Tutorial: Parallel Computing with R on LionX & Hammer
Experimentation with the Parallel Packages R

Research Computing and Cyberinfrastructure Group (RCC)
Pennsylvania State University

July 5, 2013

Abstract

Two major thrusts for the increased interest in high performance computing in the Computational Sciences, like computational bio-science and chemical engineering, include larger data sets (Berman et al. 2003) and increased computational requirements stemming from more sophisticated computational models.

There are a lot of examples, to support this argument. In bio-informatics and genomic researches, data-sets appear to be growing at a rate that is faster than the corresponding performance increase in hardware (Shaffer 2007). Also, many popular important some-times hybrid computational methods like Markov chain Monte Carlo (MCMC) and Gibbs sampling, bootstrapping, and Monte Carlo sim-ulation are used in diverse areas of statistical analysis- investigating geographic, econometric, and biological data.
One way to approach these two major demands, increased data size and increased simulation demands, is via parallel computing.
In the simplest case, data may simply be divided among parallel pro-cessors.
In this Tutorial we'll perform few simple experimentation to review the state of parallel computing among R two parallel libraries- snow and Rmpi, and provides a starting point for researchers interested in adopting parallel computing methods in R. The Tutorial is organized

as follows: Section 1, reviews some common techniques to improve R code (Markus et al. 2009). Section 2 presents R packages for LionX and Hammer clusters at RCC (as of today), Section 3 provides a simple PBS for R for Batch use on clusters and Section 4 reviews few useful tips for parallel computation with R.

1 Code Analysis for R

Common parallellization techniques in R include (Gentelman 2008, M. Schmidberger et al, 2009):

Vectorized computation: Vectorized computation means any function call that, when applied to a vector, automatically operates directly on all elements of the vector. Many functions in R are vectorized. These are often implemented in C , and hence very fast. This greatly reduces the need for loops and similar constructs that are prevalent in other languages. Vectorized computations should be used when possible.

apply()-functions: There are a number of functions that can be used to apply a function, iteratively, to a set of inputs. Appropriately implemented apply()-like functions help to recognize vectorized structure of code, and are often very readily adopted to parallel computation.

Foreign language interfaces: R provides a powerful interface to functions and libraries written in other languages. The main reason for using foreign languages is efficiency, since R is not compiled. In some situations the performance of R can be substantially improved by writing code in a compiled language(M. Schmidberger et al, 2009).
The most widely used interfaces are the .C() and .Call() functions which provide linkage to C and compiled code. R itself is largely written in C and it is easy to make use of the internal data structures, macros and code from routines written in C (see R High Performance Computing, Task View).

2 Parallel Computing Packages in R on LionXs Machines and Hammer

A thorough over view of all packages on R for parallel computing is provided by M. Schmidberger et al, 2009, in the Table 1 as of 2009.

Also, Portable Batch Scripting (PBS) used at Penn State for resource management is a key part of parallel computation. We'll talk about it later and provide a sample script to begin with.

2.1 Rmpi

The package Rmpi (Yu 2002) is a wrapper to MPI, providing the R interface to low-level MPI Functions (Yu 2002). The Rmpi package includes scripts to launch R instances at the workers from the master using the command mpi.spawn.Rslaves() The instances run until closed, with mpi.close.Rslaves() .

Listing 1: Rmpi Example

```
#Set up R and load the required package
library(snow)
library(Rmpi) #(optional)

#Create a cluster of 2 slave processes for MPI
cl <- makeCluster(2, type = "SOCK")  #MPI for mpi
#feed from the workers
do.call("rbind", clusterCall(cl, function(cl) Sys.info()
                                 ["nodename"]))

#Lower Level Functions
clusterEvalQ(cl, library(splines))   #evaluation on all nodes

x = c(1,2,3,4,5) #from master to each worker assign values
clusterExport(cl, "x")

#Higher Level Functions
parApply(cl, matrix(1:10, ncol=2), 2, sum)

x <- 1:100
# unparallel verision
```

```
pf(x,df1=4,df2=5)
# parallel verision
unlist(parLapply(cl,x,pf,df1=4,df2=5))

#Stopping a snow Cluster
stopCluster(cl)
```

Listing 2: Snow Example

```
#apply a set of functions F_i(x, y, a_i) on a set of (xi, yi)
library(snow)
cl <- makeCluster(2, type = "MPI")
a = 1:10
clusterExport(cl, "a")
wrapper_F<-function(nrep){
  x = rnorm(10)
  y = rnorm(10)
  z1 = y^2 + x^2 + a
  z2 = y^2 - x^2 - a
  outlist = list(z1 = z1, z2 = z2)
  return(outlist)}
parLapply(cl, 1:10, wrapper_F)

#(CRAN, 2012) to set up a cluster of 5 processes on 3
#machines (3 on dogleg), start all 5 processes, and stops the
   cluster.

library(snow)
machines <- c("dogleg","dogleg","dogleg", "sugar", "strike")
cl <- makeCluster(machines,type="SOCK")
invisible(clusterEvalQ(cl, library(nice)))
invisible(clusterEvalQ(cl, set.my.priority(19)))

stopCluster(cl)
```

2.2 nws

The package nws (Bjornson et al. 2007) provides functions to store data in a so-called "Net-WorkSpace (NWS), and uses the 'sleigh' mechanism for parallel execution. And its implemented in Python language.
There are two basic functions for executing tasks in parallel eachElem(), eachWorker().

nws requires a running NetWorkSpace server NetWorkSpaces currently supports the Python, Matlab, and R languages and allows limited cross-language data exchange.

2.3 snow

The package snow (Rossini et al. 2007) supports simple parallel computing in R.
The snow package includes scripts to launch R instances on the slaves with cl < − makeCluster() .

The instances run until they are closed explicitly via the command stopCluster(cl) .

The package provides support for high-level parallel functions like apply() and simple error- handling to report errors from the workers to the manager.

3 PBS script for R

Depending on the number of jobs on the computing nodes or one a queue, a resource manager system should allocate the requested memory and CPU time for each job submitted in batch. Also, they provide load balancing which is very useful especially when the package doesnt provide it itself-Rmpi, snow, snowfall and biopara support load balancing for the apply() functions. At RCC, this is done through a PBS, independent of the R language. PBS and other schedulers make parallel computing for administrators. One example of a PBS scripts used with R is described here.
Consult `http://rcc.its.psu.edu/user_guides/system_utilities/pbs/` for more information.

Listing 3: Sample PBS for R Script

```
#PBS -l nodes=1:ppn=4
#PBS -l walltime=2:00:00
#PBS -j oe
cd $PBS_O_WORKDIR
# load module and then run R
module load R/2.15.2
time R --no-restore --no-save <inputfile_name.in> output_name.
   out
#PBS -M your_email_address_here
```

Installation

Please contact us at *rcc@rcc.psu.edu* to install or customize any package that you may need for your work.

Fault tolerance and load balancing

Fault tolerance for parallel algorithms provides continued operations in the event of failure of some workers. On the other hand, load balancing is a technique to spread processes between resources in order to get optimal resource utilization (M. Schmidberger et al, 2009).
Only the packages snowFT and biopara have intergraded solutions for faults tolerance (M. Schmidberger et al, 2009). However, one can take necessary measures for backing up temporary results through his PBS.

4 Tips for Parallel Computing with R

Some generally-accepted best practices from parallel computing (M. Schmidberger et al, 2009)

- Communication is much slower than computation; care should be taken to minimize unnecessary data transfer to and from workers
- Maximize the amount of computation performed in each remote evaluation. Some functions produce large results, so it pays to reduce these on the workers before returning the final return.
- R's rules for lexical scoping, its serializing functions and the environments they are defined in all require some care to avoid transmitting unnecessary data to workers.
- Functions used in `apply()` -like calls should be defined in the global environment, or in a package name space.

- Every R instance requires its own main memory, and the amount of main memory will frequently limit scalability.
- Grid computing with R is relatively new, at early stage level (M. Schmidberger et al, 2009)

More Advanced tips:

- Very large objects, e.g., provided as additional apply() -like function parameters, might in some cases be more efficiently constructed on each worker, rather than being forwarded from the manager to each worker (M. Schmidberger et al, 2009).
- Random number generators require extra care. It is important to avoid producing the same random number stream on each worker and at the same time be able to facilitate reproducible research. Special-purpose packages (rsprng, rlecuyer) are available; the snow package provides an integrated interface to these packages.

References

[1] Markus Schmidberger, Martin Morgan, Dirk Eddelbuettel, Hao Yu, Luke Tierney, Ulrich Mansmann (2009), State of the Art in Parallel Computing with R, Journal of Statistical Software.

[2] Berman F, Fox G, Hey AJ (2003). Grid Computing: Making the Global Infrastructure a Reality.

[3] Gentleman R (2008). R Programming for Bioinformatics, volume 12 of Chapman & Hall/CRC Computer Science & Data Analysis. Chapman & Hall/CRC.

[4] Gentleman RC, Carey VJ, Bates DM, Bolstad B, Dettling M, Dudoit S, Ellis B, Gautier L, Ge Y, Gentry J, Hornik K, Hothorn T, Huber W, Iacus S, Irizarry R, Leisch F, Li C, Maechler M, Rossini AJ, Sawitzki G, Smith C, Smyth G, Tierney L, Yang JY, Zhang J (2004). Bioconductor: Open Software Development for Computational Biology and Bioinformatics. Genome Biology, URL `http://genomebiology.com/2004/5/10/R80`.

[5] Yu H (2002). Rmpi: Parallel Statistical Computing in R. R News, 2(2), 10-14. URL http://CRAN.R-project.org/doc/Rnews/.

[6] Bjornson R, Carriero N, Weston S (2007). Python NetWorkSpaces and Parallel Programs. Dr. Dobb's Journal, pp. 1-7. URL http://www.ddj.com/web-development/200001971.

[7] Rossini AJ, Tierney L, Li NM (2007). Simple Parallel Statistical Computing in R. Journal of Computational and Graphical Statistics, 16(2), 399-420

[8] CRAN Task View: High-Performance and Parallel Computing with R: http://cran.r-project.org/web/views/HighPerformanceComputing.html

[9] Luke Tierney, HPC computing for HPC class, University of Iowa http://www.stat.uiowa.edu/~luke/classes/295-hpc/ http://users.stat.umn.edu/~yiyang/seminar/