On the lambda (http://www.onthelambda.com/)

another blog from a data analyst – by tony fischetti

Parallel R (and air travel)

tony.fischetti@gmail.com (http://www.onthelambda.com/author/tony-fischettigmail-com/) November 13, 2013 (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/) 5 Comments (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/#comments)

My heart sinks a little when I check on my laptop in the morning and the computation I started the night before still hasn't finished. Even when the data I'm playing with isn't particularly.... large... (I'm not going to say it), I have a knack for choosing expensive algorithms late at night.

Because of my reluctance to get remote servers and tmux involved at ungodly hours, I've been exploring ways to speed up my code without too much trouble (I take the first of the Three virtues of programming (http://threevirtues.com) very seriously) and without having to translate from my native tongue: R.

- **writing idiomatic R**

  taking advantage of the way R stores numerics in contiguous blocks of RAM and uses C code for vectorized functions

- **using Rcpp or inline**

  Identify bottlenecks and replace time critical/innermost-loops with C or C++ code

- **distribute processing over machines**

  It's *relatively* easy to set up a computing cluster using Amazon Web Services and use the package snow (http://cran.r-project.org/web/packages/snow/snow.pdf), or a similar package.

- **taking advantage of multiple cores**

  Besides for writing idiomatic R (which is the subject of a future post), I've found that the easiest way to get a speedup is to use parallel processing on a single machine using the multicore (http://cran.r-project.org/web/packages/multicore/multicore.pdf) package. This usually gives me a very substantial speedup, even if the code is less than elegant, as we'll see.

Testing it:

Just for the sake of choosing a problem that exhibits combinatorial explosion (http://en.wikipedia.org/wiki/Combinatorial_explosion), let's use the multicore package to get the mean distance between all airports in the U.S.

If we think of a network of air-travel as a complete graph where the vertices are airports and a bee-line between any two airports as an edge, we want to find the average length of the edges. The number of edges in a complete graph is $\frac{n(n-1)}{2}$ where n is the number of vertices. If this looks familiar, it's because it's equivalent to the binomial coefficient $\binom{n}{2}$. Intuitively, this makes sense, since every edge connecting two vertices is a unique pairing of two airports. The fact that this number exhibits polynomial growth and that the computation of the distance between two airports does not depend on the distance between any other two, makes this a prime candidate for parallelizing.

The dataset of US airport codes and their longitude and latitude is available here (https://opendata.socrata.com/dataset/Airport-Codes-mapped-to-Latitude-Longitude-in-the-/rxrh-4cxm). There are 13,429 airport codes, which makes the total number of combinations 90,162,306. Clearly, extreme measures have to be taken to make this a tractable problem on commodity hardware.

Since the Earth isn't flat (it's not even, strictly speaking, a perfect sphere (http://en.wikipedia.org/wiki/Oblate_spheroid)) the distance between longitude and latitude degrees is not constant. Luckily, there's a great package, Imap (http://cran.r-project.org/web/packages/Imap/Imap.pdf), to handle conversion from two long/lat points to miles or kilometers.

The code:

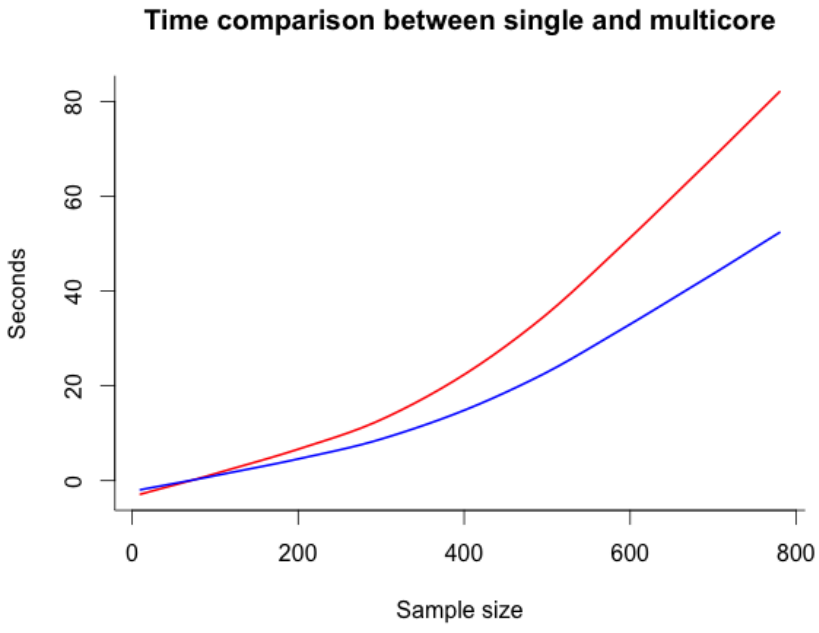
```
library(multicore)
library(Imap)

calc.distance.two.rows <- function(ind1, ind2){
  return(gdist(air.locs[ind1, 3],
               air.locs[ind1, 2],
               air.locs[ind2, 3],
               air.locs[ind2, 2], units="km"))
}

sum.of.distances <- function(a.mat){
  return(sum(apply(a.mat, 2, function(x){
                              calc.distance.two.rows(x[1], x[2])
                            })))
}

# read airport long/lat data set
air.locs <- read.csv("airportcodes.csv", stringsAsFactors=FALSE)

# read command-line args
args <- commandArgs(TRUE)

# read the number of airports to use (sample size) from the command-line
smp.size <- as.numeric(args[1])

# choose a random sample of airports
sampling <- sample((1:nrow(air.locs)), smp.size)

# shrink dataframe
air.locs <- air.locs[sampling, ]

# create a matrix of unique subsets of indices from the
# data frame that stores the airport information
combos <- combn(1:nrow(air.locs), 2)
```

```
num.of.comps <- ncol(combos)

# use single core
single <- function(){
  the.sum <- sum.of.distances(combos)
  result <- the.sum / num.of.comps
  return(result)
}

# use two cores
mult <- function(){
  half <- floor(num.of.comps/2)
  f1 <- parallel(sum.of.distances(combos[, 1:half]))
  f2 <- parallel(sum.of.distances(combos[, (half+1):num.of.comps]))
  the.sum <- sum(as.vector(unlist(collect(list(f1, f2)))))
  result <- the.sum / num.of.comps
  return(result)
}

# compare the execution times (in time elapsed)
perform <- function(){
  first <- system.time(res1 <- single())
  second <- system.time(res2 <- mult())
  cat(smp.size); cat(",first,"); cat(first[3]); cat(","); cat(res1); cat
  cat(smp.size); cat(",second,"); cat(second[3]); cat(","); cat(res2); c
}

perform()
```

Then, I wrote a wrapper program in python that compared the speeds using sample sizes from 10 to 800 in increments of ten.

The results:

The



(http://www.onthelambda.com/wp-content/uploads/2013/11/Time-comparison-between-single-and-multicore.png)

Time comparison between single and multicore execution time in seconds. The curves were smoothed using LOESS.

parallelized solution is much faster but it is not twice as fast, as one might expect. This is because, not only is there overhead involved in the process of forking and collecting the result, but part of the program (namely the reading of the dataset) is not parallelized. (For more information, check out Amdehl's Law (http://en.wikipedia.org/wiki/Amdahl%27s_law)).

Some cursory curve-fitting suggests that the single core solution's execution time is fairly well-modeled by the function $.00014(n)(n-1)$ and the dual core solution is well modeled by $.00009(n)(n-1)$. This would make the total time to completion about 7 hours and 4.5 hours, respectively.

After about 1 hour, I psychosomatically smelled melting plastic from my laptop so I called off the actual distance calculation. But repeated trials with sample sizes of 300 suggested that the sampling distribution of the sample mean (whew) had a mean of about 1,874 km and a standard deviation of 64 km (this is the standard error of the mean). This would suggest that the mean distance between any two US airports is about 1,874 km +- 125 (543 miles +- 78) with a standard deviation of around 1109 km (689 miles).

As this example shows, even a kludge-y solution that uses more than one thread can give you pretty substantial speed increases. In future posts, I plan to cover applying Rcpp to this problem and setting up a cluster to perform this calculation. In the meantime, I'll still be starting things at night and regretting it in the morning.

Footnote: If you're interested in this topic, I suggest you check out this site (http://cran.r-project.org/web/views/HighPerformanceComputing.html) from CRAN. Also, everything I know about parallel R, I learned from this book (http://shop.oreilly.com/product/0636920021421.do).

R (http://www.onthelambda.com/category/r/) high performance computing (http://www.onthelambda.com/tag/high-performance-computing/), R (http://www.onthelambda.com/tag/r/), statistics (http://www.onthelambda.com/tag/statistics/) Bookmark (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/)

PREVIOUS ARTICLE
qstats - quick and dirty statistics tool for the Unix pipeline (http://www.onthelambda.com/2013/11/05/qstats-quick-and-dirty-statistics-tool-for-the-unix-pipeline/)

NEXT ARTICLE
Compiling R from source and why you shouldn't do it (http://www.onthelambda.com/2013/11/22/compiling-r-from-source-and-why-you-shouldnt-do-it/)

5 RESPONSES

Pingback: The performance gains from switching R's linear algebra libraries | On the lambda (http://www.onthelambda.com/2013/12/26/the-performance-gains-from-switching-rs-linear-algebra-libraries/)

Pingback: Squeezing more speed from R for nothing, Rcpp style | On the lambda

(http://www.onthelambda.com/2014/06/27/squeezing-more-speed-from-r-for-nothing-rcpp-style/)

OWE JESSEN
June 28, 2014 / 3:46 am

I think one problem in the abysmal performance could be the apply function in the sum.of.distances function. I get the following results

single <- function(){
the.sum <- sum.of.distances(combos)
result <- the.sum / num.of.comps
result
}

vectorized = function(){
the.sum <- sum(calc.distance.two.rows(combos[1,], combos[2,]))
result benchmark(single(), vectorized())
test replications elapsed relative user.self sys.self user.child sys.child
1 single() 100 1.06 7.067 1.07 0 NA NA
2 vectorized() 100 0.15 1.000 0.16 0 NA NA

Reply (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/?replytocom=3935#respond)

TONY.FISCHETTI@GMAIL.COM
June 30, 2014 / 11:21 am

Maybe wordpress got the formatting of your comment wrong, but I don't see how the "vectorized()" function could work

Reply (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/?replytocom=3955#respond)

Pingback: Lessons learned in high-performance R – On the lambda (http://www.onthelambda.com/2015/05/31/lessons-learned-in-high-performance-r/)

Leave a Reply

Your email address will not be published. Required fields are marked *

Name*

Email*

Website

Message...

POST COMMENT

ABOUT ME

Data scientist at CollegeFactual (http://www.collegefactual.com). Vegetarian. Interests lie in R, python, cognition, game theory, machine learning, books, ethics, post-punk, animal rights, and oxford commas.Follow @tonyfischetti (https://twitter.com/tonyfischetti)

(https://www.facebook.com/tony.fischetti) (https://github.com/tonyfischetti) (https://twitter.com/tonyfischetti)

TAG CLOUD

awk (http://www.onthelambda.com/tag/awk/) classification (http://www.onthelambda.com/tag/classification/) data mining (http://www.onthelambda.com/tag/data-mining/) datavis (http://www.onthelambda.com/tag/datavis/) debian (http://www.onthelambda.com/tag/debian/) dplyr (http://www.onthelambda.com/tag/dplyr/) evolutionary biology (http://www.onthelambda.com/tag/evolutionary-biology/) functional programming (http://www.onthelambda.com/tag/functional-programming/) ggplot2 (http://www.onthelambda.com/tag/ggplot2/) graph theory (http://www.onthelambda.com/tag/graph-theory/) haskell (http://www.onthelambda.com/tag/haskell/) high performance computing (http://www.onthelambda.com/tag/high-performance-computing/) hypothesis testing (http://www.onthelambda.com/tag/hypothesis-testing/) linguistics (http://www.onthelambda.com/tag/linguistics/) lisp (http://www.onthelambda.com/tag/lisp/) mac (http://www.onthelambda.com/tag/mac/) machine learning (http://www.onthelambda.com/tag/machine-learning/) magrittr (http://www.onthelambda.com/tag/magrittr/) markov chains (http://www.onthelambda.com/tag/markov-chains/) monads (http://www.onthelambda.com/tag/monads/) music (http://www.onthelambda.com/tag/music/) natural language processing (http://www.onthelambda.com/tag/natural-language-processing/) nhst (http://www.onthelambda.com/tag/nhst/) optical character recognition (http://www.onthelambda.com/tag/optical-character-recognition/) os x (http://www.onthelambda.com/tag/os-x/) p-values (http://www.onthelambda.com/tag/p-values/) pedagogy (http://www.onthelambda.com/tag/pedagogy/) perl (http://www.onthelambda.com/tag/perl/) python (http://www.onthelambda.com/tag/python/) R (http://www.onthelambda.com/tag/r/) reading

(http://www.onthelambda.com/tag/reading/) research (http://www.onthelambda.com/tag/research/) sake (http://www.onthelambda.com/tag/sake/) science (http://www.onthelambda.com/tag/science/) scientific workflow systems (http://www.onthelambda.com/tag/scientific-workflow-systems/) self-tracking (http://www.onthelambda.com/tag/self-tracking/) shell (http://www.onthelambda.com/tag/shell/) shiny (http://www.onthelambda.com/tag/shiny/) software (http://www.onthelambda.com/tag/software/) statistics (http://www.onthelambda.com/tag/statistics/) twitter (http://www.onthelambda.com/tag/twitter/) unix (http://www.onthelambda.com/tag/unix/)

RECENT POSTS

- Damn the torpedoes, full speed ahead: making the switch to Python 3 (http://www.onthelambda.com/2014/05/13/damn-the-torpedoes-full-speed-ahead-making-the-switch-to-python-3/)
- Take a look, it's in a book: distribution of kindle e-book highlights (http://www.onthelambda.com/2014/04/10/take-a-look-its-in-a-book-distribution-of-kindle-e-book-highlights/)
- How to make an absurd twitter bot in python (http://www.onthelambda.com/2014/03/20/how-to-make-an-absurd-twitter-bot-in-python/)
- How to fake a sophisticated knowledge of wine with Markov Chains (http://www.onthelambda.com/2014/02/20/how-to-fake-a-sophisticated-knowledge-of-wine-with-markov-chains/)
- How dplyr replaced my most common R idioms (http://www.onthelambda.com/2014/02/10/how-dplyr-replaced-my-most-common-r-idioms/)
- Using Last.fm to data mine my music listening history (http://www.onthelambda.com/2014/01/27/using-last-fm-to-data-mine-my-music-listening-history/)
- The performance gains from switching R's linear algebra libraries (http://www.onthelambda.com/2013/12/26/the-performance-gains-from-switching-rs-linear-algebra-libraries/)
- Visualizing data analysis pipelines using NetworkX (http://www.onthelambda.com/2013/12/08/visualizing-data-analysis-pipelines-using-networkx/)
- Compiling R from source and why you shouldn't do it (http://www.onthelambda.com/2013/11/22/compiling-r-from-source-and-why-you-shouldnt-do-it/)
- Parallel R (and air travel) (http://www.onthelambda.com/2013/11/13/parallel-r-and-air-travel/)
- qstats - quick and dirty statistics tool for the Unix pipeline (http://www.onthelambda.com/2013/11/05/qstats-quick-and-dirty-statistics-tool-for-the-unix-pipeline/)
- Linguistics, meet Evolutionary Biology (http://www.onthelambda.com/2013/10/29/linguistics-meet-evolutionary-biology/)
- Unsupervised correction of optical character misrecognition (http://www.onthelambda.com/2013/10/22/unsupervised-correction-of-optical-character-misrecognition/)
- The state of package management on Mac OS X (http://www.onthelambda.com/2013/10/14/the-state-of-package-management-on-mac-os-x/)
- On the treachery of point-and-click "black-box" data analysis (http://www.onthelambda.com/2013/10/07/on-the-treachery-of-point-and-click-black-box-data-analysis/)
- On the misinterpretation of p-values: (http://www.onthelambda.com/2013/10/02/on-the-misinterpretation-of-p-values/)

Flato (http://www.themememe.com/flato) by ThemeMeme