Appendix to "State of the Art in Parallel Computing with R"

Markus Schmidberger
Ludwig-Maximilians-Universität München

Martin Morgan
Fred Hutchinson Cancer Research Center

Dirk Eddelbuettel
Debian Project

Hao Yu
University of Western Ontario

Luke Tierney
University of Iowa

Ulrich Mansmann
Ludwig-Maximilians-Universität München

Abstract

This appendix for the paper "State of the Art in Parallel Computing with R" (Schmidberger, Morgan, Eddelbuettel, Yu, Tierney, and Mansmann 2009) gives some short code examples for all in the paper presented R packages in order to introduce the reader to the basic functionality. Furthermore, the full R code for performance evaluation is provided and some additional tables and figures are included.

Keywords: R, high performance computing, parallel computing, computer cluster, multi-core systems, grid computing, benchmark.

A. R code examples for parallel computing packages

This appendix gives some short code examples for all presented R packages in order to introduce the reader to the basic functionality. The code chunks do not cover the full functionality of the packages. For further details and more functions see the R help files or the vignettes of the packages. For details starting the corresponding technology, please contact your cluster administrator or see the manuals.

The R help files contain descriptions for all R functions. To get more information on any specific named function, for example `mpi.spawn.Rslaves()`, the command is

```
R> library("Rmpi")
R> help(mpi.spawn.Rslaves)
```

Vignettes contain executable examples and are intended to be used interactively. You can access the PDF version of a vignette for any installed package from inside R as follows:

```
R> vignette(package = "Rmpi")
```

This will present a menu where the desired vignette can be selected. Selecting a vignette should cause the corresponding PDF file to open on the system.

A.1. R packages for computer clusters

Rmpi: *R package for MPI*

Depending of the used MPI implementation and resource manager there are different ways for starting R with connection to the running MPI universe. Using Unix the commands for LAM/MPI and openMPI in the latest version look like the following examples:

LAM/MPI: Start LAM universe: `lamboot HOSTFILE`
Start R: `R`
Stop LAM universe: `lamhalt`

openMPI: The MPI universe (openMPI daemons) should be running for default.
Start R: `orterun -n 1 -hostfile HOSTFILE R -no-save`

Start cluster: R code to start or initialize the cluster with four workers:

```
R> library("Rmpi")
R> mpi.spawn.Rslaves(nslaves = 4)
```

Sending data to all slaves: R code to send data `a` and `b` to all slaves:

```
R> a <- 3
R> b <- 1:10
R> mpi.bcast.Robj2slave(a)
R> mpi.bcast.Robj2slave(b)
```

Calculations at all slaves: R code to calculate `a+5` and `sin(5)` at all slaves:

```
R> mpi.remote.exec(a+5)
R> mpi.remote.exec(sin, 5)
```

Calculations at list element distributed to slaves: R code to calculate `sqrt(3)` at first slave, `sqrt(6)` at second slave and `sqrt(9)` and third slave.

```
R> mpi.apply(c(3,6,9), sqrt)
```

Stop cluster: R code to stop cluster and to close R instances at workers.

```
R> mpi.close.Rslaves()
```

rpvm: *R package for PVM*

Start pvm: `pvm HOSTFILE`
Exit pvm console: `pvm> quit`
Start R: `R`
Stop pvm: `pvm> halt`

Start cluster: There is no command to launch R instances at the workers. There is only a command to execute R script files at the slaves.

```
R> library("rpvm")
R> slaves <- .PVM.spawnR(R_Script_File, ntask = 4)
```

For example the "R instance launch files" from the package **snow** can be used. But it is easier to use the package **snow** with **PVM** as communication layer.

The following examples are code chunks which could be used for R script files.

Sending data to all slaves:

```
R> TAG <- 22
R> a <- 3
R> b <- 1:10
R> for (id in 1:length(slaves)) {
+          .PVM.initsend()
+          .PVM.serialize(assign("a", a, envir = .GlobalEnv) )
+          .PVM.send(slaves[[id]], TAG)
+           cat("Work sent to", slaves[id], "\n")
+           .PVM.recv(slaves[[id]], TAG)
+  }
R> for (id in 1:length(slaves)) {
+          .PVM.initsend()
+          .PVM.serialize(assign("b", b, envir = .GlobalEnv) )
+          .PVM.send(slaves[[id]], TAG)
+          cat("Work sent to", slaves[id], "\n")
+          .PVM.recv(slaves[[id]], TAG)
+  }
```

Calculations at all slaves:

```
R> for (id in 1:length(slaves)) {
+          .PVM.initsend()
+          .PVM.serialize(get("a", envir = .GlobalEnv) + 5)
+          .PVM.send(slaves[[id]], TAG)
+          cat("Work sent to", slaves[id], "\n")
+  }
R> results <- list()
R> for (id in 1:length(slaves)) {
+          .PVM.recv(slaves[[id]], TAG)
+           results[id] <- .PVM.unserialize()
+  }
R> print(results)
R> for (id in 1:length(slaves)) {
+          .PVM.initsend()
+          .PVM.serialize( sin(5) )
+          .PVM.send(slaves[[id]], TAG)
+          cat("Work sent to", slaves[id], "\n")
+  }
R> results <- list()
R> for (id in 1:length(slaves)) {
+          .PVM.recv(slaves[[id]], TAG)
+          results[id] <- .PVM.unserialize()
+  }
R> print(results)
```

Calculations at list element distributed to slaves:

```
R> array <- c(3,6,9)
R> for (id in 1:length(array)) {
+          .PVM.initsend()
+          .PVM.serialize( sqrt( array[id] ) )
+          .PVM.send(slaves[[id]], TAG)
+          cat("Work sent to", slaves[id], "\n")
+  }
R> results <- list()
R> for (id in 1:length(array)) {
+          .PVM.recv(slaves[[id]], TAG)
+          results[id] <- .PVM.unserialize()
+  }
R> print(results)
```

Stop cluster:

```
R> .PVM.exit()
```

***nws**: R package for Net Work Spaces*

Start nws: `twistd -y nws.tac`
Start R: `R`
Stop nws: `kill "cat twistd.pid"`

Start cluster: Starting four workers and initialize master. Furthermore remove the first workername from the list. In most cases this is the master and we do not want to have a slave running at the same machine than the master.

```
R> library("nws")
R> ws <- netWorkSpace("test")
R> system("cat HOSTFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> s <- sleigh(nodeList = mycluster[-1],
+                  launch = sshcmd, workerCount = 4)
```

Sending data to all slaves: Using the package **nws** you have to store the data in the network space.

```
R> a <- 3
R> b <- 1:10
R> nwsStore(ws, "a", a)
R> nwsStore(ws, "b", b)
```

To get data from the network space server call:

```
R> a <- nwsFindTry(ws, "a")
R> b <- nwsFindTry(ws, "b")
```

Calculations at all slaves:

```
R> eachWorker(s, function(host, name){
+                            ws <- netWorkSpace(serverHost = host, name)
+                            a <- nwsFindTry(ws, "a")
+                            return(a+5)
+                    }, ws@server@serverHost, ws@wsName)
R> eachWorker(s,sin,5)
```

Calculations at list element distributed to slaves:

```
R> eachElem(s, sqrt, c(3,6,9))
```

Stop cluster:

```
R> stopSleigh(s)
```

snow: *Simple network of workstations*

Start the underlying technology you want to use. For details see the other sections.

Start cluster: Starting four workers and initialize master

```
R> library("snow")
R> c1 <- makeMPIcluster(4)
```

Sending data to all slaves:

```
R> a <- 3
R> b <- 1:10
R> clusterExport(cl, list("a", "b"))
```

Calculations at all slaves:

```
R> clusterEvalQ(cl, a+5)
R> clusterCall(cl, sin, 5)
```

Calculations at list element distributed to slaves:

```
R> clusterApply(cl, c(3,6,9), sqrt)
```

Stop cluster:

```
R> stopCluster(cl)
```

snowFT: *Simple network of workstations with fault tolerance*

For details starting the pvm see Section A.1.2.

Start cluster: Starting four workers and initialize master

```
R> library("snowFT")
R> cl <- makeClusterFT(4)
```

Sending data to all slaves:

```
R> a <- 3
R> b <- 1:10
R> clusterExport(cl, list("a", "b"))
```

Calculations at all slaves:

```
R> clusterEvalQ(cl, a+5)
R> clusterCall(cl, sin, 5)
```

Calculations at list element distributed to slaves:

```
R> clusterApplyFT(cl, c(3,6,9), sqrt)
```

Stop cluster:

```
R> stopCluster(cl)
```

snowfall: *Simple network of workstations with more userfriendly interface*

Start the underlying technology you want to use. For details see the other sections.

Start cluster: Starting four workers and initialize master.

```
R> library("snowfall")
R> sfInit(parallel = TRUE, cpus = 4, type = "MPI")
```

Sending data to all slaves:

```
R> a <- 3
R> b <- 1:10
R> sfExport("a", "b")
```

Calculations at all slaves:

```
R> sfClusterEval(a+5)
R> sfClusterCall(sin, 5)
```

Calculations at list element distributed to slaves:

```
R> sfClusterApply(c(3,6,9), sqrt)
```

Stop cluster:

```
R> sfStop()
```

papply

The package **papply** extends the package **Rmpi** with an improved `apply()` function. All functions of the **Rmpi** package can or have to be used.

Start cluster: Starting four workers and initialize the master.

```
R> library("Rmpi")
R> library("papply")
R> mpi.spawn.Rslaves(nslaves = 4)
```

Calculations at list element distributed to slaves:

```
R> papply(as.list(c(3,6,9)), sqrt)
```

Stop cluster:

```
R> mpi.close.Rslaves()
```

biopara

In the package **biopara** all R instances have to be started manually and **biopara** has to be initialized.

Start cluster: Start workers – four times:

```
R> library("biopara")
R> biopara(37000, "/tmp", "localhost", 38000)
```

Start master:

```
R> library("biopara")
R> slaves <- list( list("slave1", 37000, "/tmp", 38000, ""),
+                  list("slave1", 37000, "/tmp", 38000, ""),
+                  list("slave1", 37000, "/tmp", 38000, ""),
+                  list("slave1", 37000, "/tmp", 38000, "") )
R> biopara("master", 36000, 39000, slaves)
```

Start one more R instance for client:

```
R> library("biopara")
```

Sending data to all slaves:

```
R> out <- biopara(list("master",39000), list("localhost",40000), 4,
+                  assign("a", 3, envir = .GlobalEn) )
R> out <- biopara(list("master",39000), list("localhost",40000), 4,
+                  assign("b", 1:10, envir = .GlobalEn) )
```

Calculations at all slaves:

```
R> out <- biopara(list("master",39000), list("localhost",40000), 4,
+                  get("a", envir = .GlobalEn)+5 )
R> out <- biopara(list("master",39000), list("localhost",40000), 4,
+                  sin(5) )
```

Calculations at list element distributed to slaves: not possible

Stop cluster: Kill all R instances!

taskPR

Depending of the used MPI implementation and resource manager there are different ways for starting R with connection to the running MPI universe.

Start cluster: Starting four workers and initialize the master

```
R> library("taskPR")
R> StartPE(4)
```

Sending data to all slaves:

```
R> PE(a <- 3, global = TRUE)
R> PE(b <- 1:10, global = TRUE)
```

Calculations at all slaves:

```
R> PE(x <- a+5, global = TRUE)
R> PE(y <- sin(5), global = TRUE)
R> POBJ(x)
R> print(x)
R> POBJ(y)
R> print(y)
```

Calculations at list element distributed to slaves:

```
R> sapply(c(3,6,9), function(x){
+                            PE(temp<-sqrt(x))
+                            POBJ(temp)
+                            return(temp)
+                  })
```

Stop cluster:

```
R> StopPE()
```

A.2. R packages for grid computing

GridR

Start cluster / Grid:

```
R> library("GridR")
R> grid.init()
```

Calculation somewhere in the Grid: This is no parallel calculation!

```
R> grid.apply("erg", sin, 5)
```

Parallel calculation in the Grid:

```
R> func <- function(x){
+           library("GridR")
+           grid.init(new-parameters)
+           grid.apply("result2", sin, 5)
+           return(result2)
+  }
R> grid.apply("result", func, 5)
```

multiR

Up to the time of review, there was no code available.

Biocep-R

Biocep-R is a GUI project. Therefore there is no special code.

A.3. R packages for multi-core systems

R/parallel

Start cluster: There is no start command required. You only have to load the package.

```
R> library("rparallel")
```

Paralleization of a loop without data dependencies:

```
R> colSumsPara <- function( x )
+  {
+           result <- NULL
+           if( "rparallel" %in% names( getLoadedDLLs()) )
+           {
+                   runParallel( resultVar = "result",
+                                   resultOp = "cbind", nWorkers = 4 )
+           } else {
+                   for( idx in 1:dim(x)[2] ){
+                           tmp <- sum(x[,idx])
+                           result <- cbind(result,tmp)
+                   }
+                   return( result )
+           }
+  }
R> x  <- matrix(sample(100), nrow = 10)
R> colSumsPara(x)
```

pnmath

Loading the packages they replace the builtin math functions by the parallel versions. The user can use the standard math functions without learning any new code syntax.

```
R> library("pnmath")
R> library("pnmath0")
```

romp

Up to the time of review, there was no package available. Only some example code without documentation was available.

The R code is available in an additional appendix file `v31i01-appendix1.R`.

B. R code for performance evaluation

The benchmark contains three different components: Sending data from the master to all slaves, distributing a list of data from the master to the slaves and a small classic parallel computation example. For every component the execution time will be measured. The time for starting and stopping the cluster and as well for calculating results or parameters will not be measured. To avoid foreign network traffic or processor load the measurement will be done five times and the computation time will be averaged. The results are presented in Table 5 in the paper.

B.1. Used software and hardware

For the benchmark the "hoppy" cluster at the Fred Hutchinson Cancer Research Center in Seattle, WA, USA was used. For the evaluation ten workers were used and at every worker only one R instance was started. The cluster has the following hardware and software configuration:

- each worker: 4 GB main memory, 4 x Intel(R) Xeon(TM) CPU 3.60GHz, Suse Linux version 2.6.12
- Connection network: bandwidths 100 MB/s.
- R Version 2.8.0 and latest packages.
- PVM 3.4.5
- openMPI 1.2.4
- NWS Server 1.5.2 (located at the master)
- Resource Manager: Torque 2.3.2; Using Torque the HOSTFILE is stored in the variable PBS_NODEFILE.

B.2. Sending data from the master to all slaves

In this test component a "big" R object will be sent from the master to all slaves and saved in the Global Environment at the slaves. As "big" R object a list with five 500×500 matrices is used.

```
R> dim <- 500
R> Robj <- rep( list( matrix(1, nrow = dim, ncol = dim) ), 5)
```

Rmpi

```
R> library("Rmpi")
R> mpi.spawn.Rslaves(nslaves = 10)
R> t <- replicate(5, system.time(mpi.bcast.Robj2slave(obj = Robj)))
R> tint <- summary(t[3,])
R> mpi.close.Rslaves()
```

nws

```
R> library("nws")
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> s <- sleigh(nodeList = mycluster[-1],
+                  launch = sshcmd, workerCount = 10)
R> slavefunc <- function(x){
+          assign("Robj", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( eachWorker(s, slavefunc, Robj) ))
R> tint <- summary(t[3,])
R> stopSleigh(s)
```

snow

```
R> library("snow")
```

snow with MPI:

```
R> c1 <- makeMPIcluster(10)
R> t <- replicate(5, system.time( clusterExport(c1, list("Robj")) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with PVM:

```
R> c1 <- makePVMcluster(10)
R> t <- replicate(5, system.time( clusterExport(c1, list("Robj")) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with NWS:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt" ,what="character")
R> c1 <- makeNWScluster(mycluster[2:11])
R> t <- replicate(5, system.time( clusterExport(c1, list("Robj")) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with SOCKET:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeSOCKcluster(mycluster[2:11])
R> t <- replicate(5, system.time( clusterExport(c1, list("Robj")) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snowfall

```
R> library("snowfall")
```

snowfall with MPI:

```
R> sfInit(parallel = TRUE, cpus = 10, type = "MPI")
R> t <- replicate( 5, system.time( sfExport( "Robj" ) ) )
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with PVM:

```
R> sfInit(parallel = TRUE, cpus = 10, type = "PVM")
R> t <- replicate( 5, system.time( sfExport( "Robj" ) ) )
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with NWS:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "NWS",
+                    socketHosts = mycluster[2:11])
R> t <- replicate( 5, system.time( sfExport( "Robj" ) ) )
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with SOCKET:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "SOCK",
+                  socketHosts = mycluster[2:11])
R> t <- replicate( 5, system.time( sfExport( "Robj" ) ) )
R> tint <- summary(t[3,])
R> sfStop()
```

snowFT, papply, biopara, taskPR, rpvm: These packages were excluded from the benchmark. **snowFT** and **papply** are only add on packages. **biopara** only supports socket connections and **taskPR** only supports the LAM/MPI implementation. The package **rpvm** does not have scripts to launch R instances at the workers.

The R code is available in an additional appendix file `v31i01-appendix21.R`.

B.3. Distributing a list of data from the master to the slaves

In this test component a list of R objects will be distributed from the master to the slaves and saved in the global environment at the slaves. Therefore a list with 10 (number of workers) elements will be created. Each element is a 500×500 matrix. Matrix 1 goes to slave 1, matrix 2 goes to slave 2 and so on.

```
R> dim <- 500
R> Robj <- rep( list( matrix(1, nrow = dim, ncol = dim) ), 10)
```

Rmpi

```
R> library("Rmpi")
R> mpi.spawn.Rslaves(nslaves = 10)
R> t <- replicate(5,system.time(mpi.scatter.Robj2slave(Robj)))
R> tint <- summary(t[3,])
R> mpi.close.Rslaves()
```

The function `mpi.scatter.Robj2slav()` is available in **Rmpi** version > 0.5-6.

nws

```
R> library("nws")
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> s <- sleigh(nodeList = mycluster[-1], launch = sshcmd, workerCount = 10)
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
```

```
R> t <- replicate(5, system.time( eachElem(s, slavefunc, Robj) ))
R> tint <- summary(t[3,])
R> stopSleigh(s)
```

snow

```
R> library("snow")
```

snow with MPI:

```
R> c1 <- makeMPIcluster(10)
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( clusterApply(c1, Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with PVM:

```
R> c1 <- makePVMcluster(10)
R> slavefunc <- function(x){
+          assign("x", x, envir=.GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( clusterApply(c1, Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with NWS:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeNWScluster(mycluster[2:11])
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( clusterApply(c1, Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snow with SOCKET:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeSOCKcluster(mycluster[2:11])
```

```
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( clusterApply(c1, Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> stopCluster(c1)
```

snowfall

```
R> library("snowfall")
```

snowfall with MPI:

```
R> sfInit(parallel = TRUE, cpus = 10, type = "MPI")
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( sfClusterApply(Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with PVM:

```
R> sfInit(parallel = TRUE, cpus = 10, type = "PVM")
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( sfClusterApply(Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with NWS:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "NWS",
+                  socketHosts = mycluster[2:11])
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( sfClusterApply(Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> sfStop()
```

snowfall with SOCKET:

```
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "SOCK",
+                   socketHosts = mycluster[2:11])
R> slavefunc <- function(x){
+          assign("x", x, envir = .GlobalEnv); return(NULL)
+  }
R> t <- replicate(5, system.time( sfClusterApply(Robj, slavefunc) ))
R> tint <- summary(t[3,])
R> sfStop()
```

The R code is available in an additional appendix file `v31i01-appendix22.R`.

B.4. Small classic parallel computation example

In this test component the integral of a three-dimensional function should be calculated. Integration of a function over a defined region is a very common example for parallel computing.

Problem formulation

```
R> xint <- c(-1, 2)
R> yint <- c(-1, 2)
R> func <- function(x, y) x^3-3*x + y^3-3*y

R> library("lattice")
R> g <- expand.grid(x = seq(xint[1], xint[2], 0.1),
+                   y = seq(yint[1], yint[2], 0.1))
R> g$z <- func(g$x, g$y)
```

Figure 1 shows the surface of the function. The exact solution of the integral can be analytically calculated.

$$\begin{aligned}\int\int_{[-1,2]\times[-1,2]} x^3 - 3x + y^3 - 3y \; dxdy &= \int_{[-1,2]} \left[\frac{1}{4}x^4 - \frac{3}{2}x^2 + xy^3 - 3xy\right]_{x=-1}^{x=2} dy \\ &= \int_{[-1,2]} -\frac{3}{4} + 3y^3 - 9y \; dy \\ &= \left[-\frac{3}{4}y + \frac{3}{4}y^4 - \frac{9}{2}y^2\right]_{x=-1}^{x=2} = -4.5\end{aligned}$$

Implementation

The volume under the function has to be split in several parts, then the volume of the rectangular boxes can be computed. Width and depth are the size of the split intervals and the height is the function value (in the middle or at a corner). Summing the values returns

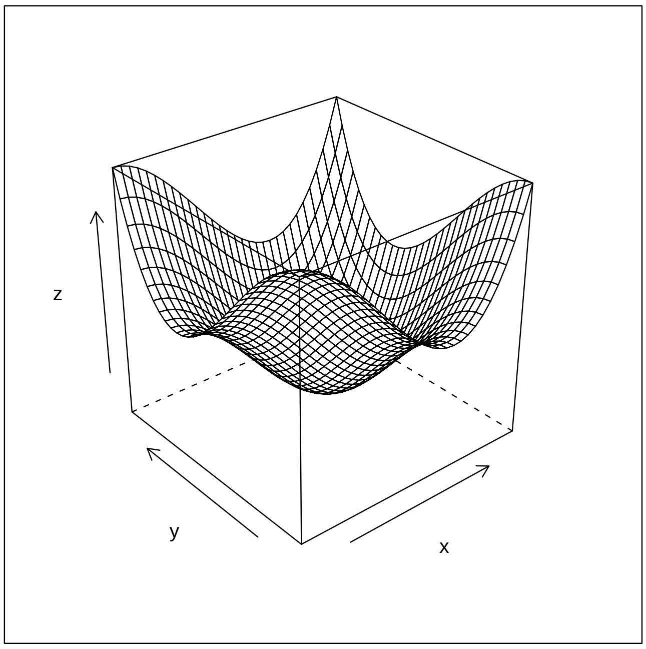

Figure 1: Surface of the example equation for test component 3.

the full integral. The more boxes calculated (smaller boxes), the more exact the calculation gets.

Serial code

There are principle three ways to implement the integration in serial.

Serial calculation of the integral with "for" loops: First of all define interval sizes, then calculate the rectangular box and summarize the boxes:

```
R> integLoop <- function(func, xint, yint, n)
+  {
+          erg <- 0
+          xincr <- ( xint[2]-xint[1] ) / n
+          yincr <- ( yint[2]-yint[1] ) / n
+          for(xi in seq(xint[1], xint[2], length.out = n)){
+                  for(yi in seq(yint[1], yint[2], length.out = n)){
+                          box <- func(xi, yi) * xincr * yincr
+                          erg <- erg + box
+                  }
+          }
+          return(erg)
+  }
```

Serial calculation of the integral with vectorization: In the vectorized code vectors can be used instead of the double loop.

```
R> integVec <- function(func, xint, yint, n)
+ {
+         xincr <- ( xint[2]-xint[1] ) / n
+         yincr <- ( yint[2]-yint[1] ) / n
+         erg <- sum(
+                         func( seq(xint[1], xint[2], length.out = n),
+                                 seq(yint[1], yint[2], length.out = n) )
+                         ) * xincr * yincr * n
+         return(erg)
+ }
```

Serial calculation of the integral with apply functions: In the apply()-like implementation, for every x-interval the complete volume is calculated:

```
R> integApply <- function (func, xint, yint, n)
+ {
+         applyfunc <- function(xrange, xint, yint, n, func)
+         {
+                 yrange <- seq(yint[1], yint[2], length.out = n)
+                 xincr <- ( xint[2]-xint[1] ) / n
+                 yincr <- ( yint[2]-yint[1] ) / n
+                 erg <- sum( sapply(xrange, function(x)
+                                         sum( func(x, yrange)
+                                 )) ) * xincr * yincr
+                 return(erg)
+         }
+         xrange <- seq(xint[1], xint[2], length.out = n)
+         erg <- sapply(xrange, applyfunc, xint, yint, n, func)
+         return( sum(erg) )
+ }
```

Parallel code

Slave function which will be executed at every slave and calculates the area under the function. This slave function will be used for every R package.

```
R> slavefunc<- function(id, nslaves, xint, yint, n, func){
+     xrange <- seq(xint[1], xint[2], length.out = n)[seq(id, n, nslaves)]
+     yrange <- seq(yint[1], yint[2], length.out = n)
+     xincr <- ( xint[2]-xint[1] ) / n
+     yincr <- ( yint[2]-yint[1] ) / n
+     erg <- sapply(xrange, function(x)
+                             sum( func(x, yrange ) )
+                     ) * xincr * yincr
+          return( sum(erg) )
+  }
```

R code for different cluster packages. In principle all the same, only the different syntax of the parallel apply function will be used.

Rmpi:

```
R> integRmpi <- function (func, xint, yint, n)
+  {
+          nslaves <- mpi.comm.size()-1
+          erg <- mpi.parSapply(1:nslaves, slavefunc,
+                          nslaves, xint, yint, n, func)
+          return( sum(erg) )
+  }
```

nws:

```
R> integNWS <- function(sleigh, func, xint, yint, n)
+  {
+          nslaves <- status(s)$numWorkers
+          erg <- eachElem(sleigh, slavefunc, list(id = 1:nslaves),
+                          list(nslaves = nslaves, xint=xint,
+                                  yint = yint, n = n, func = func))
+          return( sum(unlist(erg)) )
+  }
```

snow:

```
R> integSnow <- function(cluster, func, xint, yint, n)
+  {
+          nslaves <- length(cluster)
+          erg <- clusterApplyLB(cluster, 1:nslaves,
+                          slavefunc, nslaves, xint, yint, n, func)
+          return( sum(unlist(erg)) )
+  }
```

snowfall:

```
R> integSnowFall <- function(func, xint, yint, n)
+  {
+          nslaves <- sfCpus()
+          erg <- sfClusterApplyLB(1:nslaves,
+                          slavefunc, nslaves, xint, yint, n, func)
+          return( sum(unlist(erg)) )
+  }
```

Results comparison

R code to calculate the results of the integral.

```
R> n <- 10000

R> erg <- integLoop(func, xint, yint, n)
R> erg <- integVec(func, xint, yint, n)
R> erg <- integApply(func, xint, yint, n)

R> library("Rmpi")
R> mpi.spawn.Rslaves(nslaves = 10)
R> erg <- integRmpi(func, xint, yint, n)
R> mpi.close.Rslaves()

R> library("nws")
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> s <- sleigh(nodeList = mycluster[-1],  launch = sshcmd, workerCount = 10)
R> erg <- integNWS(s, func, xint, yint, n)
R> stopSleigh(s)

R> library("snow")
R> c1 <- makeMPIcluster(10)
R> erg <- integSnow(c1, func, xint, yint, n)
R> stopCluster(c1)

R> library("snowfall")
R> sfInit(parallel = TRUE, cpus = 10, type = "MPI")
R> erg <- integSnowFall(func, xint, yint, n)
R> sfStop()
```

Time measurement

R code to run the time measurements.

Time measurement for serial code:

```
R> rep <- 5
R> tLoop <- replicate(rep, system.time( integLoop(func, xint, yint, n) ))
R> summary(tLoop[3, ])
R> tVec <- replicate(rep, system.time( integVec(func, xint, yint, n) ))
R> summary(tVec[3, ])
R> tApply <- replicate(rep, system.time( integApply(func, xint, yint, n) ))
R> summary(tApply[3, ])
```

Time measurement for parallel code:

```
R> library("Rmpi")
R> mpi.spawn.Rslaves(nslaves = 10)
R> tRmpi <- replicate(rep, system.time(
+                          integRmpi(func, xint, yint, n)
+  ))
R> summary(tRmpi[3, ])
R> mpi.close.Rslaves()

R> library("nws")
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> s <- sleigh(nodeList = mycluster[-1],  launch = sshcmd, workerCount = 10)
R> tNWS <- replicate(rep, system.time(
+                          integNWS(s, func, xint, yint, n)
+  ))
R> summary(tNWS[3, ])
R> stopSleigh(s)

R> library("snow")
R> c1 <- makeMPIcluster(10)
R> tSnow1 <- replicate(rep, system.time(
+                          integSnow(c1, func, xint, yint, n)
+  ))
R> summary(tSnow1[3])
R> stopCluster(c1)
R> c1 <- makePVMcluster(10)
R> tSnow2 <- replicate(rep, system.time(
+                          integSnow(c1, func, xint, yint, n)
+  ))
R> summary(tSnow2[3, ])
R> stopCluster(c1)
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeNWScluster(mycluster[2:11])
R> tSnow3 <- replicate(rep, system.time(
+                          integSnow(c1, func, xint, yint, n)
+  ))
R> summary(tSnow3[3, ])
R> stopCluster(c1)
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeSOCKcluster(mycluster[2:11])
R> tSnow4 <- replicate(rep, system.time(
+                          integSnow(c1, func, xint, yint, n)
+  ))
```

```
R> summary(tSnow4[3, ])
R> stopCluster(c1)
```

```
R> library("snowfall")
R> sfInit(parallel = TRUE, cpus = 10, type = "MPI")
R> tSnowFall1 <- replicate(rep, system.time(
+                          integSnowFall(func, xint, yint, n)
+ ))
R> summary(tSnowFall1[3, ])
R> sfStop()
R> sfInit(parallel = TRUE, cpus = 10, type = "PVM")
R> tSnowFall2 <- replicate(rep, system.time(
+                          integSnowFall(func, xint, yint, n)
+ ))
R> summary(tSnowFall2[3, ])
R> sfStop()
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt" ,what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "NWS",
+                socketHosts = mycluster[2:11])
R> tSnowFall3 <- replicate(rep, system.time(
+                          integSnowFall(func, xint, yint, n)
+ ))
R> summary(tSnowFall3[3, ])
R> sfStop()
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> sfInit(parallel = TRUE, cpus = 10, type = "SOCK",
+                socketHosts = mycluster[2:11])
R> tSnowFall4 <- replicate(rep, system.time(
+                          integSnowFall(func, xint, yint, n)
+ ))
R> summary(tSnowFall4[3, ])
R> sfStop()
```

Time visualization

In the package **snow** the function `snow.time()` can be used to visualize timing informations for the cluster usage. The function `plot()`, motivated by the display produced by ”xpvm”, produces a Gantt chart of the computation, with green rectangles representing active computation, blue horizontal lines representing a worker waiting to return a result, and red lines representing master/worker communications.

```
R> library("snow")
R> c1 <- makeMPIcluster(10)
R> visSnow1 <- snow.time( integSnow(c1, func, xint, yint, n) )
```

```
R> stopCluster(c1)
R> c1 <- makePVMcluster(10)
R> visSnow2 <- snow.time( integSnow(c1, func, xint, yint, n) )
R> stopCluster(c1)
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeNWScluster(mycluster[2:11])
R> visSnow3 <- snow.time( integSnow(c1, func, xint, yint, n) )
R> stopCluster(c1)
R> system("cat $PBS_NODEFILE > cluster.txt")
R> mycluster <- scan(file = "cluster.txt", what = "character")
R> c1 <- makeSOCKcluster(mycluster[2:11])
R> visSnow4 <- snow.time( integSnow(c1, func, xint, yint, n) )
R> stopCluster(c1)

R> pdf(file="snow_times_comp3.pdf")
R> dev.set()
R> par(mfcol=c(2, 2), oma=c(0, 0, 0, 0), mar=c(4, 4, 2.5, 1))
R> plot(visSnow1, title = "Cluster Usage - MPI")
R> plot(visSnow2, title = "Cluster Usage - PVM")
R> plot(visSnow3, title = "Cluster Usage - NWS")
R> plot(visSnow4, title = "Cluster Usage - Socket")
R> dev.off()
```

The R code is available in an additional appendix file `v31i01-appendix23.R`.

C. Additional tables and figures

This appendix section includes additional tables and figures.

Packages	Version	Websites
proftools	0.0-2	`http://CRAN.R-project.org/package=proftools`
		`http://www.cs.uiowa.edu/~luke/R/codetools`
codetools	0.2-1	`http://CRAN.R-project.org/package=codetools`
		`http://www.cs.uiowa.edu/~luke/R/codetools`
profr	0.1.1	`http://CRAN.R-project.org/package=profr`
		`http://github.com/hadley/profr`

Table 1: Overview about R packages for code analysis available on CRAN, including version numbers (as of December 2008) and corresponding URLs.

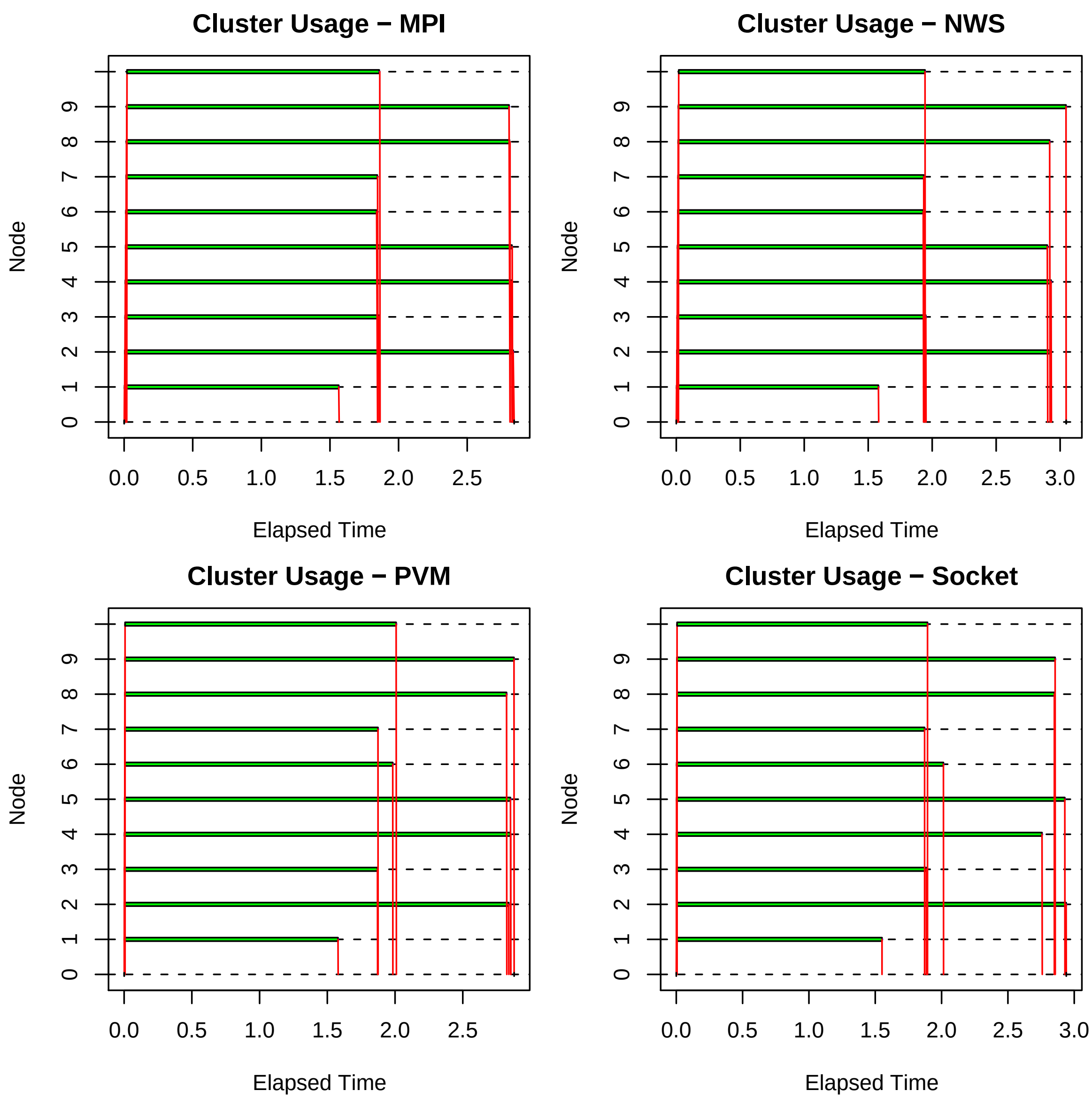

Figure 2: Time visualization of timing informations for the cluster usage for component test 3 with **snow** package. Green rectangles representing active computation, blue horizontal lines representing a worker waiting to return a result (not required in this example), and red lines representing master-slave communications.

Packages / Software	Version	Websites
sfCluster	0.4beta	http://www.imbi.uni-freiburg.de/parallel
SLURM	1.3.11	https://computing.llnl.gov/linux/slurm
		http://sourceforge.net/projects/slurm
SGE	6.2	http://gridengine.sunsource.net
Rsge (Package)	0.4	http://CRAN.R-project.org/package=Rsge

Table 2: Overview about software and R packages for resource manager systems, including version numbers (as of December 2008) and corresponding URLs.

Packages	Version	Websites
rsprng	0.4	http://CRAN.R-project.org/packages=rsprng
		http://www.biostat.umn.edu/~nali/SoftwareListing.html
rlecuyer	0.1	http://CRAN.R-project.org/packages=rlecuyer
rstream	1.2.2	http://CRAN.R-project.org/packages=rstream

Table 3: Overview about R packages for parallel random number generators, including version numbers (as of December 2008) and corresponding URLs.

References

Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, Mansmann U (2009). "State of the Art in Parallel Computing with R." *Journal of Statistical Software*, **31**(1), 1–27. URL http://www.jstatsoft.org/v31/i01/.

Affiliation:

Markus Schmidberger
Division of Biometrics and Bioinformatics, IBE
Ludwig-Maximilians-Universität Munich
81377 Munich, Germany
E-mail: Markus.Schmidberger@ibe.med.uni-muenchen.de
URL: http://www.ibe.med.uni-muenchen.de/