Tutorial on Parallel Programming in R

Hana Ševčíková

Working Group on Model-Based Clustering, 02-04-05
University of Washington
hana@stat.washington.edu

http://www.stat.washington.edu/hana
Setup

Requirements:

- Hardware cluster
- PVM (MPI)
- RPVM (Rmpi)
- R packages for higher level parallel programming: snow + snowFT
- R packages for distr. random number generation: rlecuyer, rsp rng
Setting environment variables in your .cshrc (.bashrc,...) file:

```bash
setenv PVM_ROOT /usr/lib/pvm3
setenv PVM_ARCH LINUX
set path=( $path $PVM_ROOT/lib
            $PVM_ROOT/lib/$PVM_ARCH
            $PVM_ROOT/bin/$PVM_ARCH ) # in 1 line
setenv PVM_RSH /usr/bin/ssh
setenv PVM_EXPORT LD_LIBRARY_PATH:PATH
```

Check:

```
pvm
pvm> quit
```
RPVM Setup

```r
startpvm.R
library(rpvm)
hostfile <- paste(Sys.getenv("HOME"),
                   "/.xpvm_hosts", sep="")
.PVM.start.pvmd(hostfile)
.PVM.config()

.xpvm_hosts (in $HOME directory)
  * ep=/net/home/hana/lib/rpvm  # optional,
    # dir where slaveR.sh resides, if local
  mos1.csde.washington.edu
  mos2.csde.washington.edu
```
Set the environment variable `R_LIB` to directory with your local packages. E.g. in `.cshrc`:
```
setenv R_LIBS /net/home/hana/lib/inst
```
Halt your PVM deamon (`pvm> halt`, or in XPVM).
Run `startpvm.R`.
Check result in XPVM.
SNOW: Simple Network of Workstations

(L. Tierney, A. Rossini, M. Na Li, 2003)

- higher level framework for simple parallel jobs
- communication: via socket, rpvm or rmpi
- based on master-slave model
- one call creates the cluster \( \text{makeCluster}(\text{size}) \)
- automatic handling of parallel random number generator – rlecuyer, rsprng
  \( \text{clusterSetupRNG}(\ldots) \)
- one call for repeated evaluation of an arbitrary function on the cluster
SNOW Example

Execute function \texttt{fun} 10 times on 5 nodes

1. \texttt{clusterApply (cl, rep(2,5), fun)}

   - runtime depends on the slowest node, results reproducible

2. \texttt{clusterApplyLB (cl, rep(1,10), fun)}

   - faster but not reproducible
snowFT: Fault Tolerant SNOW

(H. Ševčíková, A. Rossini, 2004)

- Built on SNOW.
- Load balancing AND reproducibility: one RN stream associated with one replicate.
- Fault tolerance: failure detection and recovery in master’s waiting time.
- Computation transparency (define a print function, file .proc, .proc_fail).
- Dynamic cluster resizing (file .clustersize).
- Easy to use.
snowFT: Usage

Only one function needed for all features.

`performParallel (count, vector_of_length_n, fun, ...)`

- creates a cluster of size `count`
- [initializes each node with a given function]
- [initializes RNG (rlecuyer or rsprng)]
- performs the actual computation on the cluster (`n × fun`)
- [performs aftermath operations on each node]
- stops the cluster and returns a list of results
Programming

- Implement the peace of code to be run in parallel as a separate function.
- Make sure that all user-defined functions that are called on slave nodes can be seen by slave.

```r
initfct <- function(){
  source(file_with_local_functions.R)
}
performParallel(..., initfun=initfct, ...)

fun <- function(...) {
  localfun <- function(...) {...}
  : 
  a <- localfun (...)
}```
Tips

- Integrate a sequential and a parallel version of your code in one program.

```r
myprogram <- function(..., parallel=1,...) {
  if (parallel > 1) {
    require(snowFT)
    res <- performParallel(parallel, 1:n,
                          fun, myargs, ...)
  } else {
    res <- list()
    for (i in 1:n)
      res <- c(res, list(fun(i, myargs)))
  }
}
```
Always check the output of `performParallel`. It may contain error messages from slaves.

Check the behavior of your program in XPVM.

Be aware of the total load of the system!!! *(mosmon)*
- There are other people using it.
- No performance gain by adding slaves if no. of processes > no. of processors.
Example 1

Estimation of fractal dimension (via variogram) of time series of size $n$ repeated $r$ times.

$n = 50000, r = 100$


Sequential version: 142.2s
Parallel version (6 processors): 29.0s
Speedup: 4.9
Example 2

Variable selection for model-based clustering (Nema Dean).

1. First step: Select the best variable in terms of univariate clustering.
   \[ \Rightarrow \text{Run } EMclust \text{ for } i = 1, \ldots, d_1. \]

2. Second step: Select the best variable in terms of bivariate clustering.
   \[ \Rightarrow \text{Run } EMclust \text{ for } i = 1, \ldots, d_2. \]

3. Iterate:
   (a) Addition step: Propose a variable for adding.
       \[ \Rightarrow \text{Run } EMclust \text{ for } i = 1, \ldots, d_3. \]
   (b) Removal step: Propose a variable for removal.
       \[ \Rightarrow \text{Run } EMclust \text{ for } i = 1, \ldots, d_4. \]
Example 2 (cont.)

```
performParallel(min(P, d_1), 1:d_1, FirstStep, ...)
           ↓
           sync
           ↓
performParallel (min(P, d_2), 1:d_2, AddStep, ...)
           ↓
           sync
           ↓
performParallel (min(P, d_3), 1:d_3, AddStep, ...)
           ↓
           sync
           ↓
performParallel (min(P, d_4), 1:d_4, RmStep, ...)
```
Example 2 (cont.)

Data with 15 variables:

\[ d_1 = 15 \rightarrow d_2 = 14 \rightarrow d_3 = 13 \rightarrow d_4 = 2 \rightarrow \text{stop} \]

Sequential version: \(418.8s\)
Parallel version (6 processors): \(90.0s\)
Speedup: \(4.7\)

Code at www.stat.washington.edu/hana/code/example2.R
Conclusions

Parallel programming in R is VERY EASY !!!