Computational Methods for Statistic with Applications
Computer Exercise no. 5: Sparse matrices

NGSSC, LU, SLU, UU

September, 2011

The goal with this lab is that you get acquainted with some packages in $\mathbb{R}$, which enable parallel computations.
At the end, selected results of the exercises have to be sketched and sent to the lab-consultant.
Some examples of functions and script files can be downloaded via
`http://user.it.uu.se/~maya/Courses/NGSSC/index_Stat.html`.
Make a copy of these files in some of your directories.

Utilizing a multicore processor

Exercise 1 ($\mathbb{R}$ package 'multicore')

Tasks

1. Login on some of the multicore Unix machines

linne.it.uu.se	scheele.it.uu.se
sernander.it.uu.se	svedberg.it.uu.se
tiselius.it.uu.se	fries.it.uu.se
polhem.it.uu.se	

   These are 8-core machines and we can use Shared Memory type of parallelism to execute computation in parallel.

   Please coordinate with your colleagues so that not too many of you login on one and the same machine.

2. Start $\mathbb{R}$ and install the package `multicore`

3. Perform the following test and check the time spent for various values of n: $10^3, 10^5, 10^6, 10^7$

```
library('multicore')
y <- function(x) { z <- (x^3+sqrt(x))/x^(1/3) }
x=1:n
system.time(lapply(x, y))

system.time(mclapply(x, y))
```

Check if you are running in parallel: in a separate terminal window type 'top' and you should see multiple tasks with your user name running simultaneously.

4. Lower the computational load. Consider

   ```
   y <- function(x) { z <- (x^3}
   ```

   Do you have improvement in the run time (over `lapply`) when you use `mclapply`?

   Why?

5. Choose your own function and experiment. Have a look at the user manual of `multicore` - there are more possibilities to explore.

6. Since Kalkyl has also multicore processors, you could repeat your experiments there.

   Proceed as follows:

   ```
   ssh -AX kalkyl.uppmax.uu.se
   interactive -A g2011131 -t 0:15:00
   module unload pgi
   module load gcc
   R
   > install.packages('multicore')
   library('multicore')
   ...
   ```

   Again, via 'top' you could see 8 threads running in parallel.

Utilizing Kalkyl as a cluster

<u>Exercise</u> 2 (ℝ packages 'Rmpi, snow')

Experience the packages `Rmpi` and `snow`, and how to submit jobs to the queueing system.
Tacks:

1. You should read (have read) the user guide `http://www.uppmax.uu.se/support/user-guides/kalkyl-user-guide`. It is expected that you understand how the batch scripts look like, how to check the status of the submitted jobs, cancel submitted or running jobs etc.

2. Install the necessary packages as follows:

   ```
   module unload pgi openmpi
   module load gcc openmpi
   R
   > install.packages('Rmpi')
   > install/packages('snow')
   ```

3. Repeat the example from the lecture notes to compute a double integral in parallel:

$$\int_{-1}^{2}\int_{-1}^{2}(x^3 - 3x + y^3 - 3y)dx\,dy = -4.5$$

   To this end, you are given the three $\mathbb{R}$ routines, which compute the integral in a serial fashion: `integLoop, integVec, integApply`.

   (a) It is recommended to check the performance of the three versions, to also see the effect of utilizing the vectorization in the computations. Check and run `integr_serial_run.r`.

   (b) For the parallel experiments you have at your disposal `slavefunc, integRmpi, integSnow, integr_Rmpi_run.r, integr_Snow_run.r` and examples of batch files: `script_run_Rmpi_integr, script_run_Snow_integr`

4. You are given another simple example for computing the row-sum of a matrix (`snow_rowsum.r` and `script_run_snow_rowsum`). It is suggested that you run the example, monitor the parallel performance and reason on the timing results - is it scalable?

   Use different sizes and number of processes.

   Prepare a summary of the timing results for various number of processes and problem sizes.

Exercise 3 (More packages)

Install the packages `foreach` and `doMC`. Then try:

```
require(foreach) # loads package foreach
require(doMC)    # loads both doMC and multicore
search()         # make sure all 3 packages are loaded
n=1000           # size of the matrices
registerDoMC(cores=2) #tells R to spawn a maximum of 2 processes
x <- foreach(i=1:10) %dopar% svd(matrix(rnorm(n*n),ncol=n))
```

Change the number of threads, test with various sizes. What is the speedup you obtain?