UPPSALA
UNIVERSITET



# Parallelization issues

http://cran.r-project.org/web/views/HighPerformanceComputing.html

UPPSALA
UNIVERSITET

# Parallel computing - not the primary development goal for $\mathbb{R}$

- Large data seta
- More sophisticated methodologies and, thus, increasing computational requirements (MCMC, bootstrapping, Gibbs sampling, resampling)

Very often the data can be split into ’chunks’ and analysed in parallel - no data dependencies and embarrasingly parallel algorithms.

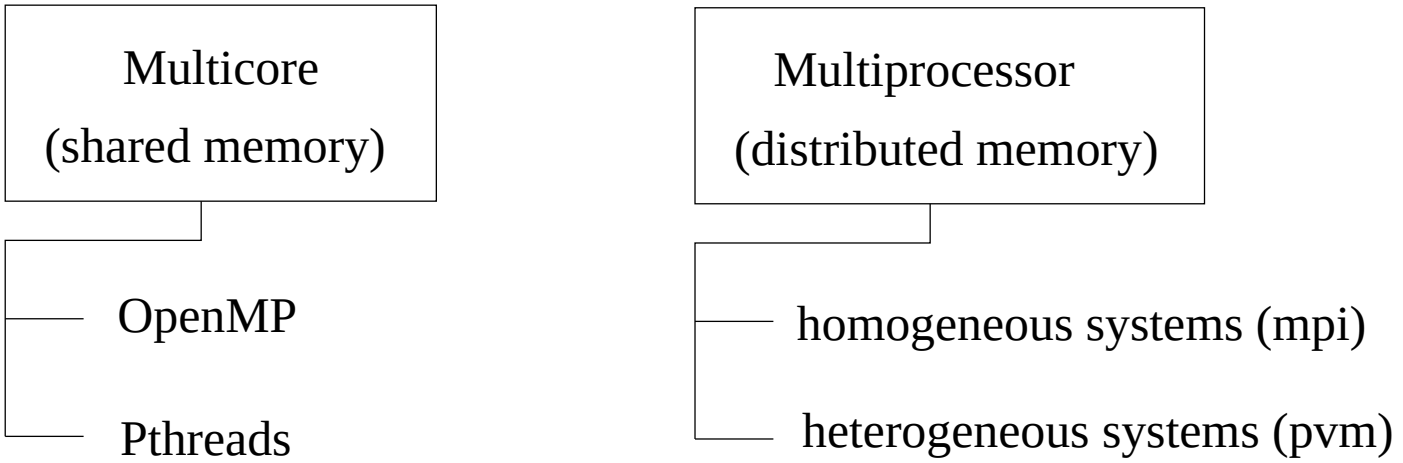

# Parallel $\mathbb{R}$

Code characteristics, suitable for parallelizing:

- vectorized computations
- `apply()`-type functions
- foreign language interfaces

UPPSALA UNIVERSITET

# Parallel $\mathbb{R}$ - target hardware systems



UPPSALA
UNIVERSITET

# Parallel $\mathbb{R}$

- PC/Servers (multicore machines)
  - Windows/MacOS: doSMP
  - Unix/Linux:
- Clusters - typically, some message passing library is used (mpi, pvm)
  - Linux: Rmpi, rpvm, snow, RScaLAPACK, paRc
- Grid

UPPSALA
UNIVERSITET

# Parallel $\mathbb{R}$ - Multicore

UPPSALA
UNIVERSITET

- `pnmath` (OpenMP), `pnmath0` (Pthreads), 2009
  Replaces math functions by hand-craft parallel versions
- `fork`, 2007
  wrappers around Unix processe management API calls, s.a. fork, signal, wait ,...
- `rparallel`, 2008
  single function `rinParallel()`
  The package enables automatic parallelization of loops with no data dependencies.
- `romp`, 2008
  the package transforms the core to fortran, incerts OpenMP directives and compiles the code, which is then executed in R.


# Parallel $\mathbb{R}$ - Multicore

- `multicore`
  All jobs share the full state $\mathbb{R}$ when the parallel instances are spawned, thus, no data or code his copied (FAST!).
  Spawning uses the `fork` system call.
  A pipe is established between the master and the child process, and can be used to send data to the master process.
  `mclapply()`

UPPSALA
UNIVERSITET

# Parallel $\mathbb{R}$ - Clusters

UPPSALA UNIVERSITET

# Parallel $\mathbb{R}$ - Clusters: by 2009 - 9 packages

- `Rmpi` 2002
  a wrapper to MPI, providing R-interface to MPI functions.
  MPI should be installed. Linux, Windows, Mac OS X
  Launches R slaves `mpi.spaw.Rslaves()` until `mpi.close.Rslaves()`
- `snow`, 'Simple Network Of Workstations' ,(L. Tierney, A. Rossini, M. Na Li, 2003)
  Provides interface to MPI, PVM, raw sockets
  based on master-slave model
  `cl<-makeCluster(10,type="MPI"`
  `stopCluster(cl)`
  Supports `apply(), lapply(), C<-parMM(cl,A,B)`

UPPSALA
UNIVERSITET

# $\mathbb{R}$ - using external libraries

Blas, PBLAS (Netlib 2007)
Configuration option for $\mathbb{R}$ : '-with-blas'

UPPSALA
UNIVERSITET

# ℝ - GRID

`GridR`, 2007
`multiR`, 2008

UPPSALA
UNIVERSITET

# Windows: doSMP

The doSMP package provides a parallel backend for the foreach/%dopar% function. It executes tasks on a single, multiprocess/multicore machine.

| | |
|---|---|
| `startWorkers` | start worker processes |
| `stopWorkers` | shutdown worker processes |
| `registerDoSMP` | register doSMP to be used by foreach/%dopar% |
| `rmSessions` | cleanup orphaned doSMP sessions |

UPPSALA UNIVERSITET

# Windows: doSMP

Install doSMP

```
> require(doSMP)
Loading required package: doSMP
Loading required package: foreach
Loading required package: iterators
Loading required package: codetools
foreach: simple, scalable parallel programming from Revolution Analytics
Use Revolution R for scalability, fault tolerance and more.
http://www.revolutionanalytics.com
Loading required package: revoIPC
Warning messages:
1: package 'doSMP' was built under R version 2.13.1
2: package 'foreach' was built under R version 2.13.1
3: package 'iterators' was built under R version 2.13.1
4: package 'revoIPC' was built under R version 2.13.1
```

UPPSALA UNIVERSITET

# ℝ : Windows: doSMP

http://www.r-statistics.com/2010/04/...
...parallel-multicore-processing-with-r-on-windows/

```
require(doSMP)
workers <- startWorkers(2) # My computer has 2 cores
registerDoSMP(workers)

# create a function to run in each itteration of the loop
check <-function(n) { for(i in 1:1000) {
sme <- matrix(rnorm(100), 10,10)
solve(sme) }
}

times <- 10 # times to run the loop

# comparing the running time for each loop
system.time(x <- foreach(j=1:times ) %dopar% check(j)) # 2.56
seconds
system.time(for(j in 1:times ) x <- check(j)) # 4.82 seconds

stopWorkers(workers)
```

UPPSALA UNIVERSITET

# ℝ : multicore

One may need to install R-pathed-devel-2.XX.Y
For example,

http://cran.r-project.org/bin/linux/suse/README.html

```
library('multicore')
```

Typical application: `lapply`

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

x <- 3:5

b <- lapply(x, y)
```

UPPSALA
UNIVERSITET

# $\mathbb{R}$ : multicore

Case 1: Not enough computing load

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

Nothing interesting happens.

UPPSALA UNIVERSITET

# ℝ : Linux: multicore (on Maya's laptop)

Case 1: More reasonable computing load

```
y <- function(x) { z <- (xˆ3+sqrt(x))/xˆ(1/3) }
x=1:10000000
> system.time(lapply(x, y))
   user  system elapsed
125.776   0.546 126.899

> library('multicore')
> system.time(mclapply(x, y))
   user  system elapsed
 22.187   0.606  83.444
```


# $\mathbb{R}$ : multicore

| Size | lapply | | | mclapply | | |
|---|---|---|---|---|---|---|
| | user | system | elapsed | user | system | elapsed |
| | | | Unix (halfrunt) | | | |
| $10^5$ | 0.86 | 0.002 | 0.86 | 0.89 | 0.80 | 0.36 |
| $10^6$ | 10.06 | 0.07 | 10.13 | 6.65 | 4.76 | 3.51 |
| $10^7$ | 155.32 | 0.76 | 156.09 | 51.25 | 28.54 | 39.13 |
| $5\,10^7$ | 1016.56 | 4.26 | 1020.92 | 138.97 | 8.58 | 254.11 |
| | | | Linux (kalkyl) | | | |
| $10^7$ | 111.97 | 0.83 | 112.82 | 24.38 | 0.87 | 25.25 |

# ℝ : Unix (halfrunt): multicore

```
> top
   PID USERNAME  LWP  PRI  NICE   SIZE    RES STATE
  7941 maya        1    0     0 2329M   274M cpu/6
  7945 maya        1    0     0 2329M   271M cpu/0
  7942 maya        1    0     0 2329M   274M run
  7947 maya        1    0     0 2329M   267M run
  7944 maya        1    0     0 2329M   274M cpu/5
  7943 maya        1    0     0 2329M   274M cpu/7
  7948 maya        1    0     0 2329M   267M cpu/3
  7946 maya        1    0     0 2329M   264M cpu/2
```

UPPSALA
UNIVERSITET

# Clusters: SNOW: Simple Network of Workstations

(L. Tierney, A. Rossini, M. Na Li, 2003)

- higher level framework for simple parallel jobs
- communication: via sockets, rpvm or rmpi
- based on master-slave model
- one call creates the cluster (makeCluster(size))
- automatic handling of parallel random number generator
- one call for repeated evaluation of an arbitrary function on the cluster

UPPSALA
UNIVERSITET

# Examples:

## Evaluating a double integral: serial implementation

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

UPPSALA UNIVERSITET

# Evaluating a double integral: serial implementation

```
integLoop <- function(func, xint, yint, n)
{
 local_sum <- 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
         local_sum <- local_sum + box
    }
 }
 return(local_sum)
}
```

UPPSALA UNIVERSITET

# Evaluating a double integral: vectorized implementation

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

UPPSALA UNIVERSITET

# Evaluating a double integral: apply()

```
 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
      local_sum <- sum( sapply(xrange, function(x)
      sum( func(x, yrange)
         )) ) * xincr * yincr
      return(local_sum)
   }
   xrange <- seq(xint[1], xint[2], length.out = n)
   local_sum <- sapply(xrange, applyfunc, xint, yint, n, func)
   return( sum(local_sum) )
}
```

UPPSALA UNIVERSITET

# Evaluating a double integral: parallel code

```
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
 local_sum <- sapply(xrange, function(x)
                      sum( func(x, yrange ) )
    ) * xincr * yincr
 return( sum(local_sum) )
}
```

UPPSALA
UNIVERSITET

# Evaluating a double integral: Rmpi

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

UPPSALA
UNIVERSITET

# Evaluating a double integral: snow

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

UPPSALA UNIVERSITET

# Evaluating a double integral: the batch file

```
#!/bin/bash -l
#SBATCH --job-name=integr_Rmpi
#SBATCH -t 00:20:00
#SBATCH --output=integr_Rmpi_10000_16.out
#SBATCH --error=integr_Rmpi_10000_16.err
#SBATCH -p node -N 2
#SBATCH -A p2009041

echo "TMPDIR=$TMPDIR"
echo "job script running on host $(hostname)"

module unload pgi openmpi
module load gcc openmpi

echo "--->>> Starting on: " $(date)
mpirun -np 1 R --no-save < integr_Rmpi_run.r
echo "--->>>  Finalizing: " $(date)
```

UPPSALA
UNIVERSITET

# Example: parallel row-sum of a matrix

## 8 cores

Serial runs on kalkyl

| Size | Time |
|---|---|
| $10^3$ | 4 |
| $10^4$ | 32 |

UPPSALA
UNIVERSITET

# Example: numerical integration

Serial runs on kalkyl

| Type | $n = 1000$ | $n = 10000$ |
|---|---|---|
| Loop | 6.44 | 678.23 |
| Vec | 0.001 | 0.003 |
| Apply | 0.269 | 16.90 |

UPPSALA
UNIVERSITET

# Example: numerical integration

Parallel runs on kalkyl, $n = 10000$

| Processes | Time |
|---|---|
| 1 | 16.90 |
| 2 | 7.16 |
| 4 | 3.61 |
| 8 | 2.63 |
| 16 | 1.39 |