Parallel_R

Cheng Ju

2/16/2015

The doParallel package is a "parallel backend" for the foreach package. It provides a mechanism needed to execute foreach loops in parallel. The foreach package must be used in conjunction with a package such as doParallel in order to execute code in parallel. The user must register a parallel backend to use, otherwise foreach will execute tasks sequentially, even when the %dopar% operator is used.

- For more information about the foreach package in R, you can visit the web: http://cran.r-project.org/web/packages/foreach/vignettes/foreach.pdf
- For more information about the arguments in doParallel, you can visit the web: http://cran.r-project.org/web/packages/doParallel/doParallel.pdf
- and for more examples, I suggest you to read http://cran.r-project.org/web/packages/doParallel/vignettes/gettingstartedParallel.pdf

The Collatz conjecture is a conjecture in mathematics named after Lothar Collatz, who first proposed it in 1937.

Take any natural number n. If n is even, divide it by 2 to get n / 2. If n is odd, multiply it by 3 and add 1 to obtain 3n + 1. Repeat the process indefinitely. The conjecture is that no matter what number you start with, you will always eventually reach 1. The property has also been called oneness

```
library(doParallel)
```

```
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
```

```
cl <- makeCluster(2)
registerDoParallel(cl)

func <- function(x) {
      n = 1
      raw <- x
      while (x > 1) {
            x <- ifelse(x%%2==0,x/2,3*x+1)
            n = n + 1
      }
      return(c(raw,n))
}
size<- 1e4
Eptime<- system.time({
      psteps <- foreach(x=1:size,.combine='rbind') %dopar% func(x)
      stopCluster(cl)
})
## Return which number will take most iteration to get 1.
psteps[which.max(psteps[,2]),]
```

```
## [1] 6171  262
```

```
Eptime
```

```
##    user  system elapsed
##   3.936   0.311   6.346
```

We can compare the running time with sequential programming. Here we can also use foreach. But we use %do% rather than %dopar%.

```
Estime<- system.time({
      ssteps <- foreach(x=1:size,.combine='rbind') %do% func(x)
})
ssteps[which.max(ssteps[,2]),]
```

```
## [1] 6171  262
```

```
Estime
```

```
##    user  system elapsed
##   5.768   0.018   5.822
```

Here is an example using doParallel and foreach package to do randorm forest.

```
library(doParallel)
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
prf.time<-system.time({
      cl <- makeCluster(4)
      registerDoParallel(cl)
      ## each cluster, we set 2500 trees
      rf <- foreach(ntree=rep(2500, 4),
                    .combine=combine,
                    .packages='randomForest') %dopar%
            randomForest(Species~., data=iris, ntree=ntree)
      stopCluster(cl)
})
## directly use sequential programming
srf.time<-system.time(
      randomForest(Species~., data=iris, ntree=10000)
)

prf.time
```

```
##    user  system elapsed
##   0.850   0.056   2.023
```

```
srf.time
```

```
##    user  system elapsed
##   0.758   0.048   0.806
```

Here is an example using doParallel and foreach package to do GLM based on bootstraped data.

```
library(doParallel)

#making cluster. Here we set the number of clusters equal to 2.
cl <- makeCluster(2)
registerDoParallel(cl)


## iris dataset
x<- iris[which(iris[,5]!='setosa'),c(1,5)]
trials<- 1000
# Parallel Computing
ptime<- system.time({
      ## icount count number of times that the iterator will fire.
      r1<- foreach(icount(trials), .combine=cbind) %dopar% {
            ## Do sample from index 1 to 100. Sample 100 times with repalcement.
            ind<- sample(100,100,replace=T)
            ## Here we use logistic model. We use glm function used sampled data
            result1<- glm(x[ind,2]~x[ind,1],family=binomial(logit))
            coefficients(result1)
      }
})
## Compared with Sequatial Computing
stime<- system.time({
      r2<- foreach(icount(trials), .combine=cbind) %do% {
            ind<- sample(100,100,replace=T)
            result1<- glm(x[ind,2]~x[ind,1],family=binomial(logit))
            coefficients(result1)
      }
})

ptime
```

```
##    user  system elapsed
##   0.399   0.041   2.182
```

```
stime
```

```
##    user  system elapsed
##   2.881   0.064   2.945
```

As you may find from some resules. If the model is too simple, like random forest and bootstraping, it may not be a good idea to use parallel programming, because it will take too much time communication with each core compared with sequential programming. However, when using more complex model like GLM, we can save a lot of time by using parallel programming.