Parallelized preprocessing algorithms for high-density oligonucleotide arrays

Markus Schmidberger, Ulrich Mansmann
Chair of Biometrics and Bioinformatics, IBE
University of Munich
81377 Munich, Germany
{schmidb,mansmann}@ibe.med.uni-muenchen.de

Abstract

Studies of gene expression using high-density oligonucleotide microarrays have become standard in a variety of biological contexts. The data recorded using the microarray technique are characterized by high levels of noise and bias. These failures have to be removed, therefore preprocessing of raw data has been a research topic of high priority over the past few years.

Actual research and computations are limited by the available computer hardware. Furthermore most of the existing preprocessing methods are very time consuming. To solve these problems, the potential of parallel computing should be used. For parallelization on multicomputers, the communication protocol MPI (Message Passing Interface) and the R language will be used.

This paper proposes the new R language package `affyPara` *for parallelized preprocessing of 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 and accelerates the methods by up to the factor ten.*

1. Introduction

Studies of gene expression using high-density oligonucleotide microarrays have become standard in a variety of biological contexts. They enable scientists to investigate the functional relationship between the cellular and physiological processes of biological organisms and their genes at genome-wide system levels.

For this purpose, a variety of microarray technology platforms are used. The most popular microarray application is based on measuring genome-wide expression levels. High-density oligonucleotide expression microarrays are commonly used for this purpose. There are a lot of chips made by different manufacturers. Affymetrix GeneChip© arrays dominate this market.

The data recorded by means of the microarray technique are characterized by high levels of noise induced by the preparation, hybridization and measurement processes. These failures have to be removed, therefore preprocessing of raw-data has been a research topic of high priority over the past few years. Preprocessing usually involves three steps: Background correction, normalization and summarization. For more details and a brief introduction see e.g. [4].

The open source projects R [15] and Bioconductor [5] are tools in the fields of computational biology and bioinformatics. R is a free software environment and provides a wide range of statistical and graphical techniques and is highly extensible. Bioconductor is an open source and open development software project for the analysis and comprehension of genomic data. Bioconductor is primarily based on the R programming language. Both support the rapid developments in microarray technologies. For each step of preprocessing, a large number of methods are implemented and stored in the `affy` package [8].

2. Problems

Actual research and computations are limited by the available computer hardware. For many users the available main memory - mostly 1 GB at a workstation - limits the number of arrays that may be quantified. Furthermore, most of the existing preprocessing methods are very time consuming and thus not useful for first and fast checks in laboratories.

2.1. Memory limits

The main memory limits are caused by the structure of the *AffyBatch* class. The *AffyBatch* will be created by importing CEL files (specially coded ASCII files containing fluorescence intensities for each probe on the microarray)

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

Table 1. Maximum number of CEL files HGU-133A for creating an *AffyBatch* at different computer systems.

System	max. CEL files
64-bit linux system with 4 GB main memory	400
32-bit linux system with 4 GB main memory	160
32-bit Microsoft Windows XP system with 1 GB main memory	60

into the R software and is a container for storing probe-level data, related phenotypic information and MIAME (Minimum Information About a Microarray Experiment). The number of arrays which can be imported strongly depends on the architecture of the computer system. Table 1 shows some maximum numbers of CEL files HGU-133A which can be used for creating an *AffyBatch* at different computer systems. The same machine can yield results differing by up to 5%. Because of the different amount of 'outliers' the CEL file size can change by up to 3-5%. Using more arrays, a segmentation fault occurs in the R function `ReadAffy`. Differences in system architecture and configuration render any prediction of the maximum number of CEL files impossible. Calculations like preprocessing methods on the *AffyBatch* require more main memory, which means that the amount of usable arrays will be smaller (see Table 2).

2.2. Computation time

Computation time is the time a computer needs to complete a program. This closely depends on the speed of the computer processor. Different measurements show a nearly linear relation between computation time and the amount of arrays. The gradient depends on the kind of method used and the speed of the processor. Figure 1 shows the computation time in relation to the amount of CEL files for several preprocessing methods. The time measurements were done at one node of the workstation cluster described in section 4.

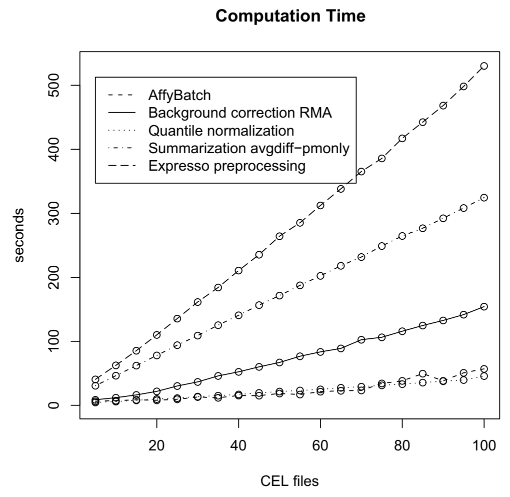

Figure 1. Computation time in relation to the amount of CEL files for several preprocessing methods. expresso(..., bgcorrect.method='rma', normalize.method='quantiles', pmcorrect.method='pmonly', summary.method='avgdiff'); computeExprSet(..., pmcorrect.method='pmonly', summary.method='avgdiff')

2.3. Existing solutions

There is a number of preprocessing methods included in the `affy` and `affyPLM` packages which try to solve these problems. For example, `justRMA` reads CEL files directly in the working directory and converts the raw data - without using an *AffyBatch* - to an expression measure using robust multi-array average (RMA)[9]. The `threestep` function [1] is primarily implemented in C code and is typically faster than `expresso` or `RMA`. It is an alternative method of computing expression measures using the three described preprocessing steps. Table 2 shows the improvements which may be achieved for HGU-133A chips. The results were calculated at one node of the workstation cluster described in section 4.

Unfortunately there are only a few methods available which are designed for fast computations on a large amount of data.

2.4. Further challenges

A further challenge is the fact that microarray experiments are becoming increasingly popular. The large number of publications with the keyword 'microarray' published in PubMed shows the rapid development in this field during the last ten years (see Figure 2). In addition, microarray chips are becoming cheaper and the number of chips used in experiments is growing. The technology for creating chips is improving daily and the number of probes per chip is growing too. Therefore, more and more data have to be managed and processed. Fig-

Table 2. Improvements by special methods (system specific).

	100 CEL files	150 CEL files	200 CEL files
`expresso`	9.3 min	29.6 min	segmentation fault
`threestep`	0.8 min	1.2 min	1.6 min

	max. CEL files
`rma`	250
`justrma`	more than 1500

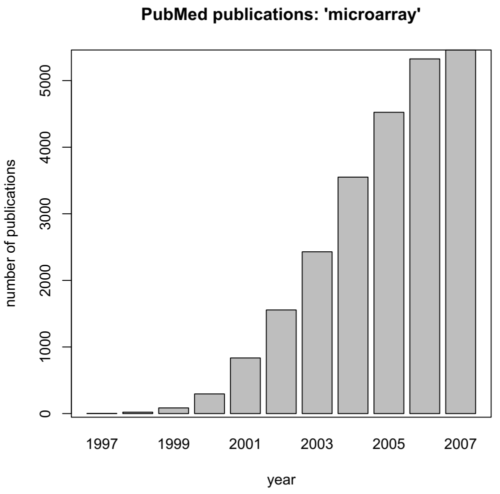

Figure 2. Publications with the keyword 'microarray' published in PubMed between 1997 and 2007.

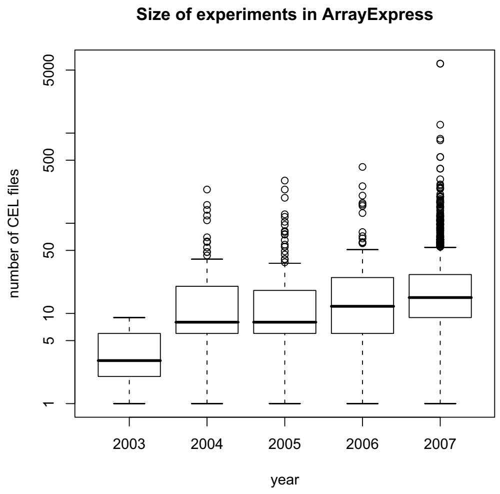

Figure 3. Size of experiments published in ArrayExpress between 1997 and 2007.

ure 3 shows the box-and-whisker diagrams[1] for the size of experiments published in the database ArrayExpress [11] between 2003 and 2007. There is only a slight increase in the mean size of experiments; however, the number of big experiments is growing. For instance, in 2007 the EMBLs European Bioinformatics Institute (EMBL-EBI) (Cambridge, UK) launched an experiment containing 5896 CEL files (Affymetrix HG-U133A) to create a human gene expression atlas [10], [11, E-TABM-185].

3. Solutions

These problems could be solved by using faster computer processors and bigger main memories (e.g. 128 GB). But as microarray experiments are becoming increasingly popular, buying a better computer can only be a temporary solution.

3.1. Parallel computing

Another option would be using the potential of parallel computing. In microarray technologies and statistical computing, parallel computing does not appear to have been used extensively up to now [13]. For parallelization on multicomputers (distributed memory systems), message passing methods are mostly used. In the R language, basic libraries like the `Rmpi`[16] and `Snow` [12] packages are still available for parallelization on process layer.

MPI (Message Passing Interface) is a well-known language-independent communication protocol used to program parallel computers. Message-passing systems are used especially on distributed machines with a separate memory for executing parallel applications. Each executing process will communicate and share its data with others by sending and receiving messages. MPI's goals are high

[1]In descriptive statistics, a boxplot (also known as a box-and-whisker diagram) is a convenient way of graphically depicting groups of numerical data through their summaries: smallest observation, outliers (circles), whiskers (extend to the most extreme data point which is no more than 1.5 times the interquartile range from the box), lower quartile, median, upper quartile, and largest observation.

performance, scalability and portability. MPI is currently available in version 2; several implementations (LAM/MPI [14], Open MPI [3], MPICH2, ...) exist and are becoming more and more a de facto standard. For the R language the `Rmpi` package is a wrapper for LAM/MPI and MPICH2. The `Snow` package (Simple network of workstations) is a simple supporting framework for doing parallel computing with the R programming language. `Snow` is a wrapper for the low-level communication mechanisms: sockets, MPI and PVM.

All these approaches are based on the master-slave design. A master process creates an universum of slave-processes, which perform computation on demand of their master.

3.2. Partition

For parallelization, the input data have to be partitioned. The easiest and most natural way of doing this is to carry out a partition on arrays and to distribute the arrays equally to all nodes. Partition on probes and other partition strategies have not yet been tested or implemented. In this context, a new data structure in the R programming language has to be developed and all preprocessing functions have to be reimplemented.

However, partitioning the *AffyBatch* at the master and distributing split *AffyBatches* to the slaves does not solve the problem of limited main memory. It creates a lot of network traffic, and therefore the processes start at the slaves with a certain delay. It is more efficient to partition the vector of CEL files and to create the split *AffyBatches* at the slaves. In the `affyPara` package all functions for preprocessing can get a partitioned list of CEL files or an *AffyBatch* as input data. For background correction, in Figure 4, the computation time for different numbers of CEL files is compared to an *AffyBatch* and a partitioned list of CEL files as input data. Doing this, the typical and expected logarithmic graphs are obtained. Computation time is similar for both types of input data, but using an *AffyBatch* makes the process more than 20% slower than using a partitioned list of CEL files as input data. This is mainly caused by the amount of network traffic for sending the *AffyBatches* to the slaves. The computation time for different input data and different numbers of microarrays is shown in table 3.

At the used workstation cluster, the CEL files are made 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 protocol RCP can be used for the process of distributing.

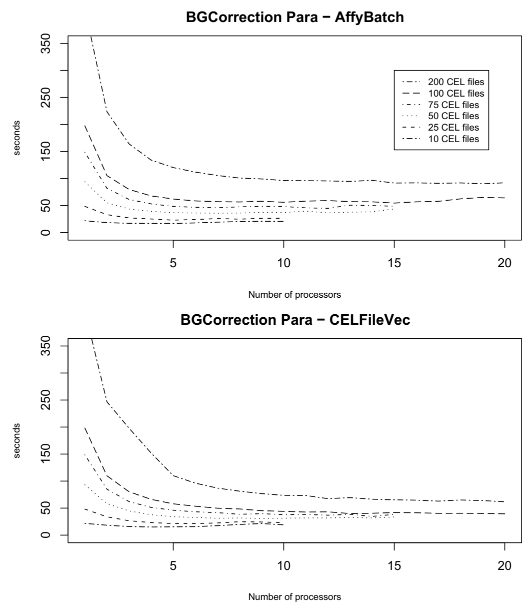

Figure 4. Computation time for parallelized background correction (*rma*) methods with an AffyBatch and a partitioned list of CEL files (CELFileVec) as input data.

Table 3. Tabularly computation time for parallelized background correction (*rma*) methods with an AffyBatch and a partitioned list of CEL files (CELFileVec) as input data.

Arrays	50	100	200
Processors	9	11	17
AffyBatch	37.9 sec	58.1 sec	91.1 sec
CELFileVec	30.7 sec	42.8 sec	62.9 sec

The size of the split *AffyBatches* and the number of parts is essential for the performance of the parallelized methods. In section 4 the partition will be discussed.

3.3. 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. The BGC methods *RMA*, *MAS 5.0* and *Ideal Mismatch* are implemented in the function `bg.correction` in the `affy` package. These methods are dependent on the actual sample only, and it is easy to parallelize them: Partition of the data, initialization of the *AffyBatch* at slaves, calling at slaves BGC methods at the parts of *AffyBatches*, sending results back to master and rebuilding the *AffyBatch*. Figure 5 shows the programming

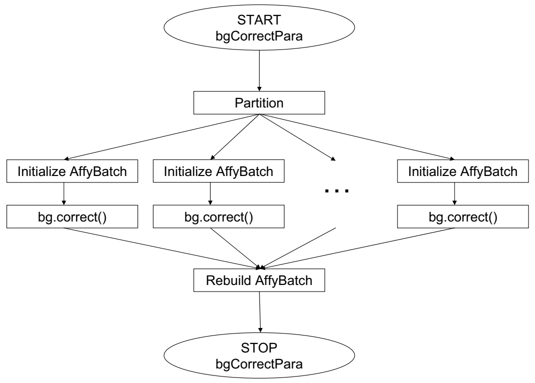

Figure 5. Flowchart for parallelized background correction methods.

flowchart of the parallelized background correction function. Parallelized BGC methods are available in the function `bgCorrectPara`.

3.4. Normalization

Normalization methods make measurements from different arrays comparable. Multi-chip methods have proved to perform very well. We parallelized the methods *contrast*, *invariantset* and *quantile* available from the `affy` package in the function `normalize`. For these parallelized functions, first of all the data have to be distributed to the slaves, and some model parameters have to be calculated. Sending these parameters back to the master, the parameters for the whole normalization model can be computed. Back at the slaves the normalization can be done with the complete parameters. Finally the *AffyBatch* has to be rebuilt at the master. Figure 6 shows the programming flowchart for the parallelized quantile normalization function. Parallelized normalization methods are available in the functions `normalizeAffyBatchConstantPara`, `normalizeAffyBatchInvariantsetPara` and `normalizeAffyBatchQuantilesPara`.

3.5. 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. In the

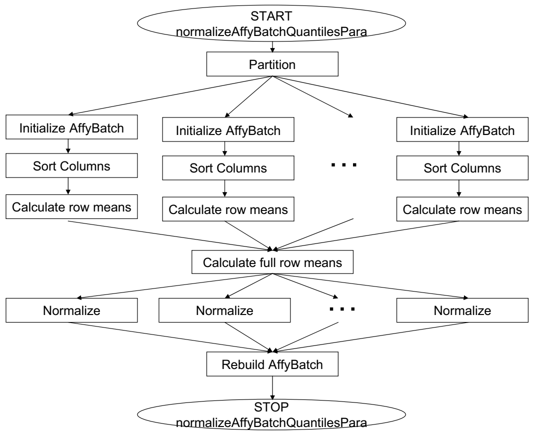

Figure 6. Flowchart for parallelized quantile normalization.

parallelized method, the required probes will be collected from the slaves, and by means of the standard summarization methods (*avgdiff*, *liwong*, *mas*, *medianpolish*), one expression value will be calculated. Parallelized summarization methods are available in `computeExprSetPara`.

3.6. preproPara

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 *AffyBatches* is no longer necessary. A parallelized complete preprocessing method is available in the function called `preproPara`. This function can compute the preprocessing steps proposed in sections 3.3, 3.4 and 3.5.

For more details on the functions, see the help or vignettes of the `affyPara` package.

4. Results and discussion

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.

All examples and results are computed on a workstation cluster with a 64 bit linux system, LAM/MPI (Version 7.1.3), R 2.5.0 and the package `Snow` (Version 0.2-9, used the package Rmpi 0.5-5) and affy (Version 1.14.2). The cluster consists of 32 personal computers with 8 GB main memory and two dual core Intel Xeon DP 5150 processors. Using this cluster at the Department for Medical Information, Biometrics and Epidemiology (IBE, University of Munich), about 16.000 (32 nodes · 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.

In the published results, only one processor per node was used because the R software has not as yet been implemented for multiprocessors. The attempt of extending the R language to this task is currently being made. This can be supported by using libraries such as openMP [2]. However, parallelization using multiprocessors (shared memory) will not solve the mentioned main memory problems. Using the power of multiprocessors for accelerating methods in R packages will be tested in the near future.

There will be a new limitation imposed by the memory size of the *ExpressionSet* class and the analysis to be done on the new big size of expression sets. However, there is currently no dataset on the market that would cause main memory problems with the *ExpressionSet* class.

4.1. Choice of partition and efficiency

It is very difficult to choose the right size and number of partitions. Figure 7 shows the computation time for parallelized normalization methods, Figure 4 the computation time for a parallelized background correction method depending on the partition size. The methods and data are distributed to n processors, which means that an improvement of the factor n should be achievable. However, due to not optimal load balancing and communication overhead, this can generally not be achieved. For BGC and normalization, the last step is the rebuilding of the whole *AffyBatch* out of a list of *AffyBatches*. This is a time-consuming process, which can take up to 35% of the whole computation time. For this reason and because of communication overhead, the computation time for many partitions is growing. Having too big partitions (only a few nodes), the computation time at the slaves is longer and also the communication traffic is higher. This means that, for partitions of big size, the computation time is longer. In order to illustrate by how much the parallel algorithms are faster than the corresponding sequential algorithms, Figure 8 shows the speedup for the parallelized preprocessing methods for 50, 100 and 200 CEL files. An average speedup of up to the factor 10 may be achieved.

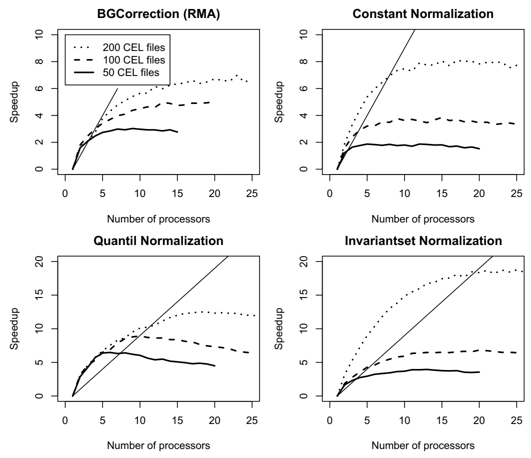

Figure 8. Speedup for the parallelized preprocessing methods for 100 and 200 CEL files.

4.2. Conclusion

In summary, a package of parallelized and efficient preprocessing methods for high-density oligonucleotide microarrays was presented. In the future, parallelization is supposed to be extended to several other well-known and approved methods (e.g. VSN [7], FARMS [6]). The combination of multiprocessors and message-passing parallelization will most likely yield much more efficient preprocessing algorithms.

The basic framework introduced in this article has been adapted to R, using the affy and Snow packages. Table 4 shows the currently available parallelized preprocessing functions. The affyPara library will be available as of April 2008 in the Bioconductor Project.

References

[1] B. M. Bolstad. *Low-level Analysis of High-density Oligonucleotide Array Data: Background, Normalization and Summarization*. PhD thesis, University of California, Berkeley, 2004.

[2] L. Dagum and R. Menon. Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, Jan.–March 1998.

[3] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[4] R. Gentleman, V. J. Carey, W. Huber, R. Irizarry, and S. Dudoit. *Bioinformatics and Computational Biology Solutions*

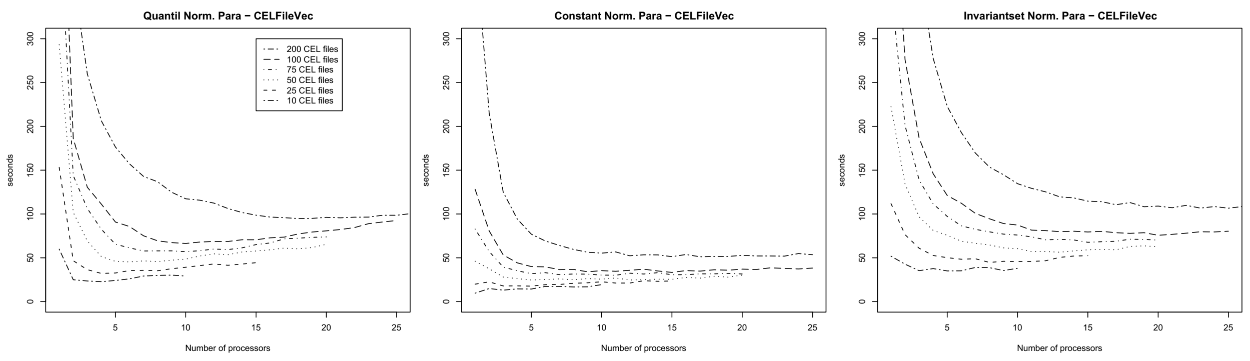

Figure 7. Computation time for parallelized normalization methods.

Table 4. Preprocessing functions in the new library affyPara.

		affy	affyPara
BGC	RMA	bg.correct.rma	bgCorrectPara
	MAS 5.0	bg.correct.mas	bgCorrectPara
	RMA alt	bg.correct.rma2	bgCorrectPara
Normalization	Scaling	normalize.AffyBatch.constant	normalizeAffyBatchConstantPara
	invariantset	normalize.AffyBatch.invariantset	normalizeAffyBatchInvariantsetPara
	Quantiles	normalize.AffyBatch.quantiles	normalizeAffyBatchQuantilesPara
Summarization		computeExprSet	computeExprSetPara
complete	preprocessing	threestep, expresso	preproPara

Using R and Bioconductor. Statistics for Biology and Health. Springer, 2005.

[5] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zhang. Bioconductor: Open software development for computational biology and bioinformatics. *Genome Biology*, 5:R80, 2004.

[6] S. Hochreiter, D.-A. Clevert, and K. Obermayer. A new summarization method for affymetrix probe level data. *Bioinformatics*, 22(8):943–949, Apr 2006.

[7] W. Huber, A. von Heydebreck, H. Sltmann, A. Poustka, and M. Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18 Suppl 1:S96–104, 2002.

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

[9] R. A. Irizarry, B. Hobbs, F. Collin, Y. D. Beazer-Barclay, K. J. Antonellis, U. Scherf, and T. P. Speed. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, Apr 2003.

[10] U. E. B. A. Lukk M, Nikkila J. Application of text mining to create human gene expression atlas from public data. (Submitted), 2007.

[11] H. Parkinson, M. Kapushesky, M. Shojatalab, N. Abeygunawardena, R. Coulson, A. Farne, E. Holloway, N. Kolesnykov, P. Lilja, M. Lukk, R. Mani, T. Rayner, A. Sharma, E. William, U. Sarkans, and A. Brazma. Arrayexpress–a public database of microarray experiments and gene expression profiles. *Nucleic Acids Res*, 35(Database issue):D747–D750, Jan 2007.

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

[13] H. Sevcikova. Statistical simulations on parallel computers. *Journal of Computational and Graphical Statistics*, 13(4):886–906, 2003.

[14] J. M. Squyres and A. Lumsdaine. A component architecture for lam/mpi. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.

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

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