Parallel programming in R

Bjørn-Helge Mevik

Research Infrastructure Services Group, USIT, UiO

RIS Course Week spring 2013

Introduction

Simple example

Practical use

The end...

Background

- R is single-threaded
- ► 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
- ► As of 2.14.0, R ships with a package parallel
- ▶ R can also be compiled against multi-threaded linear algebra libraries (BLAS, LAPACK) which can speed up calculations

Today's focus is the parallel package.

Overview of parallel

- ▶ Introduced in 2.14.0
- Based on packages multicore and snow (slightly modified)
- Includes a parallel random number generator (RNG); important for simulations
- Particularly suitable for 'single program, multiple data' (SPMD) problems
- ► Main interface is parallel versions of lapply and similar
- ► Can use the CPUs/cores on a single machine (multicore), or several machines, using MPI (snow)
- ► MPI support depends on the Rmpi package (installed on Abel)

Simple example: serial

- parallel provides substitutes for lapply, etc.
- ➤ 'Silly' example for illustration: caluclate (1:100)^2

Serial version:

```
## The worker function to do the calculation:
workerFunc <- function(n) { return(n^2) }

## The values to apply the calculation to:
values <- 1:100

## Serial calculation:
res <- lapply(values, workerFunc)

print(unlist(res))</pre>
```

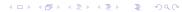
Simple example: mclapply

- ▶ Performs the calculations in parallel on the local machine
- ▶ (+) Very easy to use; no set-up
- ► (+) Low overhead
- ▶ (-) Can only use the cores of *one* machine
- ▶ (-) Uses fork, so it will not work on MS Windows

```
workerFunc <- function(n) { return(n^2) }</pre>
values <- 1:100
library(parallel)
## Number of workers (R processes) to use:
numWorkers <- 8
## Parallel calculation (mclapply):
res <- mclapply(values, workerFunc, mc.cores = numWorkers)
print(unlist(res))
                                            4 D > 4 B > 4 B > 4 B > 900
```

Simple example: parLapply

- Performs the calculations in parallel, possibly on several nodes
- ► Can use several types of communications, including PSOCK and MPI
- ► PSOCK:
 - ▶ (+) Can be used interactively
 - ► (-) Not good for running on several nodes
 - ▶ (+) Portable; works 'everywhere'
 - ► => Good for testing
- ► MPI:
 - ► (-) Needs the Rmpi package (installed on Abel)
 - (-) Cannot be used interactively
 - ▶ (+) Good for running on several nodes
 - ► (+) Works everywhere where Rmpi does
 - => Good for production



Simple example: parLapply (PSOCK)

```
workerFunc <- function(n) { return(n^2) }</pre>
values <- 1:100
library(parallel)
## Number of workers (R processes) to use:
numWorkers <- 8
## Set up the 'cluster'
cl <- makeCluster(numWorkers, type = "PSOCK")</pre>
## Parallel calculation (parLapply):
res <- parLapply(cl, values, workerFunc)
## Shut down cluster
stopCluster(cl)
print(unlist(res))
```

Simple example: parLapply (MPI)

```
simple_mpi.R:
workerFunc <- function(n) { return(n^2) }
values <- 1:100
library(parallel)
numWorkers <- 8
cl <- makeCluster(numWorkers, type = "MPI")
res <- parLapply(cl, values, workerFunc)
stopCluster(cl)
mpi.exit() # or mpi.quit(), which quits R as well
print(unlist(res))</pre>
```

Running:

```
mpirun -n 1 R --slave -f simple_mpi.R
```

Note: Use R >= 2.15.2 for MPI, due to a bug in earlier versions of parallel.

Preparation for calculations

- Write your calculations as a function that can be called with lapply
- ► Test interactively with lapply serially, and mclapply or parLapply (PSOCK) in parallel
- Deploy with mclapply on single node or parLapply (MPI) on one or more nodes
- ► For parLapply, the worker processes must be prepared with any loaded packages with clusterEvalQ or clusterCall.
- ► For parLapply, large data sets can be exported to workers with clusterExport.

Extended example

(Notes to self:)

- Submit jobs
- Go through scripts
- ► Look at results

Efficiency

- ► The time spent in each invocation of the worker function should not be too short
- ▶ If the time spent in each invocation of the worker function vary very much, try the load balancing versions of the functions
- Avoid copying large things back and forth:
 - Export large datasets up front with clusterExport (for parLapply)
 - ▶ Let the values to iterate over be indices or similar small things
 - Write the worker function to return as little as possible
- ► Reduce waiting time in queue by not asking for whole nodes; if possible, use --ntask instead of --ntasks-per-node + --nodes.



Other topics

There are several things we haven't touched in this lecture:

- Parallel random number generation
- ► Alternatives to *apply (e.g. mcparallel + mccollect)
- Lower level functions
- Using multi-threaded libraries
- Other packages and tecniques

Resources:

- ► The documentatin for parallel: help(parallel)
- ▶ The book *Parallel R*, McCallum & Weston, O'Reilly
- ► The HPC Task view on CRAN: http://cran.r-project.org/web/views/ HighPerformanceComputing.html