MilanoR

4th meeting

October 24, 2013

First steps in Parallel Computing

Anna Longari
anna.longari@quantide.com

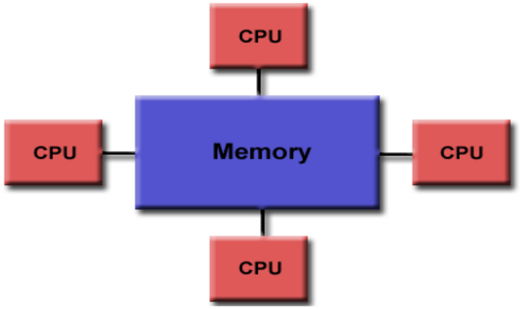
Outline

- Parallel Computing
- Implicit Parallelism
- Explicit Parallelism
- Example on Amazon Servers

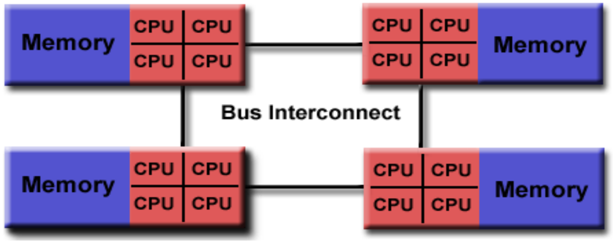
Parallel Computing

Parallel Computing is the simultaneous execution of the source code of one or more programs, specifically adapted, on

more core of the same processor
(Implicit Parallelism)



multiple processors
(Explicit Parallelism)



Parallel Computing with R

There are several packages for parallel computation in R, some of which have existed a long time, e.g. Rmpi, nws, snow, sprint, foreach, multicore...

Package parallel attempts to eliminate some of this by wrapping *snow* and *multicore* into a nice bundle.

Package parallel was first included in **R** 2.14.0

Parallel Computing with R

Detect numbers of CPU’s/cores

Almost all physical CPUs contain two or more cores that run more-or-less independently. However, on some processors these cores may themselves be able to run multiple tasks simultaneously and some OSes (e.g. Windows) have the concept of logical CPUs which may exceed the number of cores.

How many cores in my computer?

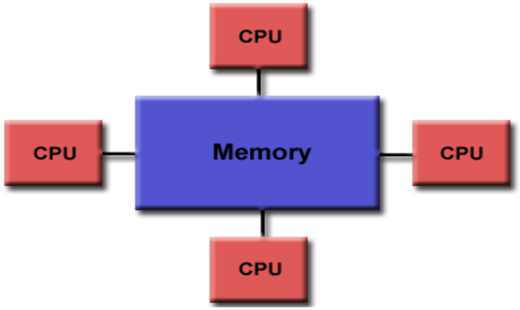
```
> library(parallel)
> detectCores()
[1] 8
```

Implicit Parallelism

Function mclapply

The most common direct applications of packages multicore and snow have been to provide parallelized replacements of lapply, sapply, apply and related functions.

Function mclapply works just like the regular lapply function to iterate across the elements of a list, but iterations automatically run in parallel to speed up the computations.



Implicit Parallelism: example

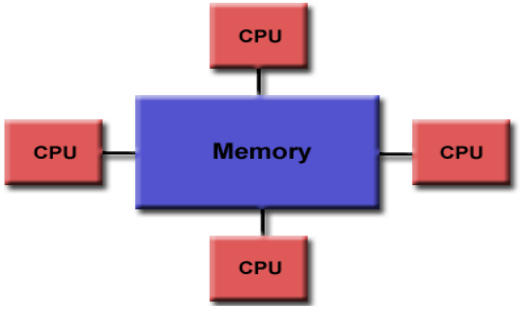
```
> library(parallel)
> f = function(x) {
  sum = 0
  for (i in seq(1,x)) sum = sum + i
  return(sum)}
> n = 10000
```

lapply in current machine:

```
> system.time({out = lapply(1:n,FUN = f)
                out = unlist(out)})
user  system elapsed
20.401   0.036  20.431
```

mclapply on multiple core of current machine:

```
> system.time({out=mclapply(X=1:n,FUN = f, mc.cores=8)
                out = unlist(out)})
user  system elapsed
38.267   1.541   5.766
```

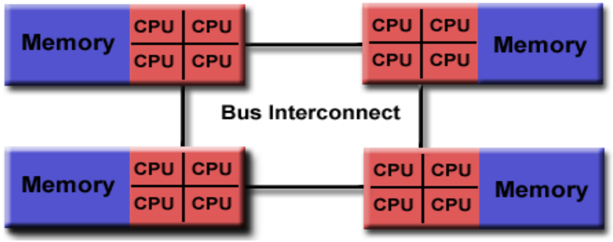


Explicit Parallelism

Explicit Parallelism has the programmer responsible for

- dividing the problem to be solved into independent chunks to run in parallel
- aggregating the result from each chunk

Computations can be extended to all cores of a single computer or networked computers

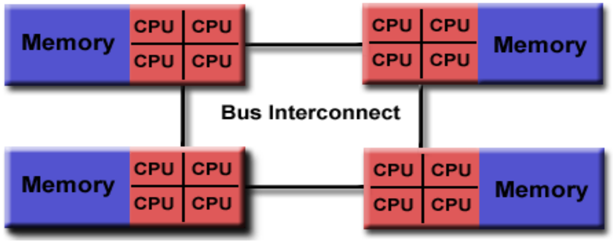


Explicit Parallelism

How to create a cluster with a single machine?

```
> library(parallel)
> nCores <- detectCores()
> nCores
[1] 8
> fx <- function(x) x^2
> cluster <- makeCluster(nCores)
> cluster
socket cluster with 8 nodes on host 'localhost'
> rbind(clusterCall(cluster,fx,x=6))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 36   36   36   36   36   36   36   36
> stopCluster(cluster)
```

clusterCall() call a function with on each node



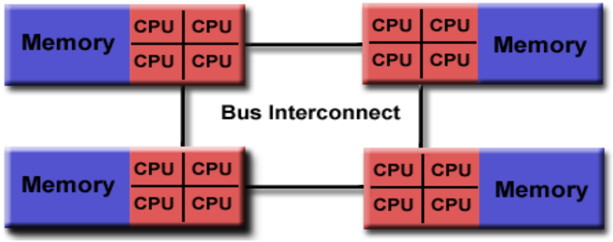
Explicit Parallelism

Variables defined at master level are not directly available to all slaves:

```
> fx <- function(x) x^y
> y <- 2
> cluster <- makeCluster(4)
> rbind(clusterCall(cluster,fx,x=6))
Error in checkForRemoteErrors(lapply(cl, recvResult)) :
  4 nodes produced errors; first error: object 'y' not found
```

Direct export of master variables to all slaves is required:

```
> clusterExport(cluster, "y")
> rbind(clusterCall(cluster,fx,x=6))
  [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,] 36   36   36   36   36   36   36   36
```



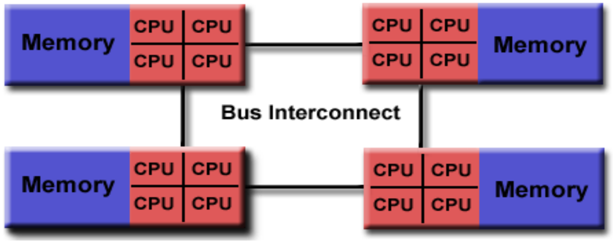
Explicit Parallelism

Similarly, library attachment is required at slave level.

```
> cluster <- makeCluster(2)
> clusterEvalQ(cluster,library(tseries))
[[1]]
[1] "tseries"   "methods"   "stats"     "graphics"  "grDevices" "utils"
"datasets"  "base"

[[2]]
[1] "tseries"   "methods"   "stats"     "graphics"  "grDevices" "utils"
"datasets"  "base"
> stopCluster(cluster)
```

The function clusterEvalQ() evaluates an expression at each cluster node

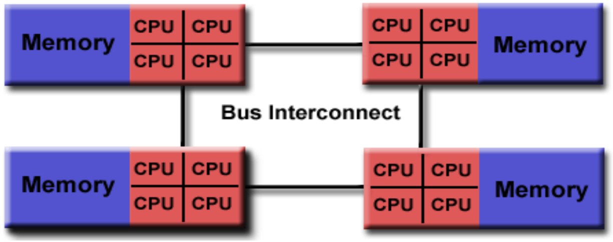


Explicit parallelism

How to create a cluster with a multiple machine

The `spec` argument of the `makeCluster()` function accept the hostname or the IP address of other computers and `master` argument the host name of the master:

```
> spec <- c(rep("localhost",4),rep("192.168.0.4",4))
> cluster <- makeCluster(spec=spec,master=spec[1],type="PSOCK",
port=10187))
```



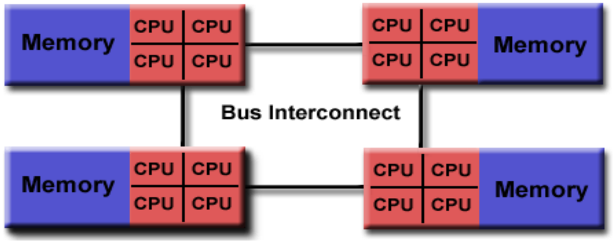
Explicit parallelism

The functions

- parLapply
- parSapply
- parApply

are parallel versions of

- lapply
- sapply
- apply



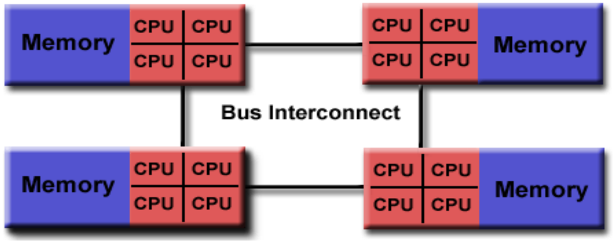
Explicit parallelism: example

mclapply on multiple core of current machine:

```
> system.time({out=mclapply(X=1:n,FUN = f, mc.cores=8)
               out = unlist(out)})
user  system elapsed
38.267   1.541   5.766
```

parLapply in clusters:

```
> spec = c(rep("54.235.155.244",8),
           rep("54.235.184.104",2),
           rep("54.225.204.141",8))
> clus = makeCluster(spec = spec, master = spec[1],
                     type = "PSOCK", port=10187)
> clusterExport(clus,"f")
> system.time({out=parLapply(cl=clus, X=1:n,fun = f)
               out = unlist(out)}
user  system elapsed
0.020   0.000   4.011
```



Example on server Amazon

Functions: simulateTariff on 10 tariff

```
system.tyme(mclapply(X=1:10, FUN=.simulateTariff,
                     lsInit = lsInit, dfTariff= dfTariff,
                     eleTariff= eleTariff, lsParameters = lsParameters,
                     lsMeta = lsMeta, modeBatch = 1,
                     mc.cores = detectCores())

system.tyme(parLapply(cl=acCluster, X=1:10, fun=.simulateTariff,
                     lsInit = lsInit, dfTariff= dfTariff,
                     eleTariff= eleTariff, lsParameters = lsParameters,
                     lsMeta = lsMeta, modeBatch = 1)
```

Thank you!

Anna Longari
anna.longari@quantide.com
3496192376