UNIVERSITÉ LUMIÈRE LYON 2

Parallel Computing for Data Science

2 – Good practices for coding

Jairo Cugliari

March 15, 2016

Plan

Plan

Best Practices for Scientific Computing[1]

1. Write programs for people, not computers
2. Automate repetitive tasks
3. Use the computer to record history
4. Make incremental changes
5. Use version control
6. Don't repeat yourself (or others)
7. Plan for mistakes
   - Defensive programming
   - Write and run tests
   - Use a variety of oracles
   - Turn bugs into test cases
   - Use a symbolic debugger
8. Optimize software only after it works correctly
9. Document design and purpose, not mechanics
10. Collaborate

1. G. Wilson et al. (2013) Download arXiv preprint

Exemple : find runs of consecutive 1s in 0-1 vectors

The problem

- Input vectors : $(1, 0, 0, 1, 1, 1, 0, 1, 1)$
- Function `findruns(vector, k)` should return the indexes of runs larger than k
- E.g. `findruns((1,0,0,1,1,1,0,1,1), 2)` should return (4, 7, 8)

(Very ugly) code

```
joe=function(x,k){
n=length(x)
r=NULL
for(i in 1:(n-k)) if(all(x[i:i+k-1]==1)) r<-c(r,i)
r
}
```

Make your code readable

Meaningful names, comments, correct code style & syntax highlight

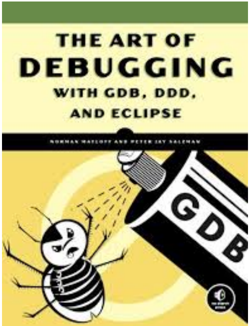
```r
## v1. Serial code ####
findruns <- function(x,k) {
  n <- length(x)
  runs <- NULL
  for (i in 1:(n - k)) {
    if (all(x[i:i + k - 1] == 1)) runs <- c(runs, i)
  }
  return(runs)
}
```

Plan

Debugging

> *Beware of bugs in the above code ; I have only proved it correct, not tried it.*
>
> *—Donald Knuth, pioneer of computer science*

- The essence of debugging : the Principle of Confirmation
- Start small
- Debug in a modular, top-down manner
- Use a debug tool ! (example with Rstudio)



Plan

1. Good Practices for Scientific Computing
2. Debugging
3. Optimize your code

Before any optimization ...

Ask yourself these questions

- Do you want your code to be readable ?
- Do you want your code to be sharable ?
- Do you have other thing to do ?

Get a bigger computer !

- RAM is cheaper and cheaper
- Amazon rents virtual machines on the web

1. FasteR ! HigheR ! StrongeR !, N. Ross, http://www.noamross.net/blog

Exemple : mutual web outlinks

The problem : analyzis of Web traffic

- How often two Web sites have links to the same third site ?
- Data : outlink information for n Web pages.
- Aim : find the mean number of mutual outlinks per pair of sites
- Similarity with many statistical methods (e.g. Kendall's τ, $U-$statistic family, ...).
- Need to compute some quantity g for each pair of observations, then sum all those values

Pseudocode

```
sum = 0.0;
for i = 1, ..., n - 1 do
    for j = i + 1, ..., n do
        sum = sum + g(obs.i, obs.j);
    end
end
return mean = sum / (n * (n - 1) / 2)
```

Serial code (1st. version)

```r
## v1. Serial code ####
mutoutser1 <- function (links) {
  nr <- nrow(links)
  nc <- ncol(links)
  tot <- 0

  for (i in 1:(nr - 1)) {
    for (j in (i + 1):nr) {
      for (k in 1:nc)
        tot <- tot + links[i , k] * links[j , k]
    }
  }
  tot / nr
}
```

Serial code (2nd. version add vectorization)

```
## v2. Serial code + Vectorization ####
mutoutser2 <- function (links) {
  nr <- nrow(links)
  nc <- ncol(links)
  tot <- 0

  for (i in 1:(nr - 1)) {
    tmp <- links[(i + 1):nr, ] %*% links[i , ]
    tot <- tot + sum(tmp)
  }
  tot / nr
}
```

Timings

Test on simulated link matrix of size 500.

Strategy	user	system	elapsed
Serial (only loops)	117.220	0.000	115.198
Serial + Vectorization	0.000	0.000	1.907

Byte Code Compilation

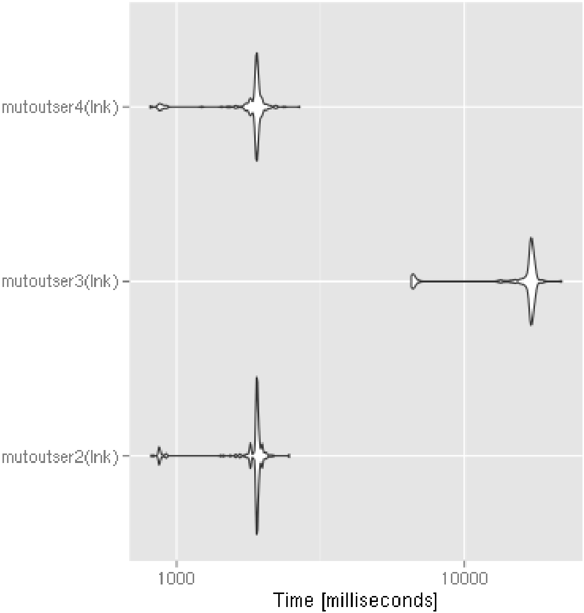
```
library(compiler)

mutoutser3 <- cmpfun(mutoutser1)
mutoutser4 <- cmpfun(mutoutser2)
compare <- microbenchmark(mutoutser2(lnk), mutoutser3(lnk),
                          mutoutser4(lnk), times = 200)

autoplot(compare)  # Plot benchmark results
```

Timings (with byte code compilation)

Test on simulated link matrix of size 500.

Strategy	user	system	elapsed
Serial (only loops) (v1)	117.220	0.000	115.198
Serial + Vectorization (v2)	0.000	0.000	1.907
v1 (with compilation)	19.284	0.000	17.089
v2 (with compilation)	0.000	0.000	1.898

Benchmark results



A gentle introduction to parallel computing in R

- Choice of Parallel Tool : `snow`, `multicore`, `foreach`, `Rmpi`
- `parallel` combines `snow` and `multicore` and is on the base of `R`
- `snow` part of `parallel` has larger architecture support (in particular for MS Windows)

- How snow works
  - scatter : The manager breaks the desired computation into chunks, and sends ("scatters") the chunks to the workers.
  - chunk computation : The workers then do computation on each chunk, and send the results back to the manager.
  - gather : The manager receives ("gathers") those results, and combines them to solve the original problem

Parallel version

```r
doichunk <- function(ichunk) {
  tot <- 0
  nr  <- nrow(lnks) # lnks global at worker

  for(i in ichunk) {
    tmp <- lnks[(i + 1):nr , ] %*% lnks[i , ]
    tot <- tot + sum(tmp)
  }
  tot
}

mutoutpar <- function(cls, lnks) {
  require(parallel)

  nr <- nrow(lnks)      # lnks global at worker
  clusterExport(cls, "lnks")
  ichunks <- 1:(nr - 1) # each "chunk" has only 1 value of i, for now
  tots <- clusterApply(cls, ichunks, doichunk)
  Reduce(sum, tots ) / nr
}
```

Timings

Test on simulated link matrix of size 500.

Strategy	user	system	elapsed
Serial (only loops)	77.000	0.004	76.980
Serial + Vectorization	1.004	0.000	1.001
Parallel (2 cores)	0.116	0.032	0.914
Parallel (8 cores)	0.128	0.016	0.666

Speepup factors for parallel wrt Serial + Vectorization are

- $1.001/0.914 = 1.09 < 2\,!$
- $1.001/0.666 = 1.503 < 8\,!\,!\,!$

Speedups are not so impresive

- several sources of overhead
- hyperthreading
- load balancing

```
  1  [|||||||||||||||||||||||||100.0%]   7  [|||||||||||||||||||||||||100.0%]    13 [|||||||||||||||||||||||||100.0%]   19 [|||||||||||||||||||||||||100.0%]
  2  [|||||||||||||||||||||||||100.0%]   8  [|||||                     14.7%]    14 [|||||||||||||||||||||||||100.0%]   20 [|||||||||||||||||||||||||100.0%]
  3  [                            0.0%]   9  [|||||||||||||||||||||||||100.0%]    15 [|||||||||||||||||||||||||100.0%]   21 [|||||||||||||||||||||||||100.0%]
  4  [|||||||||||||||||||||||||100.0%]   10 [|||||||||||||||||||||||||100.0%]    16 [|||||||||||||||||||||||||100.0%]   22 [|||||||||||||||||||||||||100.0%]
  5  [|||||||||||||||||||||||||100.0%]   11 [                         0.0%]     17 [|||||||||||||||||||||||||100.0%]   23 [|||||||||||||||||||||||||100.0%]
  6  [|||||||||||||||||||||||||100.0%]   12 [||||                      11.3%]    18 [|||||||||||||||||||||||||100.0%]   24 [|||||||||||||||||||||||||100.0%]
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||8160/64354MB]   Tasks: 3003, 309 thr; 21 running
  Swp[|||||||||                                            1040/7812MB]   Load average: 16.70 7.13 2.81
                                                                          Uptime: 155 days(!), 06:41:30

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
29876 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.34 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29765 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.23 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29742 jairo      20   0  212M   98M  4980 R 100.  0.2  1:17.35 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29774 jairo      20   0  212M   98M  4976 R 100.  0.2  1:17.26 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29807 jairo      20   0  212M   98M  4976 R 100.  0.2  1:17.25 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29826 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.34 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29816 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.12 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29701 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.29 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29712 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.29 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29797 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.29 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29691 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.31 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29788 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.07 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29681 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.35 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29753 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.10 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29731 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.18 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29722 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:16.79 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29865 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.10 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29854 jairo      20   0  212M   98M  4980 R 99.0  0.2  1:17.23 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29835 jairo      20   0  212M   98M  4976 R 100.  0.2  1:17.17 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
29845 jairo      20   0  212M   98M  4976 R 99.0  0.2  1:17.12 /usr/lib/R/bin/exec/R --slave --no-restore -e parallel:::.slaveRSOCK() --args MASTER=localhost PORT=11754
28395 jairo      20   0  913M  651M  6792 S  0.0  1.0  0:09.42 /usr/lib/R/bin/exec/R

EnterDone  EscClear   Filter: exec
```

Resorting to C

```c
#include <omp.h>
#include <R.h>

int tot;  // grand total of matches, over all threads

// processes row pairs (i,i+1), (i,i+2), ...
int procpairs(int i, int *m, int n)
{ int j,k,sum=0;
  for (j = i+1; j < n; j++) {
    for (k = 0; k < n; k++)
    // find m[i][k]*m[j][k] but remember R uses col-major order
    sum += m[n*k+i] * m[n*k+j];
  }
  return sum;
}
```

```c
void mutlinks(int *m, int *n, double *mlmean)
{ int nval = *n;
  tot = 0;
  #pragma omp parallel
  {  int i,mysum=0,
     me = omp_get_thread_num(),
     nth = omp_get_num_threads();
     // in checking all (i,j) pairs, partition the work accordin
     // this thread me will handle all i that equal me mod nth
     for (i = me; i < nval; i += nth) {
       mysum += procpairs(i,m,nval);
     }
    #pragma omp atomic
    tot += mysum;
  }

  int divisor = nval * (nval - 1) / 2;
  *mlmean = ((float) tot)/divisor;
}
```

Compilation and usage in R

Compile in bash

```
gcc -std=gnu99 -fopenmp -I/usr/share/R/include \
  -fpic -g -O2 -c romp.c -o romp.o
gcc -std=gnu99 -shared -o romp.so romp.o       \
   -L/usr/lib/R/lib -lR -lgomp
```

Call from R

```
dyn.load("romp.so")
Sys.setenv(OMP_NUM_THREADS = 4)
n <- 500
m <- matrix(sample(0:1,n^2,replace=T),nrow=n)
system.time(z <- .C("mutlinks", as.integer(m),
                    as.integer(n), result = double(1)))
```