American Journal of Applied Mathematics and Statistics, 2014, Vol. 2, No. 4, 224-230
Available online at http://pubs.sciepub.com/ajams/2/4/9
© Science and Education Publishing
DOI:10.12691/ajams-2-4-9

A Survey of R Software for Parallel Computing

Esam Mahdi*

Department of Mathematics, Islamic University of Gaza
*Corresponding author: emahdi@iugaza.edu.ps

Received June 16, 2014; Revised July 31, 2014; Accepted August 04, 2014

Abstract This article provides a summary of a selection of some of the high-performance parallel packages (libraries) available from the Comprehensive **R** Archive Network (CRAN) using the statistical software **R**. These packages can utilize multicore systems often found in modern personal computers as well as computer cluster or grid computing in order to provide linear speed up the computations in many of advanced statistical modern applications. Some illustrative **R** parallel codes are given in order to introduce the reader to some basic ideas about parallel programming in **R** packages.

Keywords: *R, high performance computing, network of workstations, message passing interface, parallel computing, computer cluster, grid computing, multicore systems*

Cite This Article: Esam Mahdi, "A Survey of R Software for Parallel Computing." *American Journal of Applied Mathematics and Statistics*, vol. 2, no. 4 (2014): 224-230. doi: 10.12691/ajams-2-4-9.

1. Introduction

Many of the recent computational statistical analysis involve advanced algorithms with massive datasets and large numbers of parameters need to be estimated. In particular; DNA sequence analysis in bioinformatics (Vera, Jansen and Suppi 2008), bootstrap and Monte-Carlo simulations in multivariate time series analysis (Mahdi and McLeod 2012), neural network (Seiffert 2002), and cross-validation are computationally very intensive. Researchers have more and more need for parallel computation in order to speed up the computational bottlenecks. This can accomplish by dividing the calculations into smaller tasks and distributing them in simultaneous processing of multiple systems in order to obtain the statistical results much faster than those obtained based on the sequential computation procedures.

In general, the parallel computing aims to do three things: splitting the calculations into chunks, executing the chunks simultaneously in parallel mode using the slaves nodes (alternative name is workers), and combining the results back together to the master node. In other words, the algorithm of the parallel computing illustrated in Figure 1 bellow can be described into steps:

Step 1: Set up the cluster by initializing the master node and the worker processes.

Step 2: Export all needed variables/objects and data to all slaves.

Step 3: Do the needed parallel calculations using calculation functions. Repeat as many times as needed. For example, 1000 repetitions on a cluster of 10 CPUs will be distributed so that each slave simultaneously deals with 100 repetitions.

Step 4: Wait for all slaves to complete their tasks and to send the results back to the master node.

Step 5: End the parallel execution and shut down the worker processes.

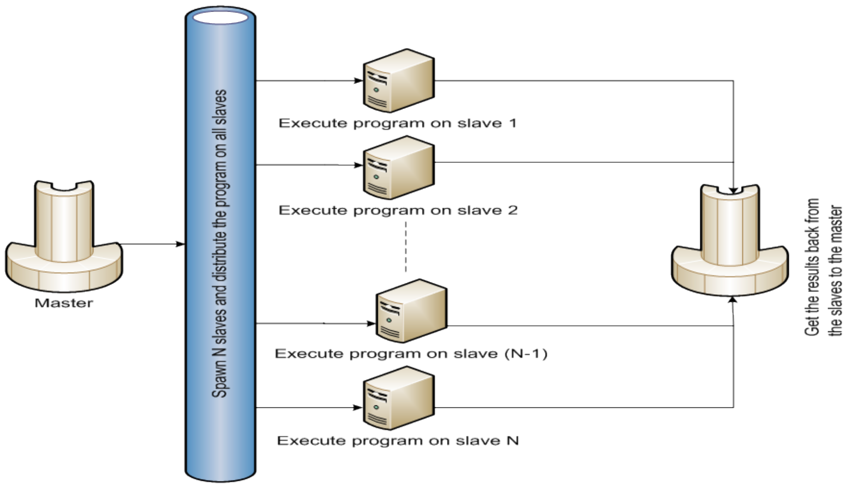
Master
Spawn N slaves and distribute the program on all slaves
Execute program on slave 1
Execute program on slave 2
Execute program on slave (N-1)
Execute program on slave N
Get the results back from the slaves to the master

Figure 1. Schematic diagram illustrating the algorithm of a parallel program.

Super-fast computers with multiple processors are useful for high-performance computing as they have the ability to provide a linear speed up with respect to the number of processors. Basically, high-performance computing can be achieved using: a computer cluster (Bader & Pennington, 2001) or a Grid computing (Rossini, Li, & Tierney, 2007) or a workstation personal desktop/laptop with multicore systems.

A cluster constitutes single machines, called nodes, in which the worker processes communicate and managed by a master node using the standardized Message Passing Interface (MPI)[1] or Parallel Virtual Machine (PVM)[2] or network sockets (SOCKETS) (Saltzer, Clarrk, Romkey, & Gramlich, 1985). In this sense, a personal computer with multicore/CPU systems may be seen as a computer cluster where each core/CPU behaves like a node in the cluster.

MPI and network SOCKETS are more popular than PVM. They have become the de facto standard in parallel computing.

MPI was designed by a group of researchers from academia and industry to function on a wide variety of parallel computers such as Beowulf clusters[3] (Sterling, et al. 1999). It can be implemented by some software such as MPICH/MPICH2[4], LAM/MPI[5], and OpenMPI[6]. Most of the implementations use **C**, **C**$^{++}$, or **FORTRAN** in order to run the computations in a parallel mode.

A network socket is defined as an endpoint of an inter-process communication flow across a computer network. The popular communication between computer sockets is based on the Internet Protocol such as TCP/IP. The term socket refers to an entity that is uniquely identified by the socket number called IP address.

Lack of knowledge about MPI, SOCKET, **C**, **C**$^{++}$, and **FORTRAN** could be a barrier for lots of statisticians who are dealing with intensive statistical computations and trying to speed up the calculations by implementing parallel algorithms.

The **R** software as a free high quality open source project has established itself as the choice of many researchers in such cases. It is a high-level programming language includes some packages listed on the Comprehensive **R** Archive Network (CRAN) to provide the communications layer required for interfaces high-performance parallel computing.

However, **R** itself does not do the parallel computing directly; users do not have to know **C** or **FORTRAN** in order to run parallel jobs. They just need to know how to configure their computers (cluster, grid, or personal multicore workstation desktop/laptop) by setting up the correct host name associated with its node number in order to implement the parallel computing using **R**.

A good introduction to parallel programming for statistical purposes can be found in Rossini, Tierney, & Li, (2007) and in Sevcikova, (2004). Eexcellent instructions to install and run **R** parallel package **Rmpi** under Linux and Windows are given by the maintainer of this package[7].

[1] http://www.mpi-forum.org/
[2] http://www.csm.ornl.gov/pvm/
[3] http://www.beowulf.org/
[4] http://www.mcs.anl.gov/research/projects/mpich2/
[5] http://www.lam-mpi.org/
[6] http://www.open-mpi.org/
[7] http://www.stats.uwo.ca/faculty/yu/Rmpi/

Knaus, (2010) breifly discussed the requirements for the **R** parallel packages **snowfall** and **snow**.

In Section 2, we provide a summary of a selection of some of the high quality published computational **R** packages that can utilize multicore/CPUs often found in modern personal computer as well as computer cluster or grid computing. We stress on the most widely used **R** parallel packages: **Rmpi**, **snow**, **snowfall**, and **parallel**. In Section 3, we briefly compare between some of the contributed loop functions in **R**. An introduction to parallel bootstrapping and Monte-Carlo simulations is given in Section 4 and simple example with illustrated **R** parallel codes is given in Section 5.

2. Some Contributed Parallel Packages to CRAN

We provide a summary of high performance computing **R** packages that might be of most general of research interest, **rpvm**[8], **Rmpi**[9], **nws**[10], **snow**[11], **snowfall**[12], **foreach**[13], **multicore**[14], and **parallel**[15]. A more complete overview is given in the task views[16].

The first **R** package introduced to CRAN was **rpvm**. It was proposed by Li and Rossini in 2001 so it can distribute the computation over many computers (a computer cluster) using the PVM standard. This package is no longer available on CRAN. More details about this package can be found in Schmidberger, Morgan, Eddelbuettel, Yu, Luke, & Mansmann, (2009).

2.1. Rmpi: Interface (Wrapper) to MPI

The **R** package **Rmpi** was introduced by Hao Yu in 2002 to run initially under LAM/MPI. It has been developing over the years to work as an interface (wrapper) under other implementations of MPI such as Microsoft MPI, LAM/MPI, MPICH/MPICH2, OpenMPI, and Deino MPI environments. It can be run under various versions of Linux and Windows.

The common **Rmpi** codes running on computer cluster and multicore systems in which **Rmpi** is properly installed can be summarized into steps:

Step 1: Start cluster and use the command *mpi.spawn.Rslaves()* to detect the host name associated with its node number automatically chosen by MPI or specific hosts assigned by the argument *hosts*. For example, the command *mpi.spawn.Rslaves(nslaves=8)* means that a cluster with 8 slaves to those hosts automatically chosen by MPI will start **R** in a parallel mode.

Step 2: Send objects to all slaves using the function *mpi.bcast.Robj2slave()*. In this step, one can use the function *mpi.scatter.Robj2slave()* in order to distribute a list of objects from the master node to all slave workers. The function *mpi.setup.rngstream()* maybe used (optional)

[8] http://cran.r-project.org/web/packages/rpvm/index.html
[9] http://cran.r-project.org/web/packages/Rmpi/index.html
[10] http://cran.r-project.org/web/packages/nws/index.html
[11] http://cran.r-project.org/web/packages/snow/index.html
[12] http://cran.r-project.org/web/packages/snowfall/index.html
[13] http://cran.r-project.org/web/packages/foreach/index.html
[14] http://cran.r-project.org/web/packages/multicore/index.html
[15] This package becomes a part of **R** >=2.14.0.
[16] http://cran.r-project.org/web/views/HighPerformanceComputing.html

to initialize the random number stream as we will discuss in Section 4.

Step 3: Do the parallel calculations on the slaves (determined in Step 1) simultaneously using suitable functions implemented in **Rmpi** such as *mpi.remote.exec(), mpi.apply(), mpi.parSapply(), mpi.parReplicate(),* and others.

Step 4: Stop the cluster by shutting down the **R** slaves spawned in Step 1 using the *mpi.close.Rslaves()* command.

Some other functions from this package are briefly summarized in the Appendix of this article. More details can be found in Schmidberger, Morgan, Eddelbuettel, Yu, Luke, & Mansmann, (2009).

2.2. Nws: R Functions for NetWorkSpaces and Sleigh

The **nws** package (stands for Net Work Spaces) was submitted to CRAN in 2006 by REvolution Computing so it provides coordination and parallel execution facilities using Sleigh.

Sleigh is a part of NetWorkSpaces (NWS) allows users to execute tasks in parallel. More details about this package can be found in Schmidberger, Morgan, Eddelbuettel, Yu, Luke, & Mansmann, (2009).

2.3. Snow: Simple Network of Workstations

The **snow** package proposed to literature by Rossini, Tierney and Li, (2007). It uses the PVM, MPI, NWS standards as well as direct SOCKETS. This package is widely used in parallel computing in **R**. It should be noted that **snow** requires the packages **Rmpi**, **nws**, and **rpvm** as interfaces to use the MPI, NWS, PVM standards respectively.

Similar to what we have discussed about **Rmpi**, the common snow codes for parallel computing can be summarized into steps:

Step 1: Start the cluster using any of the contributed functions *makeCluster(), makePVMcluster(), makeMPIcluster(), makeNWScluster(),* or *makeSOCKcluster()*. The functions *makePVMcluster(), makeMPIcluster(), makeNWScluster(),* and *makeSOCKcluster()* are used to start cluster under PVM, MPI, NWS, and SOCKETS using the corresponding **R** packages **rpvm**, **Rmpi**, **nws**, and **snow** itself respectively. By default, the function *makeCluster()* uses **Rmpi** to start the job under MPI but others ("SOCK", "PVM", or "NWS".) can be also used via its argument type.

Step 2: Send a list of objects to all workers using the function *clusterExport()*.

Step 3: Do the calculations implementing the slaves determined in Step 1 using any of the contributed functions such as *clusterEvalQ(), parLapply(), parSapply(), parApply(), parRapply(), parCapply(),* and others. In this step, one can use the functions *clusterCall()* or/and *clusterApply()* to call or/and distribute a list of objects from the master to the slave workers.

Step 4: Stop the cluster using the function *stopCluster()*.

Some other functions from this package are briefly summarized in the Appendix of this article and more discussion can be found in Tierney, Rossini and Li (2009).

2.4. Snowfall: Easier Cluster Computing (Based on Snow)

The **snowfall** package was developed by Knaus, Porzelius, Binder, & Schwarzer, (2009) in order to improve the package **snow**. It provides an easier parallel programming using SOCKETS, MPI, PVM and NWS support. The **sfCluster** package is a UNIX management tool was developed by Knaus based on LAM/MPI to help the users of the **snowfall** package setting up the cluster and shutting it down. The functions available from **snowfall** can be used in sequential or parallel modes using the build in **snowfall** function *sfInit()*. The function *sfStop()* is used to stop the cluster.

More details about these packages can be found in Knaus, Porzelius, Binder, & Schwarzer, (2009) and Schmidberger, Morgan, Eddelbuettel, Yu, Luke, & Mansmann, (2009).

2.5. Multicore: Parallel Processing of R Code on Machines with Multiple Cores or CPUs

The **multicore** package proposed by Simon Urbanek in 2009 and provides a way of running parallel computations in **R** using the forking[17] techniques on machines running with POSIX operating systems with multiple cores. It should be noted that this package is not compatible with Windows operating system and it is only running on MacOS X.

Recently, the R-Core team starts to include a new library parallel in **R** software releases >=2.14.0. This library contains a slightly revised copy of some useful functions taken from the **multicore** package as we will discuss in Section 2.7.

2.6. Foreach: Foreach Looping Construct for R

In **R**, repeated executions can be accomplish using one of the looping functions such as *for(), repeat(), while(),* and *replicate()*. The family of *apply()* function which includes the functions *apply(), lapply(), sapply(), eapply(), mapply(),* and *rapply()* can be also implemented to evaluate some statistical expressions repeatedly.

The **foreach** package provides a new looping algorithm for executing the **R** code repeatedly converting the *for()* loop statement in **R** to a *foreach()* loop. This algorithm allows general iteration over list of values in a collection (without the use of the body of the loop counter) on personal computer with multicore systems or multiple nodes of a cluster computer.

For parallel execution, the **foreach** package has been adapting the **R** parallel packages **doMC**[18] (based on the package **multicore** on single workstations), **doSNOW**[19] (based on the package **snow** with SOCKETS), and **doMPI**[20] (based on the package **Rmpi**).

2.7. Parallel: Parallel R Package

The package **parallel** was introduced by Luke Tierney[21] and R-Core team in order to support parallel computation in **R**. The first version of this package was included in **R** version 2.14.0 and since then it becomes a

[17] Fork creates a new process (child) as a copy of the current **R** process that can work in parallel to the master process (parent).
[18] http://cran.r-project.org/web/packages/doMC/index.html
[19] http://cran.r-project.org/web/packages/doSNOW/index.html
[20] http://cran.r-project.org/web/packages/doMPI/index.html
[21] Luke Tierney is the maintainer of the **R** package **snow**.

part of the core **R** packages. This package is using coarse-grained parallelization, can be installed in the usual ways and is ready to use after typing:

R> library("parallel").

This package builds on the work done for CRAN packages **multicore** and **snow** with slight revised of copies of these packages by using forking taking from the package **multicore** and sockets taken from the package **snow**.

The steps of computational parallel mode using **parallel** are almost exactly as those we discussed in Section 2.3. The **parallel** package provides new two functions *makePSOCKcluster()* and *makeForkCluster()* in order to spawn the slaves running on the same host as the master or optionally elsewhere. The *makePSOCKcluster()* is very similar to *makeSOCKcluster()* in the package **snow** whereas the function *makeForkCluster()* starts SOCKET cluster by forking on Unix-alike platforms only.

Some other functions from this package are briefly summarized in the Appendix of this article.

3. Analogues of Apply R functions In Paralle Packages

The base **R** library has a set of useful contributed functions for evaluating some expressions repeatedly that we have discussed in Section 2.6. In this article, we have named this group by the family of *apply()* function. The functions *apply(), lapply(), sapply(),* and *replicate()* are used in a sequential (not in a parallel) mode in many applications. For example, selecting tuning parameters for neural network models often uses cross validation methodology based on iteration techniques. Usually the structure codes of such techniques include some looping functions taken from the members of this family. For parallel computing these functions are not useful, hence developers of **R** parallel packages provide some alternative parallel functions in order to parallelize the loops and speed up the calculations. Some of these functions are listed in Table 1.

Table 1. List to some of sequential and parallel loop functions in some R libraries

base	Rmpi	snow	snowfall	parallel
apply	mpi.parApply	parApply	sfApply	parApply
apply (row of matrix)	mpi.parRapply	parRapply	-	parRapply
apply (column of matrix)	mpi.parCapply	parCapply	-	parCapply
lapply	mpi.parLapply	parLapply	sfLapply	parLapply/ mclapply[22]
sapply	mpi.parSapply	parSapply	sfSapply	parSapply
replicate	mpi.parReplicate	-	-	-

Roughly, the speed up calculation time of running **R** code simultanousely on a cluster with n workers is approximately:

$$Speed\ up = \frac{time\ with\ 1\ CPU}{time\ with\ n\ CPUs}$$

4. Parallel Monte-Carlo Simulation

Bootstrapping is a nonparametric technique used for deriving estimates of standard errors and confidence intervals for estimates, such as the mean, median, proportion, odds ratio, correlation coefficient or regression coefficient, based on selecting samples (with replacement) from the original dataset (observed dataset). Each selected sample (tested dataset) has the same number of elements as the observed dataset.

Monte-Carlo simulation (Barnard, 1963, Dufour and Khalaf, 2001) is more general than bootstrapping in the sense that it uses the random numbers for simulating the samples from a fitted model to the original dataset. Parametric bootstrap can be seen as a Monte-Carlo technique used to assess the performance of the bootstrapping estimators or predictors.

Monte-Carlo techniques involve massive computation due to the replications in simulations. These techniques have been used in many statistical applications such as multivariate portmanteau tests (Mahdi and McLeod 2012), DNA analysis (Hsiao and Stewart 2008). Parallel computing is a very useful tool aims to speed up the computation in such cases.

Results based on bootstrap and Monte-Carlo simulations are not reproducible unless **R** users set the random seed up in their **R** code. In CRAN, Random Number Generators (RNG) for parallel computing are derived from the libraries **rsprng**[23] and **rlecuyer**[24] based on the algorithms discussed in L'Ecuyer, (1999) and L’Ecuyer, Richard, Chen, & Kelton, (2002).

The function *mpi.setup.rngstream()* from the package **Rmpi** contains support for multiple RNG streams based on the **rlecuyer** package, whereas the functions *clusterSetupRNG(), sfClusterSetupRNG(),* and *clusterSetRNGStream()* from the packages **snow**, **snowfall**, and **parallel** respectively contain multiple RNG streams based on **rsprng** and **rlecuyer** (users can choose either **rsprng** or **rlecuyer** with these functions). These functions from **Rmpi**, **snow**, **snowfall**, and **parallel** share the same idea that each separate worker generates random numbers independently from the others.

5. Applications

Many packages available from CRAN have many applications with support for parallel processing using some contributed parallel functions from the packages that we have discussed above. For example, the package **portes**: Portmanteau Tests for Time Series Models[25], implements the Monte-Carlo significance test of portmanteau time series discussed in Lin and McLeod, (2006) and Mahdi and McLeod, (2012) based on the sequential and parallel modes using the parallel package.

[22] *mclapply()* uses forking from packages **multicore** and it is not working on Windows.

[23] http://cran.r-project.org/web/packages/rsprng/index.html
[24] http://cran.r-project.org/web/packages/rlecuyer/index.html
[25] http://cran.r-project.org/web/packages/portes/index.html

The package **survey**: Analysis of Complex Survey Samples[26], (Lumley, 2004) and the package **adegenet**: an **R** Package for the Exploratory Analysis of Genetic and Genomic Data[27], (Jombart, 2008, & Jombart & Ahmed, 2011) both have some support for parallel processing using **multicore** package. The package **PCIT**: **PCIT** Algorithm-Partial Correlation Coefficient with Information Theory[28], (Watson-Haigh, Kadarmideen, & Reverter, 2010) uses **Rmpi** package for performing the partial correlation coefficient with information theory algorithm developed by Reverter & Chan, (2008). The package **pensim**: Simulation of High-Dimensional Data and Parallelized Repeated Penalized Regression[29], (Waldron, Pintilie, Tsao, Shepherd, Huttenhower, & Jurisica, 2011) implements **snow** package in simulating of continuous, correlated high-dimensional data with time to event or binary response, and parallelized functions for Lasso, Ridge, and Elastic Net penalized regression with repeated starts and two-dimensional tuning of the Elastic Net. Many examples are given in the online reference manual and vignettes of these packages.

In this section, we give a simple example with some illustrated **R** codes using a personal computer with quad core systems. It should be noted that some of these codes are running in a sequential mode and maybe consider being computer intensive while others implement functions taken from the **Rmpi** package in parallel mode. One can modify these codes in order to utilize other packages that we have discussed before.

The example that we introduce make use of the univariate regular time series giving the luteinizing hormone in blood samples (lh) at 10 minutes intervals from a human female. The sample size is 48 samples and it is available from the **R** package datasets (Diggle 1990). The ARMA(1,1) model[30] is fitted to this data and the portmanteau diagnostic test based on the Monte-Carlo version of Ljung-Box test discussed in Lin and McLeod, (2006) and Mahdi and McLeod, (2012) is applied into steps:

Step 1: Fit an ARMA(1,1) model to the time series and then apply the observed Ljung-Box statistic (denoted by obs.stat) at lag 10 using the function *Box.test()* given in base **R** package on the residual of the fitted model:

```
R > fit <-arima(lh, order = c(1,0,1))
R > res <- ts(fit$residuals)
R > n <- length(res)
R > ans <- Box.test(res, lag =10, type = "Ljung-Box")
R > obs.stat <- as.vector(ans$statistic)
```

Step 2: Extract the estimates parameters from the fitted model. These estimators are then will be used to simulate models using Monte-Carlo simulations:

```
R>phi <- as.vector(fit$coef [1])
R>theta <- as.vector(fit$coef [2])
R>sigma <- fit$sigma2
R>demean <- as.vector(fit$coef [3])
```

Step 3: Encapsulate the Monte-Carlo procedures into a function that can be used for simulating models from ARMA(1,1). In this function, the Ljung-Box test is applied on each simulated fitted model (denoted by OneSim.stat):

```
R>MonteCarlo <- function(n, res, phi, theta, sigma, demean) {
+ innov <- sample(x=res,size=n,replace = TRUE, prob = NULL)
+ Sim.Data <- arima.sim(n = n, list(ar = phi, ma = theta), innov =
innov,
+ sd = sqrt(sigma), mean = demean)
+ FitSimModel <- arima(Sim.Data, order = c(1,0,1))
+ rboot <- FitSimModel$resid
+ OneSim.stat <- Box.test(rboot, lag =10, type = "Ljung-
Box")$statistic
+ return(as.vector(OneSim.stat))
+}
```

Step 4: Start the cluster by loading the **Rmpi** package and spawn all slaves. Then apply Monte-Carlo significance test in parallel mode:

```
R>library("Rmpi")
R>mpi.spawn.Rslaves()
R>mpi.setup.rngstream(123)
R>mpi.bcast.Robj2slave(n)
R>mpi.bcast.Robj2slave(res)
R>mpi.bcast.Robj2slave(phi)
R>mpi.bcast.Robj2slave(theta)
R>mpi.bcast.Robj2slave(sigma)
R>mpi.bcast.Robj2slave(demean)
R>mpi.bcast.Robj2slave(MonteCarlo)
R>sim.stat <- mpi.parReplicate(1000, MonteCarlo(
n,res,phi,theta, sigma,demean))
R>mpi.close.Rslaves()
```

Step 5: In the final step, calculate the Monte-Carlo portmanteau p-value (note that the p-value based on the asymptotic distribution of Ljung-Box test is 0.587):

```
R>pvalue <- (1 + sum(as.numeric (sim.stat >= obs.stat)))/(1000 + 1)
R>pvalue
0.5394605
```

Running the previous example in a computer with single core using the sequential function *replicate()* instead of using the parallel function *mpi.parReplicate()* as follows:

```
R> set.seed(123)
R> sim.stat <- replicate(1000, MonteCarlo (n, res, phi, theta, sigma,
demean))
R> pvalue <- (1 + sum(as.numeric (sim.stat >= obs.stat)))/(1000 + 1)
R> pvalue
0.5214785
```

With respect to the computer time, the CPU time is 3.21 seconds to get the output of this example in the parallel mode whereas it takes 10.4 seconds in the sequential mode.

6. Comments and Conclusion

There are many more available **R** parallel packages than those we discussed in this article and many of these are briefly described in the CRAN Task Views.

The reader should be aware that the packages available from CRAN, including those in the task views, need only to obey **R** formatting rules with no computer errors. It is not guaranteed that all of these packages produce correct results. On the other hand packages published by major publishers with high impact factor such as the Journal of Statistical Software (JSS) or Bioinformatics or Springer-Verlag or Chapman & Hall/CRC have been carefully reviewed for correctness and quality.

We have selected those parallel packages that might be of most general interest, that have been most widely used and that we are most familiar with. We have implemented some useful functions available from **Rmpi** package in the applications section using our personal computer with four quad core CPUs, but one can easily modify the

[26] http://cran.r-project.org/web/packages/survey/index.html
[27] http://cran.r-project.org/web/packages/adegenet/index.html
[28] http://cran.r-project.org/web/packages/PCIT/index.html
[29] http://cran.r-project.org/web/packages/pensim/index.html
[30] ARMA model in time series stands for autoregressive moving average model.

implemented applications' code in order to use it with other **R** parallel packages.

Reader should note that **Rmpi** is working in parallel mode with MPI, whereas **snow**, **snowfall**, and **parallel** can be implemented in either sequential or parallel mode using PVM, MPI, NWS, and SOCKETS.

Appendix

Rmpi

Table 2. Some contributed functions to CRAN taken from Rmpi package

Function	Purpose
mpi.spawn.Rslaves	Start the cluster and spawn R slaves
mpi.setup.rngstream	Setups RNG Streams based on rlecuyer package on all slaves
mpi.bcast.Robj	Move a general R object around among master and all slaves
mpi.bcast.Robj2slave	Same as *mpi.bcast.Robj()* function
mpi.scatter.Robj2slave	Distributing a list of data from the master to all slaves
mpi.remote.exec	Do calculations at all slaves
mpi.parReplicate	Parallel version of the function *replicate()* in base R
mpi.apply	Scatter an array to slaves and then apply a function
mpi.iapplyLB	Parallel apply with no blocking features
mpi.close.Rslaves	Shut down the cluster

Snow

Table 3. Some contributed functions to CRAN taken from snow package

Function	Purpose
makeCluster	Start the cluster: The default is the MPI cluster
makeSOCKcluster	Start SOCKET clusters
makeMPIcluster	Start MPI cluster
makePVMcluster	Start PVM cluster
makeNWScluster	Start NWS cluster
clusterSetupRNG	Implementation of Pierre L'Ecuyer's RNG Streams on all slaves
clusterSetupSPRNG	Load the rsprng package and initialize separate streams on all slaves
parLapply	Parallel version of *lapply()* function
parSapply	Parallel version of *sapply()* function
parApply	Parallel version of *apply()* function
parRapply	Parallel row *apply()* function for a matrix
parCapply	Parallel column *apply()* function for a matrix
clusterCall	Call a function on each node and returns list of results
clusterEvalQ	Evaluate a literal expression on each node
clusterExport	Export global variables from master to slaves
stopCluster	Shut down the cluster

Snowfall

Table 4. Some contributed functions to CRAN taken from snowfall package

Function	Purpose
sfInit	Start the cluster
sfGetCluster	Get the snow cluster handler. Use for direct calling of snow functions
sfClusterSetupRNG	Setups RNG Streams on all slaves (L'Ecuyer is default)
sfLibrary	Load an R libraries on all nodes, including master
sfExport	Export variables from the master to all slaves
sfRemove	Remove previous exported variables from slaves and (optional) master
sfExportAll	Export global variables from master to slaves with exception of given list
sfRemoveAll	Remove all global variables from the slaves
sfClusterCall	Call a function on each node and returns list of results
sfClusterEvalQ	Evaluate a literal expression on all nodes
sfLapply	Parallel version of *lapply()* function
sfSapply	Parallel version of *sapply()* function
sfApply	Parallel version of *apply()* function
sfStop	Shut down the cluster

Parallel

Table 5. Some contributed functions to CRAN taken from parallel package

Function	Purpose
detectCores	Detect the number of cores/ CPU automatically
clusterSetRNGStream	Implementation of Pierre L'Ecuyer's RngStreams on all slaves
makeCluster	Same as *makeCluster()* in snow package
makePSOCKcluster	Create a parallel socket cluster
makeForkCluster	Create socket cluster by forking (On Unix-alike platforms only)
clusterCall	Same as *clusterCall()* in snow package
clusterEvalQ	Same as *clusterEvalQ()* in snow package
clusterExport	Same as *clusterExport()* in snow package
parLapply	Same as *parLapply()* in snow package
parSapply	Same as *parSapply()* in snow package
parApply	Same as *parApply()* in snow package
mcmapply	Parallel version *mapply()* function using forking (not for Windows)
stopCluster	Same as *stopCluster()* in snow package

References

[1] Bader, D., & Pennington, R. (2001). *Cluster computing: Applications*. The International Journal of High Performance Computing, 15 (2), 181-185.

[2] Barnard, G. A. (1963). *Discussion of ''The spectral analysis of point processes'' by M. S. Bartlett*. Journal of the royal statistical society, B, 25, 264-96.

[3] Diggle, P. (1990). *Time Series: A Biostatistical Introduction*. Oxford.

[4] Dufour, J.-M., & Khalaf, L. (2001). *Monte-Carlo test methods in econometrics. In companion to theoretical econometrics* (eds B. Baltagi). Oxford:Blackwell.

[5] Hsiao, Y., & Stewart, R. D. (2008). *Monte Carlo simulation of DNA damage induction by x-rays and selected radioisotopes*. Physics in medicine and biology, 53 (1), 233-244.

[6] Jombart, T. (2008). *adegenet: a R package for the multivariate analysis of genetic markers*. Bioinformatics, 24 (11), 1403-1405.

[7] Jombart, T., & Ahmed, I. (2011). *adegenet 1.3-1: new tools for the analysis of genome-wide SNP data*. Bioinformatics, 27 (21), 3070-3071.

[8] Knaus, J. (2010, 03 04). *Developing parallel programs using snowfall. Retrieved from CRAN* : cran.r-project.org/web/packages/snowfall/vignettes/snowfall.pdf

[9] Knaus, J., Porzelius, C., Binder, H., & Schwarzer, G. (2009). *Easier parallel computing in R with snowfall and sfCluster*. The R Journal, 1 (1).

[10] L'Ecuyer, P., Richard, S., Chen, E. J., & Kelton, W. D. (2002). *An object-oriented random-numberpackage with many long streams and substreams*. Operations Research, 50, 1073-1075.

[11] L'Ecuyer, P. (1999). *Good parameters and implementations for combined multiple recursive random number generators*. Operations Research, 47, 159-164.

[12] Lin, J.-W., & McLeod, A. I. (2006). *Improved Pen˜a-Rodrıguez portmanteau test*. Computational statistics and data analysis. 51 (3), 1731-1738.

[13] Lumley, T. (2004). *Analysis of complex survey samples*. Journal of Statistical Software, 9 (1), 1-19.

[14] Mahdi, E., & McLeod, I. (2012). *Improved multivariate portmanteau test*. Journal of Time Series Analysis, 33 (2), 211-222.

[15] Reverter, A., & Chan, E. (2008). *Combining partial correlation and an information theory approach to the reversed engineering of gene co-expression networks*. Bioinformatics, 24 (21), 2491-2497.

[16] Rossini, A., Tierney, L., & Li, N. (2007). *Simple parallel statistical computing in R*. Journal of Computational and Graphical Statistics, 16 (2), 399-420.

[17] Saltzer, J., Clarrk, D., Romkey, J., & Gramlich, W. (1985). *The desktop computer as a network*. 1EEE Journal on selected areas in communications, 3 (3).

[18] Schmidberger, M., Morgan, M., Eddelbuettel, D., Yu, H., Luke, T., & Mansmann, U. (2009). *State of the art in parallel computing with R*. 31 (1), 1-26.

[19] Seiffert, U. (2002). *Artificial neural networks on massively parallel computer hardware*. ESANN'2002 proceedings-European symposium on artificial neural networks, (pp. 319-330). Bruges (Belgium).

[20] Sevcikova, H. (2004). *Statistical simulations on parallel computers*. Journal of Computational and Graphical Statistics, 13 (4), 886-906.

[21] Sterling, T., Becker, D., Salmon, J., & Daniel, S. (1999). *How to build a Beowulf-A guide to the implementation and application of PC clusters*. Cambridge, Ma: The MIT Press.

[22] Tierney, L., Rossini, A., & Li, N. (2009). *Snow: A parallel computing framework for the R system*. International journal of parallel programming, 37, 78-90.

[23] Vera, G., Jansen, R., & Suppi, R. (2008). *R/parallel-speeding up bioinformatics analysis with R*. BMC Bioinformatics, 9 (390).

[24] Waldron, L., Pintilie, M., Tsao, M.-S., Shepherd, F., Huttenhower, C., & Jurisica, I. (2011). *Optimized application of penalized regression methods to diverse genomic data*. Bioinformatics, 27 (24), 3399-3406.

[25] Watson-Haigh, N., Kadarmideen, H., & Reverter, A. (2010). *PCIT: an R package for weighted gene co-expression networks based on partial correlation and information theory approaches*. Bioinformatics, 26 (3), 411-413.