Parallel R

ANF R

Vincent Miele

CNRS

07/10/2015
Thinking parallel

Context
Principles
Traditional paradigms and languages
Parallel R - the foundations
   embarrassingly parallel computations in R
   the snow heritage
   the multicore heritage
   the Rmpi heritage (in brief)
Parallel R - the easy way

The parallel package
The foreach+doParallel packages

Parallel R - the hard way
Load balancing
Amdahl's law: towards the best speedup
Pseudo-random number generation
openMP

Parallel R - for real
Others things...
Thinking parallel

Context

Large computing problems in the era of high-throughput data

- huge data size
- wide solution spaces (combinatorics...)
- expensive algorithms (MCMC...)
- (memory requirements cannot be met by the memory of a single system)
Thinking parallel

Context

Static clock speed for a single CPU* around 3.4 Ghz (beyond, the CPU melt)
*(processing unit)

Due to the rise of multi-core machines, the computational power per die has still been increasing (Moore’s law)
Moreover, clusters are widely available

▶ before 2003: expecting more power
▶ after 2003: thinking parallel
Parallel algorithms design and implementation is the way to take advantage of multiprocessors architecture. (see Introduction to parallel computing, Gramma et al, Addison Wesley)

1. decomposition into tasks (indivisible units executed in parallel) of various size
2. listing the dependencies between tasks and evaluate their size
3. mapping tasks to process: scheduling/synchronizing to respect dependencies, avoid waiting time and minimize total time
4. distributing the input/output
5. managing access on common data
Thinking parallel

Traditional paradigms and languages

On a *shared memory* computer (like multi-cores), communication is implicit since all the memory is accessible to all the processes.

- From low (threads) to medium (openMP, Intel TBB) to high level interfaces (Python multiprocessing).

On a *distributed memory* computer (like clusters), the programmer is responsible for refactoring his code and add explicit operations for managing concurrency, assuming a partitionned address space.

- Low-level MPI standard.
Plan

Thinking parallel
  Context
  Principles
  Traditional paradigms and languages

Parallel R - the foundations
  embarrassingly parallel computations in R
  the snow heritage
  the multicore heritage
  the Rmpi heritage (in brief)

Parallel R - the easy way
  The parallel package
  The foreach+doParallel packages

Parallel R - the hard way
  Load balancing
  Amdahl's law : towards the best speedup
  Pseudo-random number generation
  openMP

Parallel R - for real

Others things...
Running large chunks of **independent** computations in parallel

≡ **coarse grain** parallelism

≡ **embarrassingly parallel**

Basic model is:

1. a *master* R process
2. starting up *m* worker processes (not threads)
3. splitting and sending the tasks along with the associated data
4. waiting for all the workers to complete their tasks
5. the *master* R collects the results
6. shut down the worker processes
Running large chunks of **independent** computations in parallel

≡ **coarse grain** parallelism

≡ *embarrassingly parallel*

Basic model is:

1. a *master* R process
2. starting up *m worker* processes (not threads)
3. splitting and sending the tasks along with the associated data
4. waiting for all the workers to complete their tasks
5. the *master* R collects the results
6. shut down the worker processes

What about the communication between *master* and *workers*?
Parallel R - the foundations

the snow heritage

"Simple Networks Of Workstations"

10 years of activity

on Linux, Mac OS X and Windows

via system("Rscript") or similar to launch a new process with an identical R installation

uses different transport mechanisms to communicate, in particular sockets (on a single machine) and MPI (on a cluster, via package Rmpi)

But, warning, network traffic can lead to overheads...
Parallel R - the foundations

the multicore heritage

- more recent (2009) but based on well established concepts
- only on Linux and Mac OS X
- via fork system call that creates complete copy of the master process (only the PID is different)
- copied workers will share memory pages with the master until modified so forking is very fast (copy-on-write mechanism)

Implicit communication through the shared memory
Parallel R - the foundations

the Rmpi heritage (in brief)

Rmpi

R bindings above MPI

the same code is executed by multiple processes, with flags to assign tasks to workers

If I am process 0 {do something; send data to others} else {wait data from 0; do something else}

explicit communications ≡ fine grain between processes, i.e. the programmer need to implement the data transfers

- which process is sending? which process is going to receive?
- where is the data to be sent? what kind of data? how much?
- where should the data be left on the receiving process?

A low-level paradigm with a high-level language... tortuous!
Plan

Thinking parallel
  Context
  Principles
  Traditional paradigms and languages

Parallel R - the foundations
  embarrassingly parallel
  computations in R
  the snow heritage
  the multicore heritage
  the Rmpi heritage (in brief)

Parallel R - the easy way
  The parallel package
  The foreach+doParallel packages

Parallel R - the hard way
  Load balancing
  Amdahl’s law: towards the best speedup
  Pseudo-random number generation
  openMP

Parallel R - for real

Others things...
Package parallel was first included in R 2.14.0. It builds on the work done for CRAN packages multicore (Urbanek, 2009–present) and snow (Tierney et al., 2003–present) and provides drop-in replacements for most of the functionality of those packages, with integrated handling of random-number generation.
The parallel package

snow-like

- makeCluster(..., type="PSOCK") + stopCluster
- parApply
- clusterEvalq + clusterExport

TP Exercise(s) 1+2

ooo both stdout() and stderr() of the workers are redirected, by default being discarded but they can be logged using the outfile option
Parallel R - the easy way

The parallel package

multicore-like

- `mclapply`:
  - `stdout()` and `stderr()` in parallel (could lead to conflicts and strange words/sentences)
  - You can use `mclapply` via `parLapply` using the fork backend. `parLapply` is the most general interface (but warning of data copies!)

- `makeCluster(., type="FORK")` + `stopCluster`

- `parLapply`, `parApply`...

TP Exercise(s) 1+2

- `mcparallel` (a submit operation) and `mcollect` (a wait/check operation)

TP Exercise(s) 3
Parallel R - the easy way

The parallel package

mpi-like

- `makeCluster(..., type="MPI")` + `stopCluster`

  - to take advantage of clusters and to get large distributed RAM
  - package Rmpi must be installed (and conseq. MPI on your system)
  - associated with a batch queueing system

Complicated... probably more appropriate to go to a low level language (and use MPI)
Parallel R - the easy way

The *foreach*+*doParallel* packages

doParallel acts as an interface between foreach and the parallel package of R

doParallel integrates the foreach flexibility and the *combine* possibility

doParallel enables *nested foreach loops*

TP Exercise(s) 4

*foreach* versus *apply*: a troll?
Plan

Thinking parallel
  Context
  Principles
  Traditional paradigms and languages
Parallel R - the foundations
  embarrassingly parallel computations in R
  the snow heritage
  the multicore heritage
  the Rmpi heritage (in brief)
Parallel R - the easy way
  The parallel package
  The foreach+doParallel packages
Parallel R - the hard way
  Load balancing
  Amdahl's law : towards the best speedup
  Pseudo-random number generation
  openMP
Parallel R - for real
  Others things...
How to schedule the tasks (i.e. assign the data to be treated)? A very difficult problem!

- **Static and blind** $\equiv$ roughly dividing by the number of processes
- **Static but optimized** $\equiv$ when knowing the task expected duration, using ad-hoc optimization algorithms (Longest processing time first (LPT) rule (Graham, 1966))
- **Dynamic** $\equiv$ sending chunks of tasks to unloaded processes, one by one
Parallel R - the hard way

Load balancing

- `mclapply(..., mc.preschedule = FALSE)`
- `parLapplyLB`

`ooo` warning, a process is created each time (communication) : possible overhead

Better version : `parLapplyLB`

TP Exercise(s) 5+6
Amdahl’s law: towards the best speedup

The theoretical maximum speedup is given by \( \frac{1}{\alpha + (1 - \alpha)/c} \), where \( \alpha \) is the fraction of time spent in the sequential part and \( c \) the number of processes.

If \( \alpha = 10\% \) then the maximum speedup is 10 even if \( c \to \infty \)
What could prevent you from reaching the Amdahl’s law?

- all operations that leads to starting R processes may be expensive (see section “Parallel R - for real”)
- for slow-like functions, the cost of data transfer (especially on clusters) may be prohibitive, so transfer no more than the required data:

The key point is the environments in R!

All that belongs to `.GlobalEnv` needs to be explicitely transferred with `clusterExport`.

TP Exercise(s) 7
Parallel R - the hard way

Amdahl’s law: towards the best speedup

What could prevent you from reaching the Amdahl’s law?

- All operations that leads to starting R processes may be expensive (see section “Parallel R - for real”)
- For slow-like functions, the cost of data transfer (especially on clusters) may be prohibitive, so transfer no more than the required data:

The key point is the environments in R!
All that belongs to .GlobalEnv needs to be explicitly transferred with clusterExport.

TP Exercise(s) 7

All that belongs to any other environments is serialized (copied as a whole). When a function is defined in a block (typically a method of a S4 class), it has its own environment... so all the data defined inside a function are serialized (even if not necessary)... so move the function to .GlobalEnv! And consider that the parameters of a function are serialized.

TP Exercise(s) 8
Worker processes might get the same seed because a workspace containing `.Random.seed` was restored or the random number generator has been used before forking...

Package `parallel` contains an implementation of the ideas of L’Ecuyer et al. (2002) that is more efficient than R’s default “Mersenne-Twister” RNG: `RNGkind("L’Ecuyer-CMRG")`
It is recommended to recode hotspot R functions (see section “Optimizing R code”) in C/C++/Fortran. Moreover, it is possible to introduce shared memory parallelism with openMP standard

widely supported and documented (see openMP)
adding openMP directives to help to compiler to parallelize (in particular loops)
requires a specific Makevars in pkg/src to add the -fopenmp compiler flag
Plan

Thinking parallel
   Context
   Principles
   Traditional paradigms and languages
Parallel R - the foundations
   embarrassingly parallel
   computations in R
   the snow heritage
   the multicore heritage
   the Rmpi heritage (in brief)
Parallel R - the easy way
The parallel package
The foreach+doParallel packages

Parallel R - the hard way
   Load balancing
   Amdahl’s law: towards the best speedup
   Pseudo-random number generation
   openMP

Parallel R - for real
Others things...

**DEMO 1** (*cghseg* in action)
Thinking parallel
  Context
  Principles
  Traditional paradigms and languages
Parallel R - the foundations
  embarrassingly parallel
  computations in R
  the snow heritage
  the multicore heritage
  the Rmpi heritage (in brief)
Parallel R - the easy way
  The parallel package
  The foreach+doParallel packages
Parallel R - the hard way
  Load balancing
  Amdahl’s law : towards the best speedup
  Pseudo-random number generation
  openMP
Parallel R - for real
  Others things...
parallel

Others things...

pbdR?
RcppParallel?
http://www.hpl.hp.com/research/systems-research/R-workshop