Parallel R and ARCHER2

Adrian Jackson

a.jackson@epcc.ed.ac.uk

Reusing this material

Partners

Parallel R

- Different motivations for parallelising R
  - Too much computational work
    - My program takes too long
  - Too much data to process
    - My program is quick enough, but I've too much data to process
  - Data is too large
    - My data set is too big to fit into memory
- Wide range of options:
  - https://cran.r-project.org/web/views/HighPerformanceComputing.html
  - Implicit parallelism
  - Explicit parallelism
  - Applications
  - Batch system integration, hardware resources, distributed computing integration
  - Native code interfaces
  - Big memory

Parallel options for R

- Shared memory parallelisation
  - Trivially parallel
    - (l)apply
    - Map
  - Loop parallelism
    - foreach
  - Threads
    - Rdsm
- Distributed memory parallelisation
  - parallel
  - snow
  - Rmpi
  - pbdR
  - foreach

Automatic parallelism

- R has some parallelism built in

- BLAS libraries and other underlying functionality already using shared memory parallelism, i.e.:

```
A <- matrix( rnorm(n*n), ncol=n, nrow=n )
B <- matrix( rnorm(n*n), ncol=n, nrow=n )
C <- A %*% B

> library("data.table")
data.table 1.14.2 using 128 threads (see ?getDTthreads).  Latest news: r-
datatable.com
```

- Can control number of workers on ARCHER2 using

```
export OMP_NUM_THREADS=
i.e.
export OMP_NUM_THREADS=2
…
> library("data.table")
data.table 1.14.2 using 2 threads (see ?getDTthreads).  Latest news: r-
datatable.com
```

- By default will use all available workers
    - Can be the wrong thing to do

Trivial trivial parallelism

- Task farms
  - Same code executed independently
  - Different initial conditions or different data sets
  - Slurm array jobs can support this

```
args <- commandArgs(TRUE)
set.seed(args[1])
data <- read.csv('dataset.csv')
result <- kmeans(data, centers=10, nstart=100)
print(result)

Rscript kmeans-trivial.R 1 > 1_output.dat
```

- Requires manual processing of the data
- Requires some care if using random number generators

Trivial parallelism

- Split, Apply, Combine
  - Partition data into groups to be worked on
  - Apply operations to each group
  - Combine results with some form of reduction at the end
- Requires making some workflows more complex to enable parallelisation:

```
data <- read.csv('dataset.csv')
result <- kmeans(data, centers=4, nstart=100)
print(result)
-----------------------------------------------------------
data <- read.csv('dataset.csv')
parallel.function <- function(i) {
    kmeans( data, centers=4, nstart=i )
}
results <- lapply( c(25, 25, 25, 25), FUN=parallel.function )

temp.vector <- sapply( results, function(result) { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]
```

mcapply

```
library(parallel)
data <- read.csv('dataset.csv')
parallel.function <- function(i) {
    kmeans( data, centers=4, nstart=i )
}
results <- mclapply( c(25, 25, 25, 25), FUN=parallel.function )

temp.vector <- sapply( results, function(result) { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]
print(result)
```

mcparallel mccollect

```
library(parallel)
p <- mcparallel(1:10)
q <- mcparallel(1:20)
# wait for both jobs to finish and collect all results
res <- mccollect(list(p, q))
```

```
library(parallel, quiet=TRUE)
source("data/airline/read_airline.R")
jan2010 <- read.airline("data/airline/airOT201001.csv.gz")
unique.planes <- mcparallel( length( unique( sort(jan2010$TAIL_NUM) ) ) )
median.elapsed <- mcparallel( median( jan2010$ACTUAL_ELAPSED_TIME, na.rm=TRUE ) )
ans <- mccollect( list(unique.planes, median.elapsed) )
ans
```

foreach

```
library(foreach)
data <- read.csv('dataset.csv')
results <- foreach( i = c(25,25,25,25) ) %do% {
    kmeans( x=data, centers=4, nstart=i )
}
temp.vector <- sapply( results,
function(result)
    { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]

print(result)
```

foreach parallel

```r
library(foreach)
library(doMC)
data <- read.csv('dataset.csv')

registerDoMC(4)
results <- foreach( i = c(25,25,25,25) ) %dopar% {
    kmeans( x=data, centers=4, nstart=i )
}

temp.vector <- sapply( results, function(result)
    { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]

print(result)
```

parLapply

```
library(snow)
data <- read.csv( 'dataset.csv' )
parallel.function <- function(i) {
    kmeans( data, centers=4, nstart=i )
}
cl <- makeCluster( mpi.universe.size(), type="MPI" )
clusterExport(cl, c('data'))


results <- parLapply( cl, c(25,25,25,25), fun=parallel.function )


temp.vector <- sapply( results, function(result) { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]
print(result)


stopCluster(cl)
mpi.exit()
```

doSNOW foreach

```r
library(foreach)
library(doSNOW)
data <- read.csv('dataset.csv')
cl <- makeCluster( mpi.universe.size(), type='MPI' )
clusterExport(cl,c('data'))
registerDoSNOW(cl)
results <- foreach( i = c(25,25,25,25) ) %dopar% {
    kmeans( x=data, centers=4, nstart=i )
}


temp.vector <- sapply( results, function(result)
    { result$tot.withinss } )
result <- results[[which.min(temp.vector)]]

print(result)
stopCluster(cl)
mpi.exit()
```

snow

```
cluster <- getMPIcluster()

# Print the hostname for each cluster member
sayhello <- function()
{
  info <- Sys.info()[c("nodename", "machine")]
  paste("Hello from", info[1], "with CPU type", info[2])
}


names <- clusterCall(cluster, sayhello)
print(unlist(names))

# stopCluster will call mpi.finalize, no need for mpi.exit
stopCluster(cluster)
```

Running snow

```bash
#!/bin/bash
#SBATCH --job-name=snow
#SBATCH --nodes=2
#SBATCH --tasks-per-node=128
#SBATCH --time=0:5:0
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=z19
#SBATCH --hint=nomultithread

module load cray-R
export R_LIBS_USER=/work/z19/z19/adrianj/Rinstall
export PATH=$PATH:/work/z19/z19/adrianj/Rinstall/snow

export OMP_NUM_THREADS=1

srun RMPISNOW < ./simple_parallel_snow.R
```

More complex snow

```r
nmax = 50
nworkers <- as.numeric(Sys.getenv("SLURM_NPROCS"))

cl <- getMPIcluster()

pbday <- function(n) {
  ntests <- 100000
  pop <- 1:365
  anydup <- function(i)
  any(duplicated(sample(pop, n,replace=TRUE)))
  sum(sapply(seq(ntests), anydup)) / ntests
}
clusterExport(cl, list('pbday'))

# print the time to do nmax tests, after distributing them to the workers
system.time( x <- clusterApply(cl, 1:nmax, function(n) { pbday(n) }) )

# compute the theoretical probability for each n
prob <- rep(0.0,nmax)
probnot <- 1.0
for (i in 2:nmax) {
  probnot <- probnot*(366.0-i)/365.0
  prob[i] = 1.0 - probnot
}

# print results, comparing tests to theory
z <- cbind(x,prob)
print(z)

# always include the following to stop the cluster
stopCluster(cl)
```

```
#!/bin/bash
#SBATCH --job-name=snow
#SBATCH --nodes=1
#SBATCH --tasks-per-node=64
#SBATCH --time=0:5:0
#SBATCH --partition=standard
#SBATCH --qos=short
#SBATCH --account=z19
#SBATCH --hint=nomultithread

module load cray-R
export R_LIBS_USER=/work/z19/z19/adrianj/Rinstall
export PATH=$PATH:/work/z19/z19/adrianj/Rinstall/snow

export OMP_NUM_THREADS=1

srun RMPISNOW < ./sample_birthday_snow.R
```