programming languages,[7] and be proficient in the twenty most frequently used Unix commands. Calculus was fine for Gauss and the Bernoullis, but this "Party like it's 1827!" approach to methodological pedagogy in the twenty-first century has some limitations.

Finally, we need to engage the policy community as assertive professionals based on the rich knowledge that we bring as political methodologists. We suggest revisions, we debate, we cajole, we use the best available techniques but point out their flaws as well as virtues and—critically—we turn down work when it doesn't make any sense.

Those who reject engagement with the policy community will quickly point to the risks—and they are real—of large-scale applied projects.[8] And such projects are not always easy—there have been many times when dealing with multiple layers of bureaucracies, contractors, and poorly-framed requirements that I've thought that it might be nice to just go back to, say, writing theoretical papers demonstrating that World War I couldn't occur, and therefore didn't. But systematic political analysis *is* being done, and because of the technological imperatives I noted earlier, it is going to be done whether political methodologists are involved or not. I believe that with our expertise in theory, design and method, it will be done better if our community is involved. There are risks in getting involved, but I believe the risks in not getting involved are even greater.

The question of whether any scientific community should engage in issues of practical import is not new: this issue (as well as the issues of openness and replicability) was at the core of Francis Bacon's foundational work in the early seventeenth century on what became the modern scientific approach. Of course, in the universities, Bacon's approach would continue to lose out to the established Scholastic approaches for the better part of three centuries.[9]

Several of our cognate disciplines—economics, demography and epidemiology—resolved this issue some time ago, as did the applied field of election forecasting. But for the most part we've lagged, and most (all?) of the twenty or so graduate programs capable of training (and innovating) in state-of-the-art methods seem to actively discourage students from pursuing a policy-oriented track despite the availability of jobs and funding. The prospect of publishing an incremental contribution to normal science in the *APSR* is still seen as superior than doing technical political analysis that can involve stakes of millions or even billions of dollars, and/or hundreds or even thousands of lives. I'm not convinced we serve ourselves and our communities well in that regard.

Computing and Software

A Note on Speeding Up `R` for Windows

Dino P. Christenson and Joshua A. Morris
The Ohio State University
christenson.24@polisci.osu.edu morris.521@polisci.osu.edu

Abstract

To what extent do different Windows PC characteristics increase the modeling efficiency of R? Do some programs or versions of R run better on different PCs? And for which kinds of models do enhanced PCs and clusters diminish processing time? This research note seeks to provide novice to intermediate level R users with a framework for understanding the benefits of explicit parallel processing and upgrades in PC hardware for large datasets and computationally burdensome models. We compare the relative benefits of each optimization with simple efficiency tests. In addition, we provide basic R code to make the transition to parallel processing easier for novice users without networked labs or cluster access.

Introduction

Today the onus of statistical modelling derives from two major sources: statistical knowledge and computational resources. This note concerns the latter. Computational resources are not limited to the operating system (OS) and

[7] except LISP

[8] Hint: before engaging in such a debate, get a promise that the first party to use the word "prostitute" has to buy a round of beer.

[9] And during the heyday of post-modernism in the 1990s, even Scholasticism was looking pretty good.

The authors' names are listed alphabetically. The authors would like to thank members of the Program in Statistics and Methodology (PRISM) and the Political Research Laboratory (PRL) at Ohio State University for encouragement and computing resources, particularly Professors Janet Box-Steffensmeier and Luke Keele as well as PRL staff Bill Miller and Isaac How; all errors are the sole responsibility of the authors. While the PRL at OSU uses Windows PCs and servers, both authors would like to note that they use Macs at home. Additional information, `R` code and related resources can be found at `http://polisci.osu.edu/prism/resources.htm`.

statistical package employed. As Sekhon (2006) notes, some operating systems are generally more efficient than others (Linux is faster than Windows XP which is faster than Mac OS X, unless the memory allocation is replaced); such is the case with statistical programs as well (Matlab is faster than R which is faster than Stata). However, substantial gains in speed can also be made within a chosen operating system and statistical package. Small upgrades in basic hardware, reformatting the data, allocating the memory and engaging explicit parallel processing all lead to relatively large jumps in the computational efficiency of statistical modelling.

We measure the extent to which hardware, package versions, cluster processing and data formatting increase the processing speed of statistical models. We focus exclusively on the interaction between the Windows XP and Vista OSs and the R statistical environment. R is increasingly popular, open source, free and, according to a recent *New York Times* article, "easy to use" for "statisticians, engineers and scientists without computer programming skills" Vance (2009). Windows, while not the statistician's current OS of choice, is the most prevalent OS in the world. Among Windows OSs, XP is the most common and Vista the latest version [1]. It is clear that regardless of the OS preferred by statisticians and programmers, the bulk of novice to intermediate applicants of statistics are still running their models on Windows. In addition, the abundance of R packages and vignettes continually make R more user-friendly to a broad spectrum of statistical modellers. Accordingly, the complex algorithms involved in various popular models can be employed without a programming background.

We believe, however, that the move to making efficiency gains in processing speed has not been dealt with as seamlessly as with the other aspects of statistical modelling. While intermediate users may make use of the models so neatly packaged in R, the benefits from such models may be mitigated by the computational burden. Should the costs be too high, students and scholars may be discouraged from using the appropriate model, in favor of less computationally intensive models and, at worst, inappropriate ones. Ultimately, we find that efficiency gains can be made through slight tweaks in the processing, hardware, version and data of models, while holding the basic OS and statistical package constant. Thus all hope for efficiency gains is not lost for Windows-based R users.

Efficiency Testing

Benchmarking processing speeds have been based on a host of models, most notably on Genetic Matching (Sekhon 2006) and bootstrapping (Rossini 2003). While the model chosen for the benchmarks will undoubtedly influence the processing speed, Sekhon (2006) recommends approaching benchmarking as a deterministic process. As such, the particular algorithm employed by the model is not critical, so long as it is consistent across machines and that one controls for the potential confounding factors. Given a lack of unobservables in computers, efficiency gains can be measured by holding all hardware and software constant and merely changing the variable of interest.

We test the processing speed with the `boot` function (Canty 1997). The `boot` function calls forth bootstrapping, a resampling method for statistical inference (Davison 1997, see also Keele 2008). While conceptually simple, bootstrapping is computationally intensive, often requiring thousands of sample replications. We follow Example 6.8 used in Davison (1997) and subsequently in Rossini's (2003) paper on simple parallel computing in R. The example seeks to demonstrate the benefit of bootstrapping on estimating the cost of constructing power plants. In the benchmarking that follows we generate bootstrap samples of a generalized linear model fit on the familiar `nuclear data` (Cox 1981). The data is comprised of the eleven variables including the dependent variable, the `cost` of the construction in millions of dollars, `date` of the construction permit and nine other seemingly relevant independent variables.

We begin by testing four different R compositions on the six different machines available in the PRL (see Table 1).[2] The different machines are identical in terms of loaded software and the version of R (2.8.0). Therefore we simply note the difference in allotted RAM, GHz, processor type, number of processors and OS, and compare the time to completion for our bootstrapping models across the machines. In each case, we run the model five times on each machine and present the average processing time across the five runs. Later we consider the potential benefit from a commercial version of R and explicit cluster processing relative to our standard R baseline. We conclude with a couple of simple tests of the benefits of data formatting and memory allocation.

[1] See W3 for a comparison of OS usage at `http://www.w3schools.com/browsers/browsers_os.asp`

[2] Note that the available machines do not allow for perfect control of the potentially confounding factors within a computer. The available machines are such that it is not possible for us to test the impact of, for example, the move from one GB of RAM to two GBs without also changing the GHz or some other feature. Thus the increase in efficiency from such a move cannot be attributed solely to the increase in GB or RAM, per se, but to an increase in both variables.

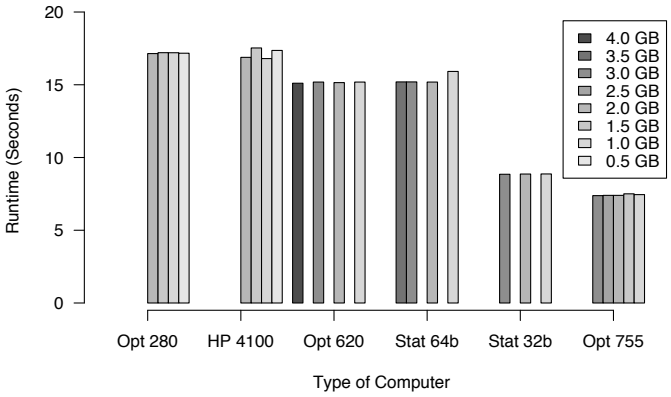
Table 1: Machines and Programs Tested

Machine	Program	RAM in GB	Processor Number	Speed in GHz	Multi Core	OS
HP 4100	*R*	.5, 1, 1.5, 2	1	2.8	no	XP
	REvolution	.5, 1, 1.5, 2	1	2.8	no	XP
	R Snow	.5, 1, 1.5, 2	1	2.8	no	XP
	ParallelR	.5, 1, 1.5, 2	1	2.8	no	XP
Optiplex 280	*R*	.5, 1, 1.5, 2	1	2.8	no	XP
	Revolution	.5	1	2.8	no	XP
	R Snow	.5	1	2.8	no	XP
	ParallelR	.5	1	2.8	no	XP
Optiplex 620	*R*	1, 2, 3, 4	1	3.2	partial	Vista
	REvolution	2	1	3.2	partial	Vista
	R Snow	2	1	3.2	partial	Vista
	ParallelR	1	1	3.2	partial	Vista
Optiplex 755	*R*	1, 1.5, 2, 2.5, 3	1	2.83	yes	Vista
	REvolution	1, 1.5, 2, 2.5, 3	1	2.83	yes	Vista
	R Snow	1, 1.5, 2, 2.5, 3	1	2.83	yes	Vista
	ParallelR	1, 1.5, 2, 2.5, 3	1	2.83	yes	Vista
32 Bit Stats	*R*	1, 2, 3	1	2.4	yes	XP
	REvolution	3	1	2.4	yes	XP
	R Snow	3	1	2.4	yes	XP
	ParallelR	2	1	2.4	yes	XP
64 Bit Stats	*R*	1, 2, 3, 3.5	2	3.2	partial	XP
	REvolution	1, 2, 3, 3.5	2	3.2	partial	XP
	R Snow	1, 2, 3, 3.5	2	3.2	partial	XP
	ParallelR	1, 2, 3, 3.5	2	3.2	partial	XP

Hardware

Upgrades in computer hardware are intended to make applications run smoother and quicker. It should be expected that hardware makes a difference in the processing speed of statistical models. We test six basic hardware factors, and hence what kind of machine composition should be used for the most efficient results, specifically: processor type, processor speed, number of processors, and the amount of RAM installed.

Figure 1: Hardware Processing Time: Memory and Speed



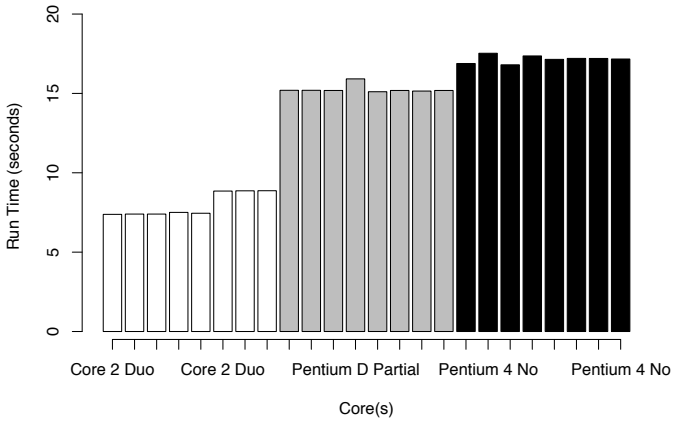
The cluster of bars denote different RAM configurations on each computer. Thus a gap between the bars are indicative of a missing RAM configuration for that computer.

Figure 1 shows that computer speed (GHz) has a minimal effect on processing time. The differences in the amount of RAM serves to increase speed for systems 2GB or greater, ortherwise it too has only a small effect on modeling efficiency. Contrarily, Figure 2 suggests that the type of processor, specifically if it is true multicore or not, is very influential. This is particularly surprising given that R does not make use of multithreaded processing (at least as of version 2.8.0). In our computation intensive test, low speed multicore processor systems perform the best. This is likely due to the fact that the processor can handle background system tasks with one core while devoting the other core to R, preventing a continual tradeoff that takes place in single core systems.

Commercial Software

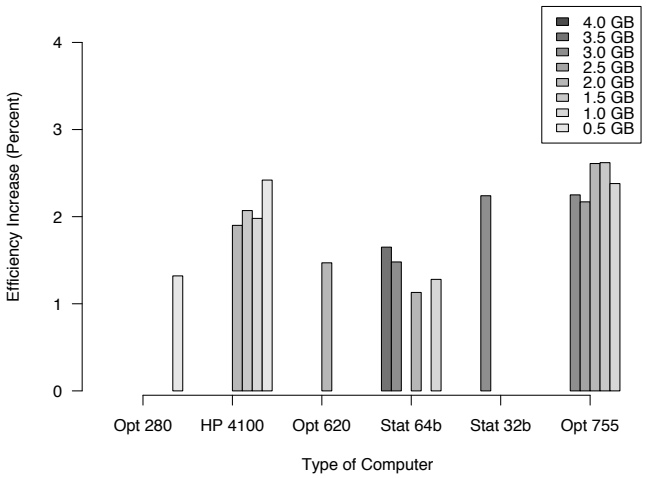
REvolution R by REvolution Computing is a commercial version of R built on the open source code base. It uses additional commercial packages and optimizations in an effort to run "many computationally-intensive programs faster." It is currently a free download with registration. We find that REvolution R outperforms the basic R in our test model across a variety of hardware. The performance increase was small, between 1.33 percent and 2.25 percent, but consistent (see Figure 3). The increase in speed was greatest for multicore processors, both greater than 2.24 percent.

Figure 2: Hardware Processing Time: Processor Core(s)



Multicore indicates two cores on a single die; the Pentium D is technically a dual core processor but the cores reside on two dies instead of one. For these tests it is considered partial multicore. The Pentium 4 No refers to a simple Pentium 4 processor without multicore. The bars denote different computer configurations.

Figure 3: Percent Increase in Efficiency of REvolution R



Parallel Processing

In computational problems the ability to process sections in parallel is highly appealing, however parallelization is not widely used. The surprising lack of use is perhaps an indicator of the troubles encountered when the statistical modeller encounters features of the programming

world. The computational difficulties encountered with parallel processing can be daunting for non-programmers.[3]. In explicit parallelization, the modeller dedicates different processes to different processors or computers. The processor which makes the calls is typically referred to as the master and the processors that carry out their given activities, the slaves. By dedicating different processes to different clusters, the statiscal program on the master node is able to take advantage of the division of labor. Each slave node is able to concentrate on its particular task and the master collects and organizes the results. In time-intensive modelling this division of labor can create enormous gains in efficiency.

We use the R package, `snow`, which appears to be the most common package for explicit parallelization and fairly user-friendly.[4] In our tests, the bootstrapping is broken up from a single set of 1000 replicates to parallel runs of 500 for a size two cluster and runs of 250 for a size four cluster. Each section of replication commands is submitted to a cluster node, where the nodes are a part of a socket cluster on the local machine. Thus we run our parallel tests on a single computer. While many advanced modellers have access to (Beowulf) clusters or networked labs, we demonstrate that efficiency gains can be made from parallel processing for novice to intermediate level users with access limited to a single machine. This means that our results can be replicated should one not have access to a networked laboratory of computers or a sophisticated cluster.[5]

Given a single machine, the process for parallelization in `snow` can be reduced to 5 simple steps. We begin by installing and loading the appropriate package.

```
# load snow library
library(snow)
```

Next, we create clusters on our computer. We utilize the `makeCluster` command to seek out nodes on the local host and specify the number of nodes to enter the cluster. Here, we also specify the type of connection between nodes: sockets, in our case, but MPI, and PVM connections are also permitted.

```
# makes a local socket cluster of size 4
c1 <- makeCluster(4, type="SOCK")
```

We can check the names and kind of processors that make up the cluster nodes with `clusterInfo` or `clusterCall`. Leaving the function expression blank returns the specified information.

```
# test cluster nodes to see name and machine
clusterCall(c1, function() Sys.info()[c("nodename",
"machine")])
```

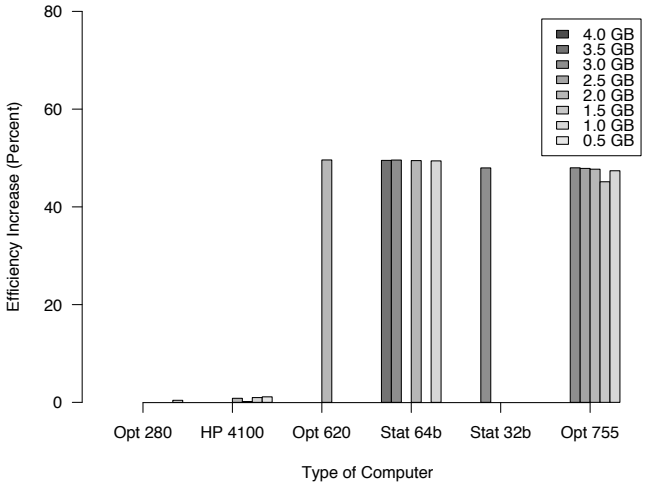
We utilize `clusterEvalQ` to evaluate a specified expression on each node. Per our example, we call it to examine the `boot` function for every node on the cluster.

```
# load boot library to cluster
clusterEvalQ(c1, library(boot))
```

We revisit the `clusterCall` command to collect the information performed at each node. `clusterCall` is the main function of the parallelization. `clusterCall` specifies arguments to be performed on each slave node and returned to the master. Finally, do not forget to call `stopCluster` at the end; otherwise, the clusters will remain connected.

```
# bootstrapping on a cluster of size 4
clusterCall(c1,boot,nuke.data,nuke.fun,R=250,m=1,
fit.pred=new.fit,x.pred=new.data))
# cleans up cluster and stops it
stopCluster(c1)
```

Figure 4: Percent Increase in Efficiency of Parallel Processing with 2 Clusters



[3]In fact, so many "user-friendly" parallel tools are in development that deciphering between them can be difficult. For example, there is an R Parellel at `http://www.rparallel.org/` that has the same objectives as REvolution's Parallel R, but is unrelated. We have not yet experimented with the former resource.

[4]R offers a host of parallel computing options. A list of R resources on parallel computing are available online at `cran.r-project.org/web/views/HighPerformanceComputing.html`. Our parallel clustering section and bootstrap model follows Rossini (2003) closely.

[5]We would expect efficiency gains from networked computers or clusters to be even greater given more time-intensive tasks.

Figure 5: Percent Increase in Efficiency of Parallel Processing with 4 Clusters

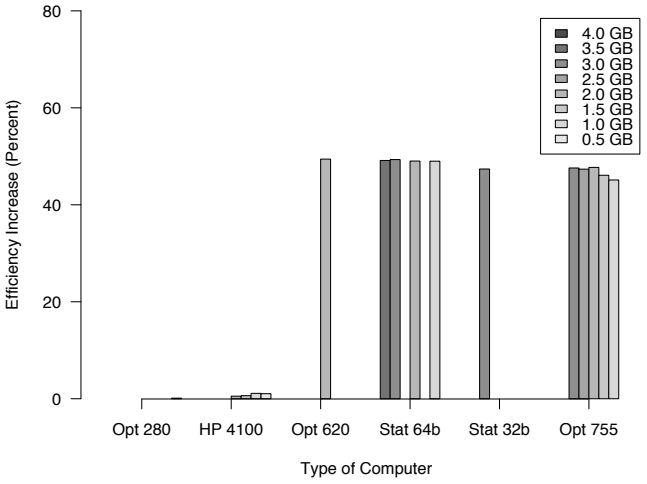
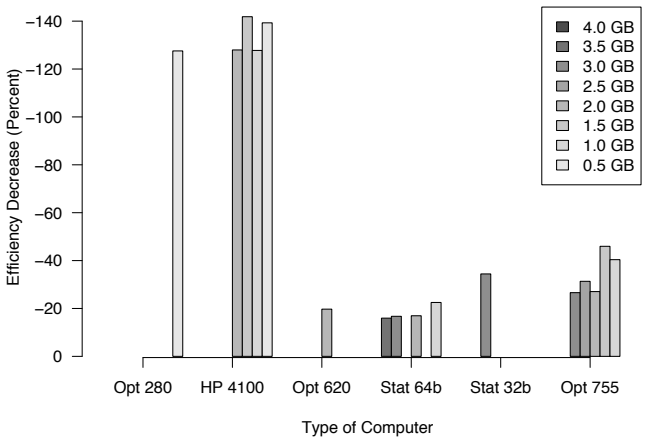


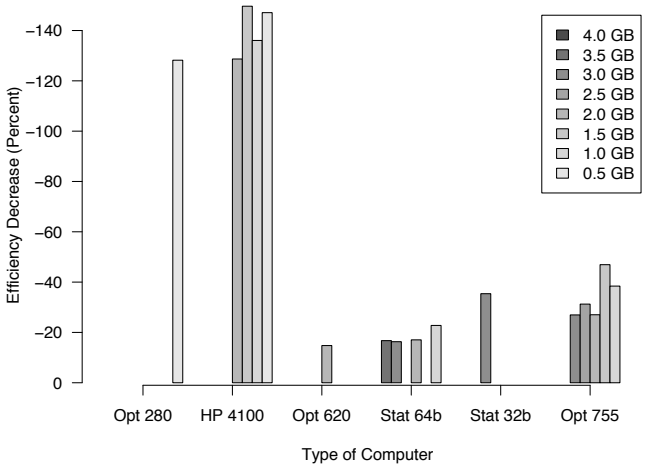
Figure 6: Percent Decrease in Efficiency of ParallelR Processing with 2 Clusters



Figures 4 and 5 show that the speed-up is substantively large in our test for machines with partial or full multicore. Multicore machines perform better than their non-clustered counterparts by over 45 percent. Single core machines see only a slight advantage, performing with less than a 1 percent speed increase over their non-clustered counterpart. We note that additional nodes do not always increase performance, as the overhead of the nodes increases with each additional node.

REvolution Computing also offers ParallelR with Enterprise, a version that includes support and automated parallel processing and optimization.[6] We compare the automated parallel processing in Enterprise with our basic R results. We found ParallelR especially user-friendly. The installation was quick and simple. The automated commands were intuitive and the move to running the new `boot` command was as simple as writing `bootNWS`. However, we do not witness any efficiency gains from the ParallelR program. Contrary to our expectations, the move to ParallelR resulted in decreases in efficiency for all machine compositions and for both size two (see Figure 6) and size four clusters (see Figure 7).[7] The results suggest that efficiency gains from parallel processing are not constant across computer compositions or types of clusters.

Figure 7: Percent Decrease in Efficiency of ParallelR Processing with 4 Clusters



Other Recommendations

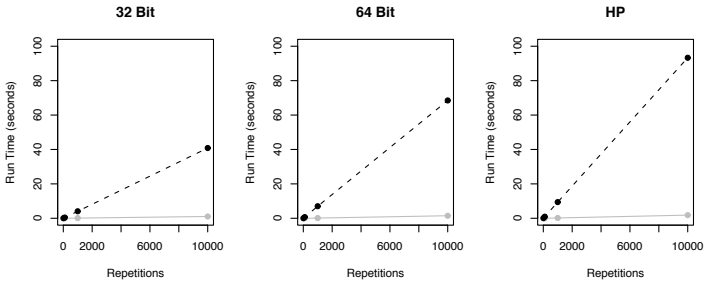
R programmers and researchers familiar with the mechanics of R have proposed additional time saving methods. Lumley (2005) of the R Core Development Team has noted various optimizations for R code, including but not limited to the following: data frames are much slower than matrices

[6] REvolution Computing was kind enough to allow us to test their ParallelR for free. More information on REvolution is available at their website `http://www.revolution-computing.com/`

[7] It is important to note, however, that we are dealing with a moderately time-intesive model, given our small dataset. In such cases, the amount of overhead it takes to supply the machine with directions for parallel processing may not be recouped by any increase in processing speed from the parallelization. In addition, it is possible that some of these programs are more efficient when run through a network cluster than on a single machine.

(especially large ones); functions that have few options and little error check are faster; and allocating memory all at once is faster than incremental allocation.

Figure 8: Processing Time of Data Frames and Matrices



We confirm these recommendations with some simple tests. We begin by running a test of the speed of data frames versus matrices. In the R environment, data frames can contain elements of mixed types while a matrix may contain elements of only one type. The test used a modified version of the `nuclear` data, called `nuclearbig`, a larger variant of `nuclear` created by duplicating the data many times over. The test consisted of comparing the differences between multiplications of the data as a matrix and as a data frame. We find that the time difference is greatest for a large number of calculations, so it is the most valuable to check what form your data is in for repeated tasks. Overall, the speed increase of using a matrix instead of a data frame in our test was around 98 percent (See Figure 8).[8]

We also checked to see if functions that have few options and little error checking are faster. Using `sum(x)/length(x)` in comparison to `mean(x)` we confirm our expectations; however, the speed difference is quite minimal for a small number of replications (see Table 2). In fact, we repeat the calculation 30,000 times before a noticeable time difference occurs. Even in this case the actual reduction was very small, less than one second. While this could result in a slight increase in speed for large repetitons, above the tens or hundreds of thousands, it is unlikely to be of substantive help otherwise. Of course, this is only one function and other function replacements may be more or less effective, but beyond this note.

We also find that memory allocation has a high overhead. We test the value of allocating memory all at once instead of incremental allocation by loading two variables full of numbers using a simple function similar to the example above. When allocating memory first using `y <- numeric(30000)` in Table 2 we find an average speed increase of 93 percent. A careful look at algorithmic approaches such as this one could make a strong difference to the run-time of the program. Of note, this optimization changes the test time from 5 seconds to 0.25 seconds on the HP 4100.

Conclusions

For the most efficiency in computationally intensive tasks in R, use multicore processor machines, if possible. Again, the gains from such a framework will not be evident on simpler models or with small to medium sized datasets, but with time-intensive models or extremely large datasets significant boosts in efficiency can be gained from utilizing the best available machines.

Programs such as REvolution R can be helpful for code that is not easily broken up for parallelization or projects which are not large enough to justify R coding tricks or rewrites; however the benefits are small relative to the other available optimizations. For computations that can be explictly parallelized, the `snow` package could be used to decrease time dramatically, even on a single machine. For bootstrapping, matching and Bayesian models, the availability of multicore processing and explicit parallel processing can be quite helpful. In these same situations it is also helpful to allocate the memory up-front. However we did not find all parallel programs to increase efficiency equally across machine compositions, and some not at all. To that end we believe that developers need to better specify the expected efficiency gains from their programs and packages, and do so with some attention to different computer compositions.

References

Canty, Angelo. 1997. "Boot Function. S original. R port" by Brian Ripley. In *Bootstrap Methods and Their Application.* Cambridge University Press.

Cox, David R. and Joyce E. Snell. 1981. *Applied Statistics: Principles and Examples.* Chapman and Hall.

Davison, Anthony C. and David V. Hinkley. 1997. *Bootstrap Methods and Their Application.* Cambridge University Press.

Keele, Luke. 2008. *Semiparametric Regression for the Social Sciences.* Wiley.

Lumley, Thomas. 2005. "R/S-PLUS Fundamentals and Programming Techniques." University of California, Berkeley.

[8] Unfortunately, this optimization is not always available, as using matrices instead of data frames is not possible for some functions, for instance, our original bootstrapping function requires data frames.

Rossini, Anthony, Luke Tierney and Na Li. 2003. "Simple Parallel Statistical Computing in R." UW Biostatistics. Working Paper Series .

Sekhon, Jasjeet Singh. 2006. "The Art of Benchmarking: Evaluating the Performance of R on Linux and OS X." *The Political Methodologist,* 14(1): 15-19.

Vance, Ashlee. 2009. "R You Ready for R?" *New York Times*.

Table 2: Average Percent Increase in Efficiency

Machine	Snow Size 2	Snow Size 4	REvolution	ParallelR Size 2	ParallelR Size 4	Matrices Frames	Less Options	Memory Up Front
Opt 755	47.22	46.77	2.41	-34.27	-34.12	96.77	88.46	91.86
32B Stats	47.97	47.38	2.24	-34.43	-35.40	94.14	86.69	92.49
64B Stats	49.49	49.12	1.38	-18.04	-18.22	97.76	88.68	89.73
Opt 620	49.60	49.42	1.47	-19.72	-14.79	98.33	88.30	91.15
HP 4100	0.77	0.83	2.09	-134.22	-140.37	97.84	88.41	96.60
Opt 280	0.42	0.12	1.32	-127.56	-128.20	98.61	89.27	90.29

Professional Development

Advice to Junior Faculty Column: Advice from John E. Jackson

John E. Jackson
University of Michigan
jjacksn@umich.edu

First, let me say these are very thoughtful and stimulating questions. I enjoyed developing answers to them, and being reminded of the pressures and opportunities I faced at the same stage. Some issues are different, but you might be a bit surprised to know how many are similar.

Expanding Networks

> Q. I really enjoy the Polmeth Summer Meeting, and am wondering about what other conferences I might attend with similar intellectual content? And what about these smaller conferences I keep hearing about? How do you find out about and get invited to those?

A. I am glad you enjoy and benefit from the Polmeth Summer Meetings. The hosts go to great length in putting on the meetings so it is nice to know their work is appreciated. The opportunities offered by the regional meetings, such as the ones organized by Jeff Gill in St. Louis and Jonathan Nagler in New York, are good examples of what you are looking for. Though they have regional names they are not restricted to scholars in those regions any more than are the Midwest meetings. These are smaller and more informal yet bring a very high quality of participant and discussion. Invitations are regularly posted on the Polmeth server.

Beyond these and the summer meetings, unfortunately, there is no simple recipe for being invited to conferences because they involve many different types, purposes, and funding arrangements. Some conferences have a broadly distributed call for papers, using venues such as the Polmeth server, *PS,* etc. Watch for these and send in a proposal if there is the slightest link from the theme to your work. At your institution become part of centers and institutes as these organizations often receive conference announcements and distribute these to faculty associated with the center. Also regularly search the websites of relevant academic and policy institutes, such as the World Bank, as they usually include calls for papers for future conferences. These conferences often have a more substantive than a methodological theme but they offer important visibility for your work and the chance to meet other scholars with interests related to