Bi7740: Scientific computing

Parallel computing in R

Vlad Popovici
popovici@iba.muni.cz

Institute of Biostatistics and Analyses
Masaryk University, Brno

Implementations in R packages

- *snow*: for traditional clusters, supports PVM, MPI,...; is portable (UNIX, Windows)
- *multicore*: targets multi-core/-CPU machines; simple; does not run on Windows; does not handle parallel RNGs
- *parallel*: snow+multicore in new R (>=2.14)
- *foreach*: parallel `for` loops
- *R+Hadoop*: based on Hadoop cluster
- *RHIPE*: based on Hadoop, targets map-reduce operations
- *Segue*: `apply`-like calculations on Hadoop clusters, using Amazon's Elastic MapReduce
- better not to use GUI in the workers

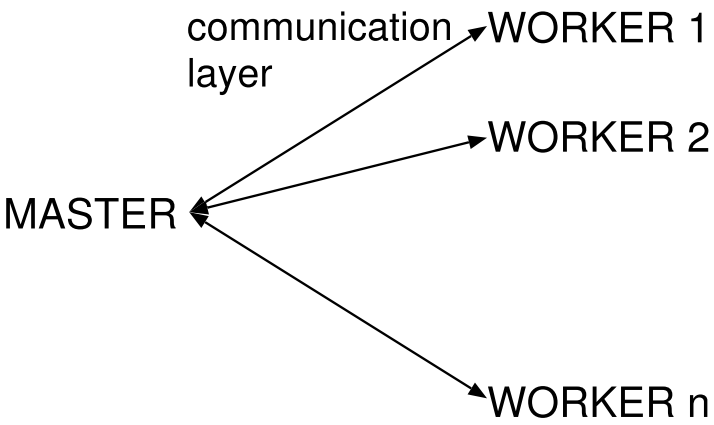
Levels of parallelism in R

- drop-in replacement of standard libraries with parallelized versions: e.g. replace classical BLAS+LAPACK with Intel's MKL or AMD's ACML to exploit multi-core machines
- packages for parallel computing using system's libraries for parallelism
- write your own custom code (e.g. C/C++/Fortran using OpenMP interfaced with R)
- running several R servers

`parallel` package

- tries to unify `snow` and `multicore` packages
- you can either use the master/slave architecture (for cluster-like computing) or the multi-threaded architecture (for shared-memory machines)
- since R 2.14.0 is included in the basic collection of packages
- geared towards massive data: single "instruction", multiple data
- implements a special RNG for parallel streams of random numbers (L'Ecuyer CMRG)

Cluster computing



- it is portable across different computing environments
- can use different communication layers: sockets, MPI, PVM, NetWorkSpaces
- all but socket communication require specific R packages
- can work locally or on a network
- may require proper configuration (e.g. password-less ssh login,...) for network access

```
library(parallel)

cl = makeCluster(4, type='PSOCK')

## computation goes here....
res = parLapply(cl, 1:1000000, sqrt)

stopCluster(cl)
```

- `makeCluster()` and `stopCluster()` are used for initializing and stopping a cluster
- `PSOCK` refers to socket transport layer - works both locally or on a network, but data has to be sent from master to workers. Warning: libraries and functions need to be loaded on each worker - no environment inheritance from the master!
- on Unix you can use `FORK` (relies on POSIX fork() system) which is faster for local usage, but cannot be used on a network; the workers inherit the work environment from the master (shared memory)
- in any case, the cluster is persistent and has to be explicitly closed

Functions for distributing the work

- `cluster...(...)` functions:
  - `...Call`, `...EvalQ`: calls the same function all all workers
  - `...Apply`: applies a function to each element of a list (see also **`lapply`**`(...)`
  - `...ApplyLB`: load balancing version: send the first *n* jobs to the *n* workers, and then submits jobs as the workers become available → my increase tremendously the overhead
  - `...Map`: distributed version of `mapply`
  - `...Split`: splits the data into equal chunks
- **`par`**`...(...)` functions:
  - `...Lapply`, `...Sapply`, `...Apply`: the parallel versions of **`lapply`**, **`sapply`**, and **`apply`**; they have also a `...LB` variant
  - `...Rapply` and `...Capply` for row- and column-wise parallel operations

Exercise: find potentially prognostic probesets/genes.

- load the data file `transbig.rdata`
- `X` is a gene expression matrix, probesets by columns
- `C` is a data frame with clinical covariates
- find the probesets prognostic for relapse-free survival (`C$t.rfs` and `C$e.rfs`) (do not consider other potentially influencing variables):
  - write the non-parallelized version to find the p-values (from Cox PH models) corresponding to each probeset item find all the probesets with adjusted p-value (FDR) ≤ 0.2
  - parallelize the code: identify the code that can be run in parallel (same code, different data), choose a method, implement

Support for parallel RNGs

- still under development but usable
- use `RNGkind("L'Ecuyer-CMRG")` to choose the right RNG
- each worker gets a stream of 2^{127} random numbers, for up to 2^{64} workers
- for `multicore` functions: use `set.seed(...)` to ensure reproducibility
- for cluster usage: `clusterSetRNGStream(clst, seed)`

Example: using cluster computing for bootstrap testing

Test the mean fold change between ER+ and ER- at probeset `205225_at`. Main bits of code:

```
RNGkind("L'Ecuyer-CMRG")
cl = makeCluster(4, type='PSOCK')
clusterSetRNGStream(cl, 1234)

clusterEvalQ(cl,
  {
  # ... initialization ...
  } )

res = clusterEvalQ(cl,
  {
  # ... do the work ...
  } )

stopCluster(cl)
```

Exercise: parallel bootstrap.

- parallelize the function `bstrap.nonparam`
- choose your favourite approach: either based on `clusterEvalQ` or on `parApply`-family of functions

The multicore approach

- the processes are limited to the local machine
- faster communication, shared memory → inherited environment in the workers
- cannot use distributed the computation to other machines
- functions taken from `multicore` package and now renamed in `parallel` package
- these functions are not available under Windows
- main functions:
  - `mclapply`, `mcmapply`
  - `mcparallel`, `mccollect`

Example

Find the probesets potentially prognostic...

```
p = mclapply(as.data.frame(X),
    my.coxph, C$t.rfs, C$e.rfs,
    mc.cores=4)
```

`foreach` package

- used for parallelizing **`for`** loops
- can use various backends: multicore or cluster
- examples:

```
foreach (i=1:10) %do% sqrt(i)
foreach (a=1:3, b=rep(10,3)) %dopar% (b^a)
```

- uses *iterators* and *combiners* for splitting the data and combining the results

Example: Find the probesets potentially prognostic...

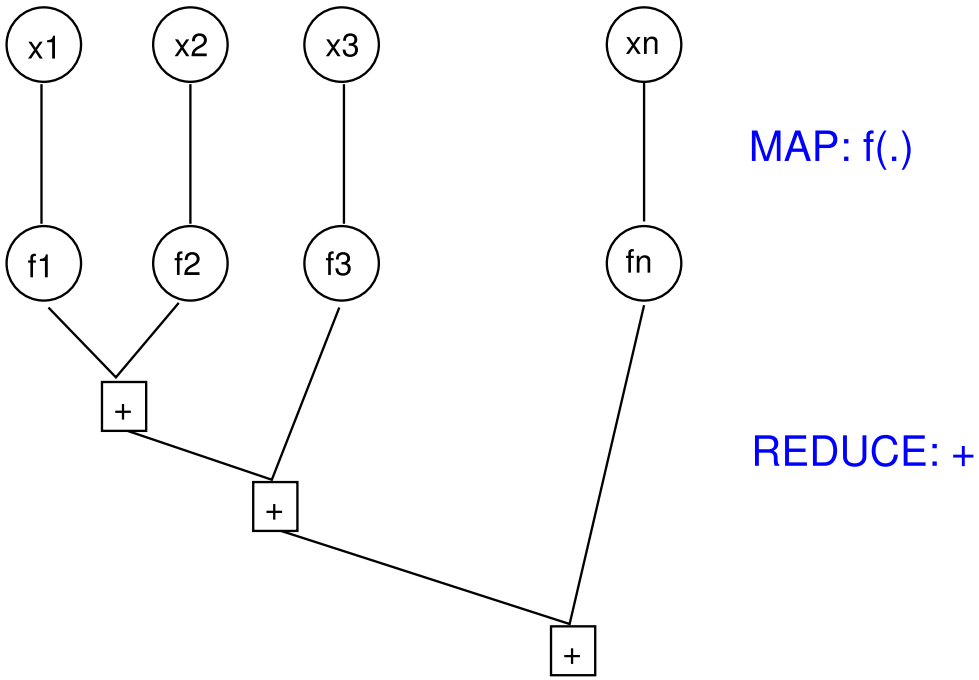
```
# foreach with a cluster backend
library(doParallel)
cl = makeCluster(4, type='PSOCK')
registerDoParallel(cl)

itx = iter(X, by='column')
pv = foreach(z=itx, .combine='c')
 %dopar% my.coxph(z, C$t.rfs, C$e.rfs)

stopCluster(cl)
```

```
# foreach with multicore backend
# not on Windows!
library(doMC)

registerDoMC(4)

itx = iter(X, by='column')

pv = foreach(z=itx, .combine='c')
     %dopar% my.coxph(z, C$t.rfs, C$e.rfs)
```

A few words about MapReduce parallelism

Analogy:

$$S = \sum_{i=1}^{n} f(x_i)$$



MapReduce for parallel computing:

- proposed by Google (2004)
- programmers get a simple API, no need to deal with remote execution, data distribution, load balancing, fault tolerance, etc..
- a scalable (both data and computation) framework
- Apache Hadoop is an open source project implementing Google's specifications
- Amazon uses Hadoop on their Elastic Cloud
- there are several packages in R that can use a Hadoop infrastructure

MapReduce: the big picture.