Background Correction 6  
  3.5.1 Use Background Correction Para 7
  3.6 Normalization 7  
  3.6.1 Use Quantile Normalization Para 7
  3.7 Summarization 8  
  3.7.1 Use Summarization Para 8
  3.8 Complete Preprocessing 9
3.8.1 Use Preprocessing Para ........................................... 9
3.8.2 Use RMA Para ....................................................... 10
3.9 Distributing data ....................................................... 11
3.10 Test if results are equal to serialized methods .............. 11

4 Results and discussion ................................................. 12
1 Abstract

The affyPara package is part of the Bioconductor project. The affyPara 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 and accelerates the methods [5].

This document was created using R version 2.7.0 and versions 1.18.0 and 0.2-9 of the packages affy and snow respectively.

2 Changes to previous Versions

2.1 Version 0.99.2

- Compilation error in vignette removed
- Vignette extended
- Object.type as input removed from all functions

2.2 Version 0.99.0

- Finalizing for BioConductor release.
- Code cleaning.

2.3 Version 0.3.0

- Documentation added.
- Function rmaPara added.
- Modularization.

2.4 Version 0.2.0

- Testcode implemented and code testing -> several small bugs fixed

2.5 Version 0.1.0

- Implementation / parallelization of functions ...
  - bgCorrectPara
  - normalizeAffyBatchConstantPara, normalizeAffyBatchInvariantsetPara, normalizeAffyBatchQuantilesPara
  - computeExprSetPara
  - preproPara
  - distributeFiles, removeDistributedFiles

3 Introduction

The functions in the `affyPara` package have the same functionality then the functions in the `affy` package. For details see the `affy` vignettes. 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` at only one workstation or processor. 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). A computer cluster and the `affyPara` package should be used when working with more than 200 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` and `snow` package. From the `affy` package 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 three low level interfaces have been implemented, one using PVM via the *rpvm* package by Li and Rossini, one using MPI via the *Rmpi* package by Hao Yu, and one using raw sockets that may be useful if PVM and MPI are not available. For more details see the literature or the webpage http://www.cs.uiowa.edu/~luke/R/cluster/cluster.html.

### 3.2 Loading of package

The first thing you need to do is load the package.

```
R> library(affyPara)
```

### 3.3 Starting and stopping cluster

After loading the library the computer cluster has to be started. Starting a workstation cluster is the only step in using a cluster that depends explicitly on the underlying communication mechanism. A cluster is started by calling the `makeCluster` function, but the details of the call depending on the type of cluster. PVM and MPI clusters 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

```
R> c1 <- makeCluster(10)
```

with a parameter (10) for the number of spawnend slaves.

To stop a cluster you should use

```
R> stopCluster(c1)
```

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*) and the implementation of your communication mechanism.

### 3.4 Inputdata: CEL Files or AffyBatch

Before doing any kind of preprocessing 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:

```R
> affybatch.example <- ReadAffy()
```

This **AffyBatch** 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). Then to build only small **AffyBatches** at the slaves, do preprocessing at the slaves and rebuild the results (**AffyBatch** or **ExpressionSet**) at the master node. This could be done using the functions from the **affyPara** package.

### 3.5 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 see the background correction methods that are built into the package the variable `bgcorrect.method` can be used:

```R
> bgcorrect.methods

[1] "mas"   "none"  "rma"
```
3.5.1 Use Background Correction Para

The function `bgCorrectPara` needs a cluster object (`c1`), an input data object (`affybatch.example`) and the background correction method (`method="rma"`) as input parameters.

\[
R> \text{affyBatchGBC} \leftarrow \text{bgCorrectPara}(c1, \text{affybatch.example}, \text{method="rma"})
\]

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

\[
R> \text{files} \leftarrow \text{list.celfiles(full.names=TRUE)}
R> \text{affyBatchGBC} \leftarrow \text{bgCorrectPara}(c1, \text{files}, \text{method="rma"})
\]

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 3.9.

3.6 Normalization

Normalization methods make measurements from different arrays comparable. Multi-chip methods have proved to perform very well. We parallelized the methods `contrast` (\texttt{normalizeAffyBatchConstantPara}), `invariantset` (\texttt{normalizeAffyBatchInvariantsetPara}) and `quantile` (\texttt{normalizeAffyBatchQuantilesPara}) available from the `affy` package in the function `normalize`.

The parallelized normalization functions need a cluster object, an input data object and the corresponding normalization parameters as input parameters.

3.6.1 Use Quantile Normalization Para

The function `normalizeAffyBatchQuantilesPara` needs a cluster object (`c1`), an input data object (`affybatch.example`) and quantile normalization parameters as input parameters (type = "pmonly").

\[
R> \text{affyBatchNORM} \leftarrow \text{normalizeAffyBatchQuantilesPara}(c1, \text{affybatch.example, type = "pmonly"})
\]
If you do not want to use an AffyBatch as input data, you can directly give the CEL files and a vector of the CEL files location respectively to the function normalizeAffyBatchQuantilesPara:

```r
R> files <- list.celfiles(full.names=TRUE)
R> affyBatchNORM <- normalizeAffyBatchQuantilesPara(c1, 
    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 3.9.

### 3.7 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 are very efficient, because no complete AffyBatch has to be build, see chapter 3.8.

The parallelized summarization functions need a cluster object, an input data object and the corresponding summarization parameters as input parameters. To see the summarization methods and PM correct methods that are built into the package the variable `express.summary.stat.methods` and `pmcorrect.methods` can be used:

```r
> express.summary.stat.methods
[1] "avdiff" "liwong" "mas" "medianpolish" "playerout"

> pmcorrect.methods
[1] "mas" "pmonly" "subtractmm"
```

#### 3.7.1 Use Summarization Para

The function `computeExprSetPara` needs a cluster object (c1), an input data object (affybatch.example) and the summarization parameters as input parameters (pmcorrect.method = "pmonly", summary.method = "avdiff").
If you do not want to use an `AffyBatch` as input data, you can directly give the CEL files and a vector of the CEL files location respectively to the function `computeExprSetPara`:

```r
R> files <- list.celfiles(full.names=TRUE)
R> eset <- normalizeAffyBatchQuantilesPara(c1, 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 3.9.

### 3.8 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` needs to be built, and the time-consuming rebuilding of the Affy-Batches is no longer necessary.

#### 3.8.1 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 `affyPara` package.

The function `preproPara` needs a cluster object (c1), an input data object (affybatch.example) and the parameters for BGC, normalization and summarization as input parameters.

```r
R> eset <- preproPara(c1, affybatch.example, bgcorrect = TRUE, bgcorrect.method = "rma2", 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` as input data object for this function. Because therefore you have to create a complete `AffyBatch` (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` needs to be built:

```r
R> files <- list.celfiles(full.names=TRUE)
R> esset <- preproPara(c1, 
  files, 
  bgcorrect = TRUE, bgcorrect.method = "rma2", 
  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 [3.9](#).

### 3.8.2 Use RMA Para

RMA is a famous [3] complete preprocessing method. This function converts an `AffyBatch` into an `ExpressionSet` 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`.

The parallelized version of `rma` is called `rmaPara` and is a ‘simple’ wrapper function for the function `preproPara`.

```r
R> esset <- rmaPara(c1, affybatch.example)
```

It is not very reasonable to have an `AffyBatch` as input data object for this function. Because therefore you have to create a complete `AffyBatch` (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` needs to be built:

```r
R> files <- list.celfiles(full.names=TRUE)
R> esset <- rmaPara(c1, 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 [3.9](#).
3.9 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.

```r
R> path <- "tmp/CELfiles" # path at local computer system (master)
R> files <- list.files(path,full.names=TRUE)
R> distList <- distributeFiles(c1, CELfiles, protocol="RCP")
R> eset <- rmaPara(c1, distList$CELfiles)
```

With the parameter `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:

```r
R> removeDistributedFiles(c1, "/usr1/tmp/CELfiles")
```

3.10 Test if results are equal to serialized methods

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.

```r
R> affybatch1 <- bg.correct(affybatch.example,
   method="rma")
R> affybatch2 <- bgCorrectPara(c1, affybatch.example,
   method="rma")
R> identical(exprs(affybatch1),exprs(affybatch2))
[1] TRUE
R> all.equal(exprs(affybatch1),exprs(affybatch2))
[1] TRUE
```
Attention: If you directly compare the AffyBatches or ExpressionSets 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 shows equal results in view of machine accuracy.

4 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.

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.

References


