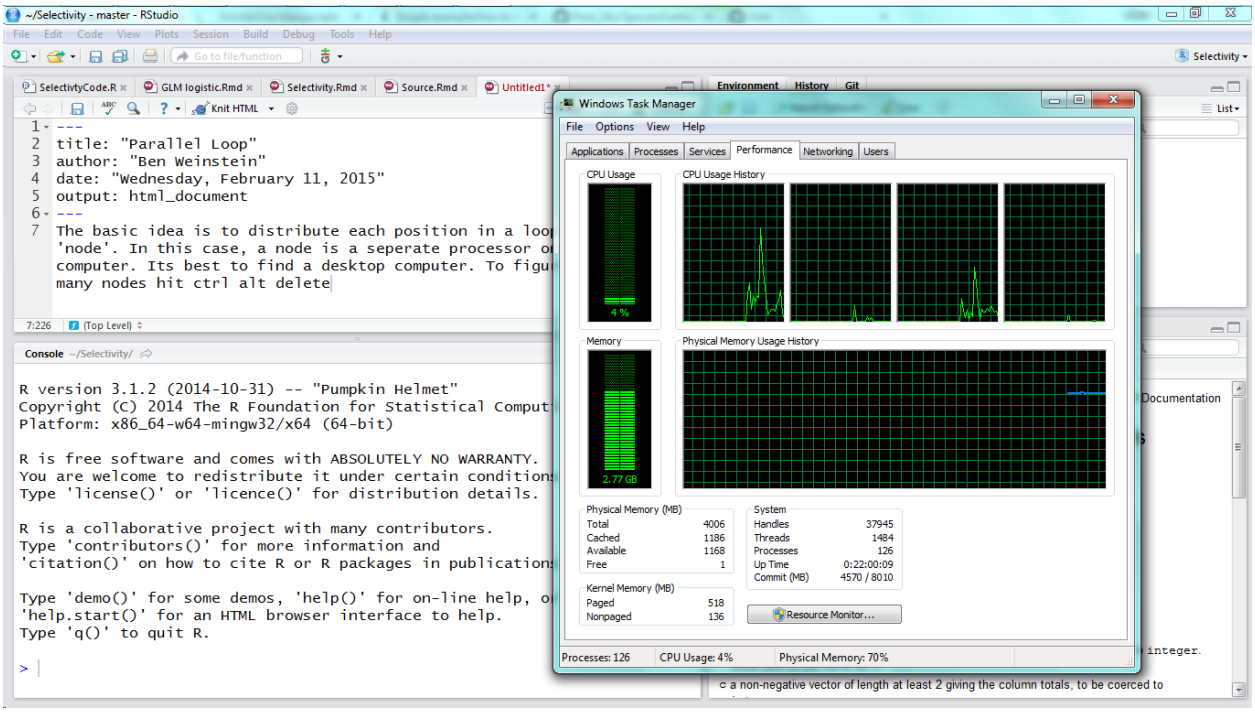
# Parallel Loop

*Ben Weinstein*

*Wednesday, February 11, 2015*

The basic idea is to distribute each position in a loop to a virtual 'node'. In this case, a node is a seperate processor on your computer. Its best to find a desktop computer. To figure out how many nodes hit ctrl alt delete and see how many panes are in the task manager, this is equal to the number of virtual nodes you can run (as a general rule of thumb).




## Simple example

Read in packages

```
library(foreach)
library(doSNOW)
```

### For Loop

```
out<-list()
for(x in 1:10){
  out[x]<-x^2
}
```

Notice how i'm using an list container to collect the results, this best mimics what foreach will do.

### Foreach loop structure NOT in parallel

```
outNP<-foreach(x = 1:10) %do% {
x^2
}
```

### Foreach loop in parallel

```
cl<-makeCluster(4,"SOCK")
registerDoSNOW(cl)
outP<-foreach(x = 1:10) %do% {
x^2
}
stopCluster(cl)
```

Okay what did that do?

```
cl<-makeCluster(4,"SOCK")
```

This makes a 'cluster' of R sessions. If you check back in on your task manager, you will literally see four versions of R open, its just that they are not visible to the user. Each has unique memory, they operate completely independently. The larger the number, the more sessions of R you create, the faster the speed up. The downside is that they all use memory. Its best to start smart small (if you have a nice desktop, start with ~10) and watch the memory spike in the task manager. If you start going over 100% memory, you will eventually crash the run.

```
registerDoSNOW(cl)
```

This just registers the cluster you made with your current R session, which is going to act as a 'master' node which sends information to the 'child' processes.

```
outP<-foreach(x = 1:10) %do% {
x^2
}
stopCluster(cl)
```

The Foreach syntax is alittle odd, but here the loop will execute 10 times and perform the action that is within the brackets. If you have a predefined function, you can add it directly.

Foreach will create an output container for you, in a list structure. See args for options to bind them together into a df and so on, but i find it easier as lists and handle postprocessing myself.

## Compare results

```
dat<-cbind(out,outNP,outP)
print(dat)
```

```
##       out outNP outP
##  [1,] 1   1     1
##  [2,] 4   4     4
##  [3,] 9   9     9
##  [4,] 16  16    16
##  [5,] 25  25    25
##  [6,] 36  36    36
##  [7,] 49  49    49
##  [8,] 64  64    64
##  [9,] 81  81    81
## [10,] 100 100   100
```

Note that for very small tasks, like the one above, parallelization is actually slower, since there is a time cost for creating, distributing and collecting results.

## Recommendation

So here's your code

```
p<-1
results<-list()
null_model_list<-c( "phylo_temp", "phylo_precip", "phylo_geo", "phylo_cost_distance", "phylo_temp_geo",
#null_model_list<-c("phylo_precip")
for (p in 1:length(null_model_list))
{
  q<-1
  result<-list()
  #quantiles<-c(0, 0.25, 0.50, 0.75, 0.85, 0.90, 0.95, 0.99, 1)
  quantiles<-c(0, 0.75, 0.95, 0.99)
  for (q in 1:length(quantiles))
  {

    result[[q]] <- full_sp_pool_analysis (as.symbol(as.character(call(null_model_list[p]))), hmat_dat, l

  }
  results[[p]] <- data.frame(do.call("rbind",result))
  safety_dat<-data.frame(do.call("rbind",results))
  write.table(safety_dat, file=null_model_list[p],row.names=F )
}
```

Which of the loops take longer? I assume the outer loop, you don't want to do nested foreach yet. So something like:

```
p<-1
null_model_list<-c( "phylo_temp", "phylo_precip", "phylo_geo", "phylo_cost_distance", "phylo_temp_geo",

#null_model_list<-c("phylo_precip")
cl<-makeCluster(8,"SOCK")
registerDoSNOW(cl)
results<-foreach (p=1:length(null_model_list)) %dopar% {
  q<-1
  result<-list()
```

```
  quantiles<-c(0, 0.75, 0.95, 0.99)
  for (q in 1:length(quantiles))
  {

    result[[q]] <- full_sp_pool_analysis (as.symbol(as.character(call(null_model_list[p]))), hmat_dat, l

  }
  results[[p]] <- data.frame(do.call("rbind",result))
  safety_dat<-data.frame(do.call("rbind",results))
  write.table(safety_dat, file=null_model_list[p],row.names=F )
  return(safety_dat)
}
#make sure to stop the cluster, if not, when you invoke it again it will make new sessions and get clut
stopCluster(cl)
```

Its best to have a return statement because its basically an in line function. Depending on how the environment is setup, you may have to manaully export packages using the .packages= argument, or specific objects that you have called outside of the scope using the .export argument. For example, i'm not 100% sure its going to know what "null_model_list" is because you defined it outside the loop, but then you call it within the loop. Your choice is to either define it in the loop or using .export, see ?foreach.

Okay i hope that helps!