Parallelization issues

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

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)

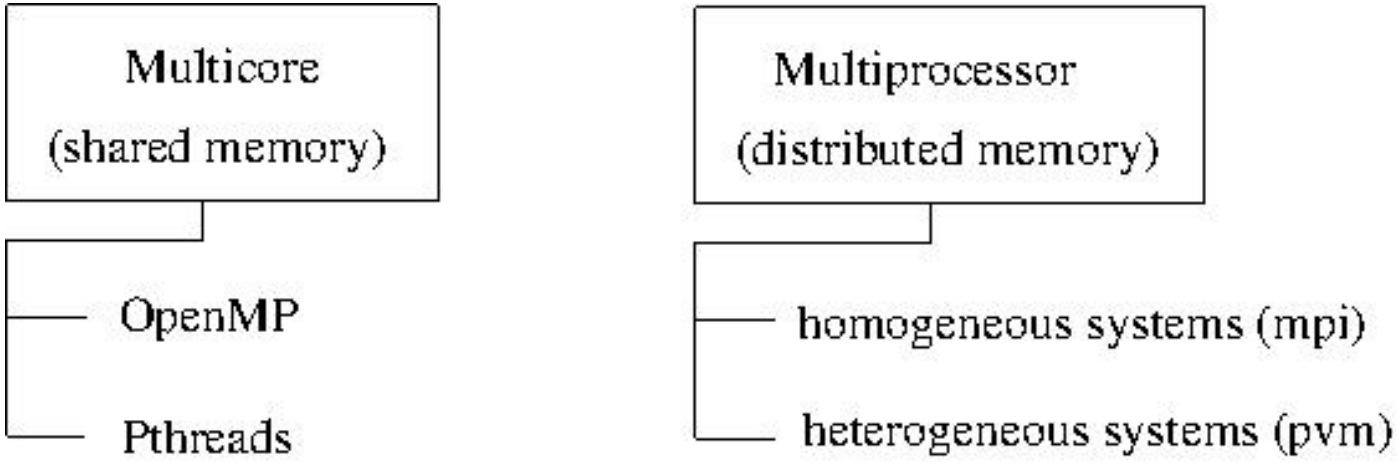
Very often the data can be split into 'chunks' and analysed in parallel - no data dependencies and embarrasingly parallel algorithms.

Parallel $\mathbb{R}$

Code characteristics, suitable for parallelizing:

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

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



Parallel $\mathbb{R}$

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

Parallel $\mathbb{R}$ - Multicore

- `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()`
  Enables automatic parallelization of loops with no data dependencies.
- `romp`, 2008
  Transforms the core to fortran, incerts OpenMP directives and compiles the code, which is then executed in R.
- NOW: foreach, doParallel, mclapply

Parallel $\mathbb{R}$ - Multicore

- `multicore`
  All jobs share the full state $\mathbb{R}$ when the parallel instances are spawned, thus, no data or code is 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()`

Parallel $\mathbb{R}$ - Clusters

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)`

$\mathbb{R}$ - using external libraries

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

$\mathbb{R}$ - GRID

`GridR`, 2007

`multiR`, 2008

$\mathbb{R}$: multicore

```
> install.packages("multicore")
...
--- Please select a CRAN mirror for use in this session
Warning message:
package 'multicore' is not available (for R version 3.2.
```

$\mathbb{R}$: multicore

`library('doParallel')`

Typical application: `lapply`

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

x <- 3:5

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

$\mathbb{R}$: multicore

Case 1: Not enough computing load

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

Nothing interesting happens.

$\mathbb{R}$: Linux: multicore (on Maya's laptop)

Case 1: More reasonable computing load

```
y <- function(x) { z <- (x^3+sqrt(x))/x^(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

$\mathbb{R}$: Unix (halfrunt): multicore

```
> top
  PID USERNAME LWP PRI NICE  SIZE   RES STATE     TIME    CPU C
 7941 maya       1   0    0 2329M  274M cpu/6     0:17 12.50% 1
 7945 maya       1   0    0 2329M  271M cpu/0     0:15 12.43% 1
 7942 maya       1   0    0 2329M  274M run       0:16 12.36% 1
 7947 maya       1   0    0 2329M  267M run       0:14 12.33% 1
 7944 maya       1   0    0 2329M  274M cpu/5     0:16 12.23% 1
 7943 maya       1   0    0 2329M  274M cpu/7     0:16 12.17% 1
 7948 maya       1   0    0 2329M  267M cpu/3     0:14 12.12% 1
 7946 maya       1   0    0 2329M  264M cpu/2     0:13 10.36% 1
```

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

Examples:

Evaluating a double integral: serial implementation

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

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)
}
```

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)
}
```

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) )
}
```

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,n
 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) )
}
```

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) )
}
```

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)) )
}
```

Example: parallel row-sum of a matrix

8 cores

Serial runs on kalkyl

Size	Time
10^3	4
10^4	32

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

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

Matlab: parallel toolbox

`parfor`

```
parpool(16)
parfor i = 1:length(A)
    B(i) = func(A(i));
end
```

Matlab: parallel toolbox

Single program, multiple data `spmd`

```
parpool(3)
spmd
  % build magic squares in parallel
  q = magic(labindex + 2);
end
for ii=1:length(q)
  % plot each magic square
  figure, imagesc(q{ii});
end
delete(gcp)
```

Matlab: parallel toolbox

```
drange

results = zeros(1, numDataSets, codistributor());
for i = drange(1:numDataSets)
    load(['\\central\myData\dataSet' int2str(i) '.mat'])
    results(i) = processDataSet(i);
end
res = gather(results, 1);
if labindex == 1
    plot(1:numDataSets, res);
    print -dtiff -r300 fig.tiff;
    save \\central\myResults\today.mat res
end
```

Parallel performance measures and models

- Time, speedup, efficiency
- Models: Amdahl's law, Gustafson-Barsy's law, ... Roofline approach