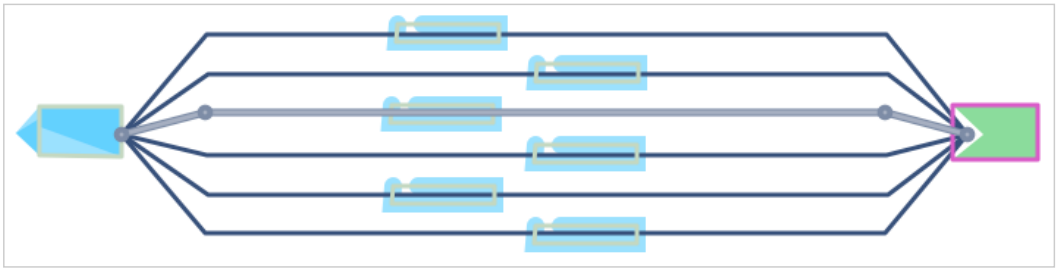
Parallel computing with package 'snowfall'

Lately I have been looking for ways to decrease the amount of time it takes me to run multiple regressions over a very large data set. There are several options that I am investigating to do this, and certainly more that I don't know of yet.

- Code more efficiently.
- Compute several operations in parallel over a two or more CPU cores.
- Tap into a network of computers, and further expand the number of CPU cores to parallelize calculations.

Because many of my computer jobs are "embarassingly parallel", the options mentioned above would immediately improve the speed I can compute (and re-compute) jobs. This post will go through an example using the CRAN package `snowfall` to parallelize a computation over several CPU cores on the same computer (bullet #2 above).



The CRAN package `snowfall` is built to make it easy to create parallel processes. I recommend taking a look at the associated vignette and tutorial.

Before beginning to use `snowfall`, do the following:

1. Upgrade to the latest version of R – as of this post version 2.14.1 (or the patched version of R-2.13.0 – available here). FYI – There is a bug in version 2.13.0 (for MS Windows 7) that prevents snowfall from operating smoothly.
2. Install the latest version of the package `snowfall` (`install.packages('snowfall', dependencies = TRUE)`)
3. Find out how many cores you have on the CPU of the machine you will be using. In my example below, I am using a machine with 8 CPU cores and running Windows 7.
4. Convert any 'for' loops into a function that you can call using `apply()`. See my previous post that outlines this process.

Using `snowfall`: A simple example

The reason I put together this post is because I couldn't easily find a 'plug'n play' code example in the existing online literature to execute the type of parallelization I wanted. Out of necessity I worked through the wrinkles and am now successfully utilizing multiple CPU cores in R. – **Note:** By default, R uses only one CPU core unless you explicitly code it to use multiple cores (as in this example).

Here is an outline of what our R code will accomplish:

1. Clear workspace.
2. Load in several dataframes needed in our calculation.
3. Define looping parameters. For my example, we want to loop through all date-hour combinations (e.g. months, days, years, hours) in a chosen date range; you could simplify this to loop through N numbers.
4. Initialize `snowfall` package and tell it the number of CPU cores we want to use.
5. Define some functions needed in our parallelization.
6. Export (using `sfExport()`) all dataframes and functions needed for the calculation to each 'slave' CPU core.
7. Using `apply`, 'loop through' all date-hour combinations with our chosen function.
8. Stop parallelization.

The outline above translates into the 'code outline' below. *Note: the example below is for illustration only. For code that can be run / tested as-is, scroll down the page to the "Another Example" section.*

```
# Clear workspace
rm(list=ls())

# load in snowfall package
require(snowfall)

# Load in datasets needed in calculation
# E.g. datasets.RData includes three dataframes: df1, df2, and df3
load('datasets.RData')

## Choose parameters here: a (month), b (day), c (hour), d (year)
## This example chooses all hours from June 1 to June 15, 2011
a = 6; b = 1:15; c = c(1:24); d = 2011

## Initialize parallel operation
```

Search this blog
Search...

Contributors

Kevin Goulding

Categories
Econometrics
Econometrics with R
Numpy
Python
R tips & tricks
Surviving Graduate Econometrics with R
TikZ for Economists
Visualizing Data with R
White Papers

Twitter feed
RT @gappy3000: This post, apparently about #julialang and #pydata, explains why #rstats has become the standard of data analysis http:// ... 3 years ago
RT @justinwolfers: "if prediction markets are really as valuable as economists think, then..more experimentation could prove worthwhile. ... 3 years ago
RT @vsbuffalo: For me the biggest victory is for statistics and empiricism. Go Nate Silver and @fivethirtyeight for a brilliant forecast ... 3 years ago
Follow @baha_kev

Tag Cloud
cluster-robust Econometrics heteroskedasticity LaTeX Numpy Parallel Computing plots Python R STATA tex TikZ

```
sfInit( parallel=TRUE, cpus=7 )

#####################################################################
## specify functions that will be used:

fun1
    # define function number 1

}

fun2
    # define function number 2
    # fun2 will use fun1

}

#####################################################################
## 'Export' functions and dataframes to all "slaves" so that
##  parallel calculations can occur simultaneously

sfExport(list=list("fun1"))
sfExport(list=list("fun2"))

sfExport('df1')
sfExport('df2')
sfExport('df3')

## call function using sfApply; will return values as a list object
 out = sfApply(expand.grid(a,b,c,d), 1,
        function(x,y,z,a) fun2(x[1],x[2],x[3],x[4]))

## stop parallel computing job
sfStop()
```

Another example

The following is an example based on the code above that will run as shown if you (1) have enough CPU cores on your computer; and (2) have the `snowfall` and `chron` packages installed in R:

```
# Clear workspace
rm(list=ls())

# load snowfall package
require(snowfall)

# load chron package
require(chron)

# Load in datasets needed in calculation
# No data sets needed in this example, so...
df1 = df2 = df3 = NULL

## Choose parameters here: a (month), b (day), c (year)
## This example chooses all days from June 1 to June 15, 2011
a = 6; b = 1:15; c = 2011

## Initialize parallel operation
sfInit( parallel=TRUE, cpus=2 )

#####################################################################
## specify functions that will be used:

fun1
    if(is.weekend(z) == TRUE) {print("It's the weekend!")}
     else {print("WORKDAY")}

}

fun2
    date = chron(julian(a,b,c))
    words = fun1(julian(a,b,c))

    list(date,words)

}

#####################################################################
## 'Export' all functions, packages, and dataframes needed
##  for to all "slaves" so that parallel calculations can
##  occur simultaneously.

# functions
sfExport(list=list("fun1"))
sfExport(list=list("fun2"))

#packages
sfLibrary(chron)

# dataframes
sfExport('df1')
sfExport('df2')
sfExport('df3')

## call function using sfApply; will return values as a list object
 out = sfApply(expand.grid(a,b,c), 1,
        function(x,y,z) fun2(x[1],x[2],x[3]))

## stop parallel computing job
sfStop()
```

All the calculations are stored in the list object "`out`". This is necessarily a trivial example; but should provide you with the confidence to utilize the method for your own parallel processing needs. Let me know if this example works for you or if any clarification is needed.

Share this:

Share

Like

Be the first to like this.

Related

Why use R? A grad student's 2 cents
In "R tips & tricks"

Surviving Graduate Econometrics with R: Fixed Effects Estimation -- 3 of 8
In "Surviving Graduate Econometrics with R"

Surviving Graduate Econometrics with R: The Basics -- 1 of 8
In "Surviving Graduate Econometrics with R"

Posted on February 21, 2012 at 4:21 pm in R tips & tricks, Surviving Graduate Econometrics with R | RSS feed | Reply | Trackback URL

Tags: Parallel Computing, R

Leave a Reply

Enter your comment here...

Tags

cluster-robust econometrics heteroskedasticity latex numpy parallel computing plots python r stata tex tikz

Archives

October 2012
February 2012
July 2011
June 2011
May 2011

Blogroll

Documentation
Plugins
Suggest Ideas
Support Forum
Themes
WordPress Blog
WordPress Planet