HPC and Big Data Analytics

MATTHEW DENNY TUESDAY 5TH AUGUST, 2014

ICPSR

Welcome to this practical introduction to high performance computing (**HPC**) and big data analytics for social science applications. This tutorial is meant to give you some basic tools and resources for data management and analysis when time, memory and processing constraints make doing so using a laptop impractical. Some preliminaries:

1. You should be using the RStudio IDE for developing HPC R code. It has Git integration, it makes seeing the size of objects in memory easy, and it is really nicely integrated with RCpp. You can get it here (it is free): https://www.rstudio.com
2. If you are just looking to dabble, RevolutionR has a free academic license and will speed up your code without the need to really do anything on your part. I personally don't use it because it masks some of the functioning of what it is doing which can be a problem if you are rolling your own programs. Feel free to check out their website: http://www.revolutionanalytics.com/academic-and-public-service-programs
3. If you plan on doing an serious HPC you need access to a computer running Linux. You should probably go with Ubuntu, although Debian is also usually fine. Other operating systems can provide more functionality for HPC and remote management, but have an even steeper learning curve. I use Ubuntu and CentOS 6. You can download Ubuntu here: http://www.ubuntu.com/. If you are a Mac user, you are already fine because you have access to all of the Unix goodies already. If you are a windows user and don't want to make the full switch or dual-boot your machine, download VirtualBox and install a virtualized version of Ubuntu. VirtualBox is free and you can get it here: https://www.virtualbox.org/
4. If you are going to do any sort of HPC work, you need to know how to program. I would suggest checking out the *Introduction to Scientific Programming and Simulation Using R* by Owen Jones, Robert Maillardet and Andrew Robinson which can be found here: http://www.crcpress.com/product/isbn/9781420068726
5. I would also suggest checking out my tutorial on advanced data management in R, available here: http://www.beyondeconomy.net/Files/Advanced_Data_Management_in_R.zip

1 A Number of Practical Considerations

There are a number of practical considerations to take into account when deciding whether the effort to make use of HPC techniques is worth it, and which strategies are best for your particular situation. In general, if you only find yourself needing access to more computing resources on a very infrequent basis, your best resource will simply be to seek out a colleague who has access to HPC resources and get their help with the project. Depending on what you need to do, developing the resources and skills to do something yourself can be less efficient than simply getting help. HPC is about getting things done efficiently so think about the most efficient approach before you go out and get access to a big cluster.

1.1 Memory

Computer memory (either RAM or hard drive space) is the most common limiting factor making a laptop impractical for the management and analysis of large datasets. Most laptops and desktops ship with 4 or

8 GB of RAM, which is large enough to hold most datasets up to about 500,000 observations of a couple hundred variables (with 8 GB RAM) in memory and actually perform analysis on them. If you try to load a dataset into memory that is too large, your computer will lock up. This is illustrated in Figure 1 where the computer is using almost all available RAM and will result in your analysis slowing to a crawl (in the best case scenario) or in your computer crashing (worst case).

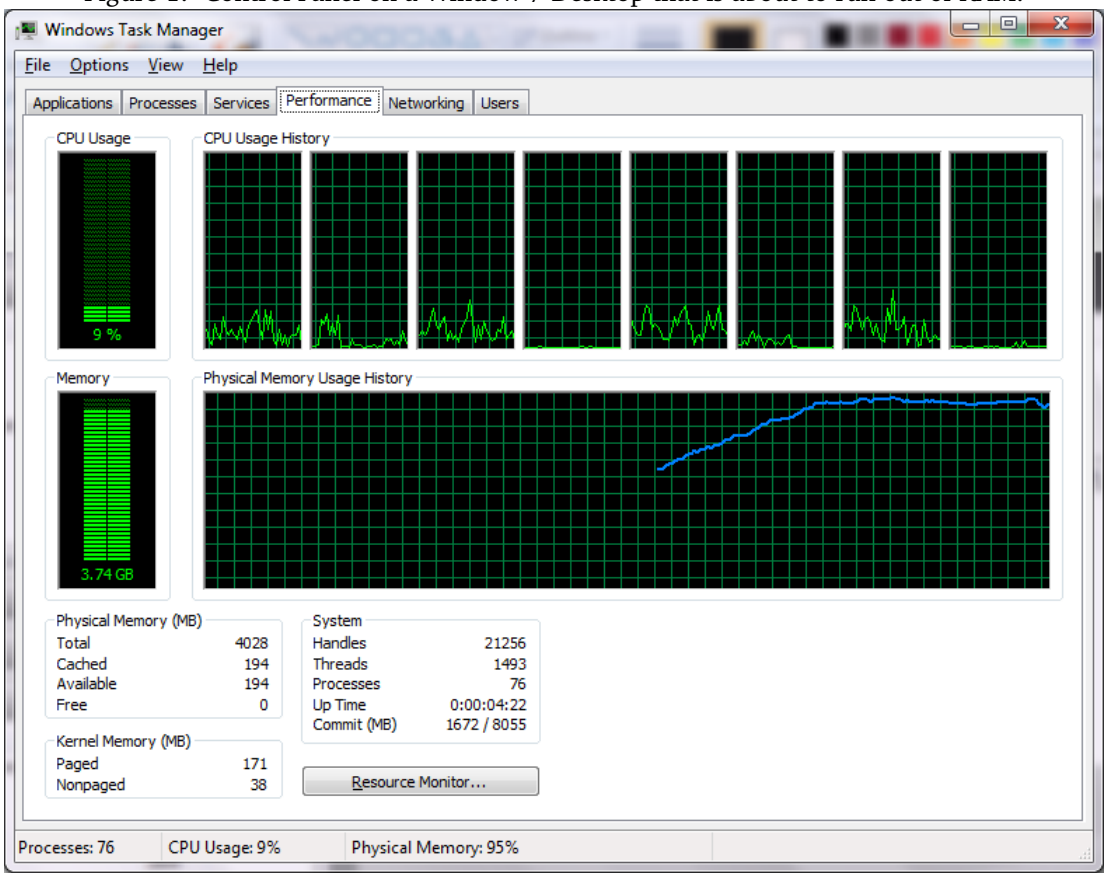
The problem with running out of memory is that in most cases it does not matter how much time you have, you simply cannot work on that problem. Fortunately, this is also one of the simplest things to take care of. What follows is a list of several strategies I have taken in this kind of situation:

1. As a special note, Stata tends to be much more memory efficient than R in most cases, so if you are just running some vanilla regression analysis, you can often get around memory problems by using Stata instead of R.
2. Have your activity monitor open when you go to open a dataset or try to run a regression using it. Monitor your RAM and CPU utilization. This should always be your first step to see if memory is going to be an issue.
3. Buy more RAM for your computer if possible. RAM is relatively cheap (16 GB DDR3-1600 kits can be had for under $150) and should be the first upgrade you make to any computer. In most cases, going from 4 to 8 or 8 to 16 GB of RAM will solve most of your problems with not being able to load a dataset on your computer. Alternatively you can just try to access a computer with more memory. A good rule of thumb is that desktops usually have more RAM than laptops so try running your analysis on a computer lab desktop before doing anything else.
4. The next thing to try is breaking down your data problem into smaller pieces. For example if you are working with panel data and need to calculate some new variables, reading in the data for one year at a time, calculating the new variables, saving the data and then removing it from working memory before loading the next year. This can also be done for GLM using the **biglm** package in R. This package has the functionality to read in data several thousand observations at a time an run a glm.
5. If breaking down the problem or using a more memory efficient language does not work, then you need to go for a computer with more RAM. The first thing you need to asses is how much memory you are likely to need. If this is 64GB of memory or less, then you can achieve this with consumer-grade hardware or a desktop workstation. Most departments at UMass have a faculty member with a workstation with 64GB RAM so this should be your first option. If you need more than 64GB then you need access to a cluster. See Section 2.5 for information on setting up access the UMass cluster. You should only take this avenue if you have genuinely exhausted resources available in your department as this will mean a larger time investment and more programming challenges.

1.2 Time

The nice thing about time is that it just keeps going. The downside is that we often have a finite amount of time for a particular project. A general rule of thumb for a social science computing task is that if it can be done in less than about 6 months on a reasonable 4 core desktop, there is not a huge need to put a lot of energy into speeding it up. If you have a desktop you can leave attached to an uninterrupted power supply to just run for a couple of months, and you know it will produce the result you need at the end of the day, this is usually a manageable solution. Yet sometimes there are problems we need to compute in a shorter amount of time, or for which running the computation with a single thread might take years or even decades. In this case, computing time becomes an issue that requires an HPC solution. Thus we have two kinds of HPC solutions that relate to computations that take a long time: set up a desktop that

Figure 1: Control Panel on a Window 7 Desktop that is about to run out of RAM.



can churn away without interruption for a long period of time – or take steps to significantly speed up the computation. Each of these is strategies is discussed below.

1.2.1 Running for an Extended Period

This is often the best solution if you are making use of some proprietary or extremely complex software that would be difficult to run in parallel or code in a different, faster language. It should be strongly preferred if the program only needs to run for a few weeks or less. I have personally run programs for up to two months at a time – it may feel like an eternity but it gets the job done and is still relatively short on the cosmic scale of things. Here are some considerations when taking this approach:

1. The most important thing you can do in this situation is to plan your computing schedule to fit with your other tasks and responsibilities. If you are generally not able to get a lot of work done on the weekends because of family responsibilities, then run your analysis over the weekend to minimize the time you have to wait for results. This is often the case with Social Network modeling which often is a 12-48 hour process to run a single model. If you plan your work flow carefully then waiting for your computer can have a minimal impact on your productivity.

2. If you will need to run an analysis for a long time then it is preferable to have a dedicated desktop that you can leave in a place where it will not be disturbed. There is nothing worse than losing a few days work because you closed your laptop or the program got closed by somebody else.

3. For very long running jobs, and uninterrupted power supply is a must. These batteries will prevent you from losing months of work to a power interruption. To this end, keeping your computer in your university office is often the best plan (so long as it locks securely), as power is often very reliable due to most campuses having their own power plant and backup power sources. It is also

important to disable automatic updates on your computer so it does not restart itself in the middle of running your program. In this vein, you should also keep all other programs closed on the computer to prevent them crashing and forcing you to re-reun your analysis.

4. You should also disable remote access or take steps to protect your computer from intrusion attempts through RDP and VNC (see Section 2.4) as leaving a computer exposed to the internet for a long period of time invites malicious activity.

1.2.2 Write Faster Code

1. The first way to speed up whatever you need to do is to write it in a faster programming language. You can get up to about a 2,000 x speedup in some cases by going from R to C++ or Fortran. This approach has a major benefit of getting a huge speedup and still running on your own computer but has a major downside in that you need to learn a new programming language (in most cases) or atleast get familiar with a new language which may take a while in itself until you are used to picking up new languages. For reference, a speed comparison between C/C++, Java, Cython and R is provided in Figure 2.

2. In general, if you are working in R, the best choice for getting a speedup is C++ as the two languages have a really nice integration by way of the RCpp package. This allows you to easily pass data between R and C++ and is what I use personally. For any really complicated programming tasks, this can lead to a very steep learning curve as C++ behaves in much more complicated ways than R. Here is a list of resources for gettign started with RCpp:

   (a) Dirk Eddelbuettel has a nice website introducing Rcpp (which he is an author of) here: http://dirk.eddelbuettel.com/code/rcpp.html

   (b) Hadley Wickham has a really nice introduction to RCpp here: http://adv-r.had.co.nz/Rcpp.html

   (c) Tutorial on using RCppArmadillo to import arrays directly from R into C++: http://markovjumps.blogspot.com/2011/12/r-array-to-rcpparmadillo-cube.html

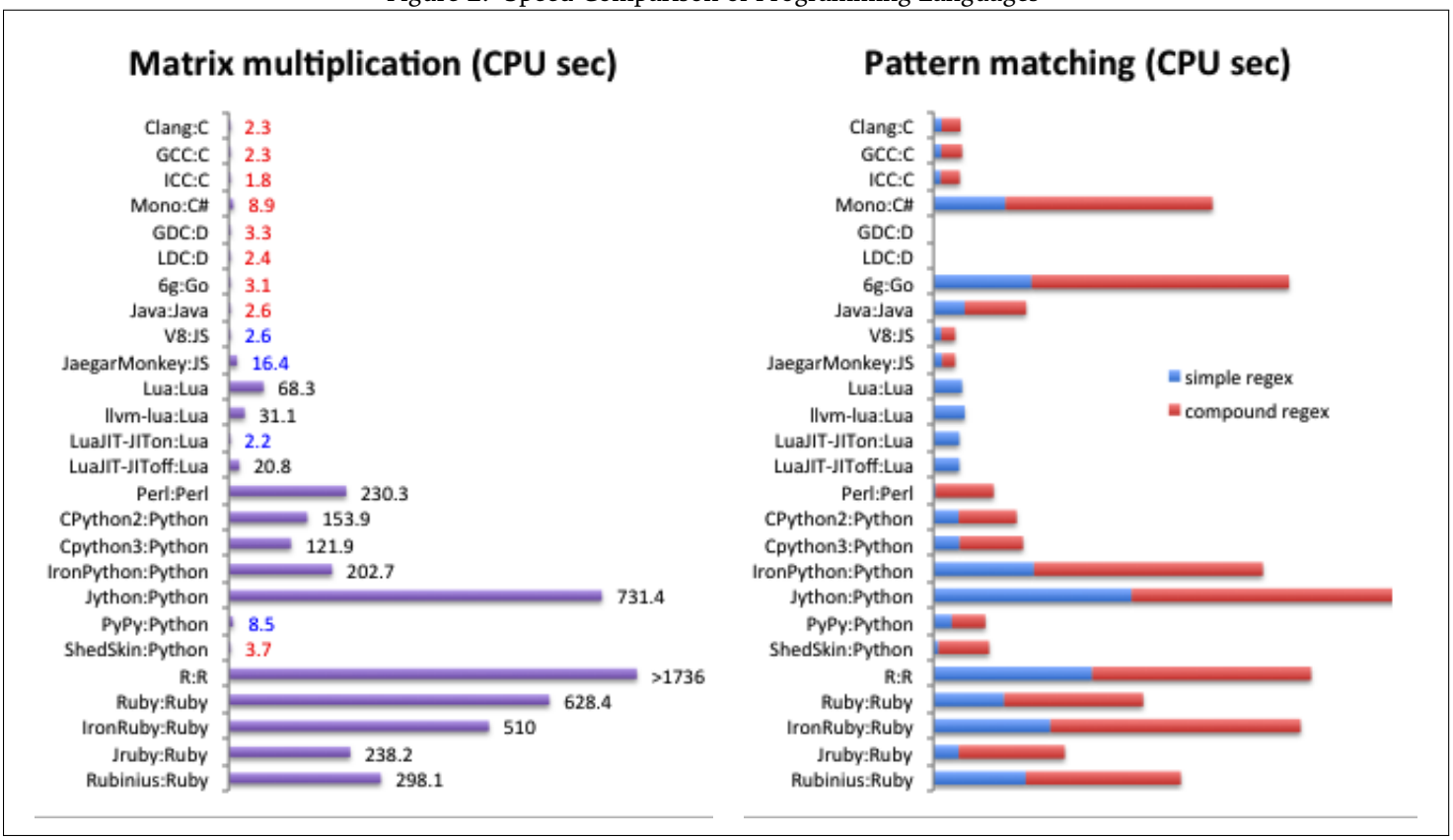
   (d) A nice quick reference guide to RCpp with commonly used code snippets: http://cran.r-project.org/web/packages/Rcpp/vignettes/Rcpp-quickref.pdf

1.2.3 Parallelize

Many data management and analysis tasks require repeating a calculation for every observation or set of observations, or repeating an analysis meany times (such as bootstrapping or simulation). These tasks can take a very long time (often months or years of CPU time) but are often amenable to being broken down into pieces that can be handled simultaneously by many processors at once. If this is possible, one can often achieve a speedup in computation that is nearly linear in the number of processors used. For example, if you have a set of 1000 simulations you need to run (so you can average over them), and each one takes one processor one day, if you were to run this on a single processor, this would take roughly 3 years. However, running this simulation distributed evenly on 40 processors would only take 25 days. Access to multiple processors can dramatically speed up certain tasks through a procedure called parallelization.

There are essentially three different kinds of parallel processing tasks. Some are very easy to implement (making it trivially worth your time) and some are so challenging to implement you will need a computer science collaborator to pursue such a project.

Figure 2: Speed Comparison of Programming Languages



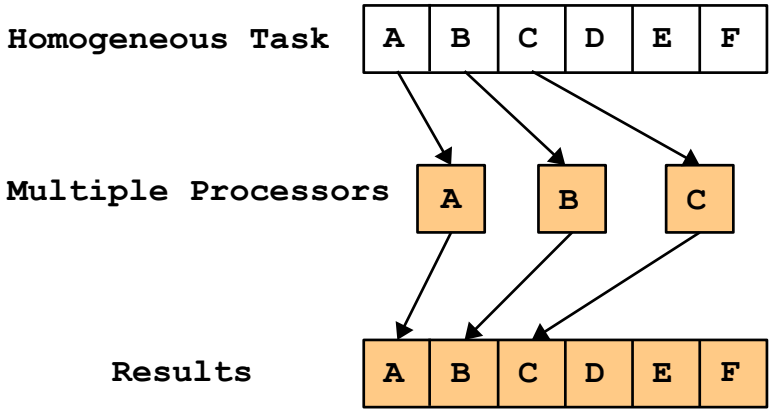
1. **Implicit Parallelization:** is a process where you take a for loop where each operation does not depend on any others and split up its calculation across multiple cores. This can be done very easily in many software packages as it only requires you to change how you write a for loop. An example in R is provided below:

```
require(doMC)
require(foreach)
nCores <- 8 #register the number of cores on your computer
registerDoMC(nCores) #set number of cores

results <- foreach(i=1:100, .combine=rbind) %dopar% {
    result <- function(i)
}
```

   One of the limitations of this approach is that it becomes difficult to run on more than one physical machine, so you are limited to the number of cores on your laptop/desktop. This is still often worth it, as you can usually get a 4-8x speedup which can reduce run time from a week to less than a day. It also usually requires that the function you want to run be mostly a function of the index, and use common data. However, this is simple, effective and easy to implement, and will most often get the trick done. Consider this for bootstrapping and some simple simulation tasks. This form of parallelization is illustrated in Figure 3.

2. **Explicit Parallelization:** involves allocating a number of cores to be used, each for their own task, and then assigning tasks explicitly to these cores. This is best if you want to repeat a process with a bunch of different datasets or just do a bunch of different tasks at once. The advantage is that you can assign any task you want to each process and they need not share anything in common. The

Figure 3: Implicit Parallelization.



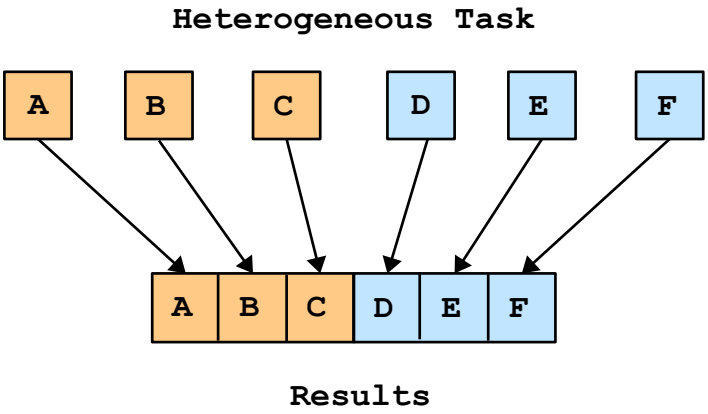
downside is that this requires more management on your part, and requires a separate R instance for each process which can eat up memory. Example code for explicit parallelization in R is provided below:

```
library(snowfall) #load the snowfall package
numcpus <- 4 #set the number of cpus to use
sfInit(parallel=TRUE, cpus=numcpus ) #initialize the snowfall cluster

if(sfParallel()){
       cat( "Running in parallel mode on", sfCpus(), "nodes.\n" )
}else{
       cat( "Running in sequential mode.\n" )
}

#export all packages currently loaded in your R session
for (i in 1:length(.packages())){
          eval(call("sfLibrary", (.packages()[i]), character.only=TRUE))
}

#export a list of R data objects that your function will need
sfExport("Object1","Object2","Object3")

#apply a function across the cluster
result <- sfClusterApplyLB(indexes,Function)

#stop the cluster (this needs to be done explicitly)
sfStop()
```

Figure 4 provides a visual representation of explicit parallelization. This is the most commonly used form of parallel computing for HPC because it is easier to distribute across many different physical machines and the tasks do not have to be done in the exact sequence they are assigned.

3. **Map-Reduce:** is a more complex paradigm where parts of a problem that are amenable to parallel processing can be mapped onto multiple processors and then recombined at intermediate or final steps to advance the solution to a problem. Steve Krenzel has a nice, detailed description here: http://stevekrenzel.com/finding-friends-with-mapreduce. This paradigm was originally implemented by Google as a way to efficiently deal with huge amounts of data in parallel. The way I would have

Figure 4: Explicit Parallelization.



you think about it is a bit of an abuse of the name, but captures the idea that some parts of an analysis of data management problem can be efficiently parallelized while others cannot. Figure 5 details a repeated parallel processing strategy where local updates are calculated in parallel and global ones are calculated serially. This represents one of the more challenging things to implement efficiently but some problems require it. For example, if we wanted to run a simulation where most information is local to groups but they periodically receive global information, this is the kind of strategy we would need to adopt.

(a) Hadley Wickam (one of the most prolific R developers out there) has a nice paper on the Split-Apply-Combine paradigm available here:
http://www.jstatsoft.org/v40/i01/paper

(b) Revolution Analytics has a package to perform MapReduce in R. You can check out hte documentation here:
https://github.com/RevolutionAnalytics/rmr2/blob/master/docs/tutorial.md

In addition to the resources listed above there are a few nice general resources on parallel computing that I have presonally used in the past that are listed below:

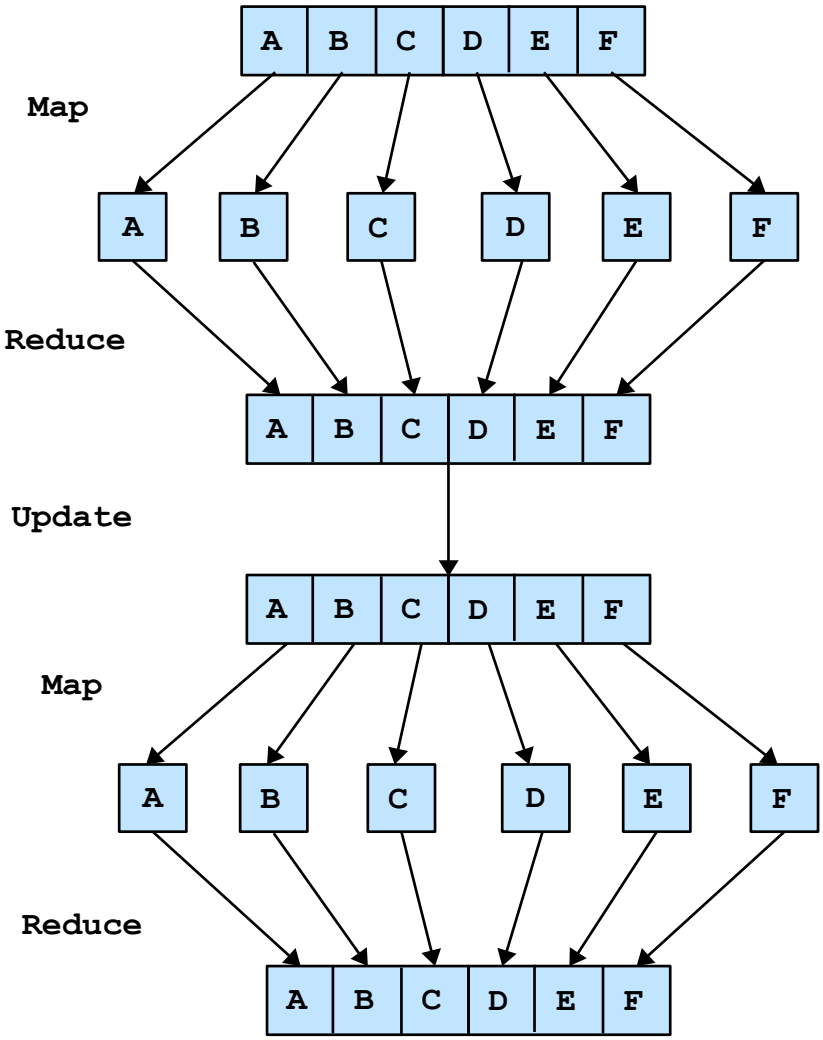
1. Ryan Rosario has a really nice introduction to paralellization in R available here: Slides
2. Ryan Rosario also has some nice slides on parallel computing in R: Slides
3. There is a nice tutorial with examples on using snowfall to create a parallel cluster in R available here: http://www.sfu.ca/~sblay/R/snow.html

2 Remote Access

Once you have decided on a strategy for dealing with your HPC problem, you need a way to execute that strategy. This will most often involve the use of a dedicated desktop, workstation or server/cluster. If the problem you are working on takes a predictable amount of time to compute on your home desktop, say a week, then this is just a matter of setting the computer to run your program and coming back in a week to check the results and enter the next task. If the computer is located in a place where you do not run into it every day, then you are performing the most reliable and simplest form of remote access (the in-person kind). For many problems, this is actually the optimal strategy, and one I have used myself on many occasions.

Setting up computer infrastructure to gain access and control of a computer from a different location is

Figure 5: Repeated Map-Reduce.



the more standard definition of **Remote Access**, and for a number of problems is something you may want to do. As an example, you may have a program you want to run, but do not know exactly when it will complete. If the computer is in a location that might take some effort to get to, going to check on it every day might become cumbersome. In this case, you will want to set up remote access to that computer so you can check on things from your laptop. The more typical need most people have for remote access is when the computer they need to use is in a location they do not have physical access to. This is the case for the UMass cluster computing resources at MGHPCC. In this situation, your ability to access the computer remotely will determine whether you can use it at all.

This section will outline the two main methods of gaining remote access to a computer over the internet, detail how you can set up Windows, Mac and Linux machines to accept remote access and control, and provide resources for gain access to and using the UMass cluster computing resources at MGHPCC.

2.1 Allowing Remote Access to a Computer

Computer operating systems do not allow just anyone to gain control of a computer over the internet by default. This is to prevent Russian hackers from turning your computer into a spam bot. Yet if you have a desktop, you may want to make it available to access from anywhere for you and only you. You will need to take two steps to do this– the first is giving your computer an address where it can be found, the

second is configuring your computer to allow access. The first step can be accomplished in one of two ways:

1. If you are a faculty member or can demonstrate a real need for one, OIT may grant you a static IP address. This is a constant, unchanging address for your computer that will allow you to contact it from anywhere in the world easily. This is a valuable resource for the university as they only have a limited number, but they are willing to make them available. You can email hostmaster@oit.umass.edu to inquire about getting a Static IP address. They will need a bunch of information about your computer but are good at explaining what they need. Note that this strategy will only work if your computer is stored on campus and comes with some added security challenges as a Static IP address will make you a target for hacking. That said, this is the gold standard and the best option if you can get it.
2. The second option, which will work anywhere, is to use a dynamic DNS service like DynDNS (http://dyn.com/remote-access/). It costs $25 a year, but will allow you to assign a unique identifier to your computer that will be continually updated as your computer's IP address changes (which it naturally does). This option should keep you safe from most major security threats and will let you login to a home computer but most reputable services cost something (I use DynDNS). They have pretty good directions on their website so just follow them when you sign up and you should be ready to go in a few minutes.

Once you have secured a static or dynamic IP address, you now need to allow access to your computer. This is detailed for different operating systems below:

1. In windows this is pretty straightforward, you just navigate to **Control Panel** → **System Properties** → **Remote** and check the box as shown in Figure 6. There is a more detailed tutorial for doing this available here: http://www.howtogeek.com/howto/windows-vista/turn-on-remote-desktop-in-windows-vista/
2. On a Mac desktop, follow the directions on the following links (Maverics): http://support.apple.com/kb/PH13759 for ssh and http://support.apple.com/kb/PH14130 for RDP.
3. Ubuntu and all Linux distros more generally also make things dead easy, and generally allow the most flexibility in accessing them. Check out this tutorial: http://www.makeuseof.com/tag/ubuntu-remote-desktop-builtin-vnc-compatible-dead-easy/

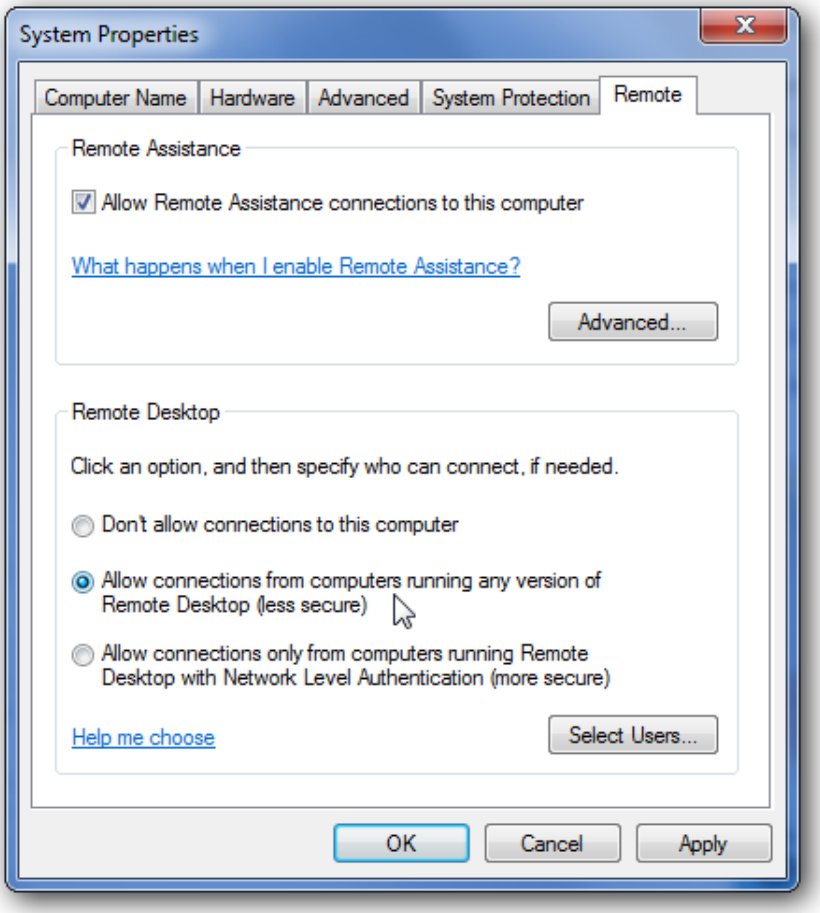
Once you have done this you should be ready to go. One thing to note if you are using a home computer and you have a router with a firewall, you may need to configure port forwarding to allow connections through your firewall. This may be a bit challenging so you will need to consult the internet for help with you particular router.

2.2 Adding Security for A Static IP

If you have a Windows machine with a static IP, you should be alright and should not spend time worrying about security threats due to remote access being turned on. This is also because it is much more difficult to do something about it if you are using Windows because it does not have as nice of a command line interface. If you are using a Linux machine with a static IP located on campus, criminals will start to ping your machine to try and guess your password to login and gain remote control of your computer. This can be very bad. Fortunately you can do something about it that will make your computer totally safe – you can restrict access to your computer to IP address on the UMass network. If anyone is stupid enough to try to hack your computer from the campus network, OIT will catch them (they are watching for this activity 24/7 and they take severe disciplinary action including expulsion if it is a student (which they can do because they know the identity of every user).

The following directions are written for CentOS 6 and may need to be modified for your Linux distro, but all you will need to do is look up the location of your iptables file. Open up a new terminal and type in the following (or equivalent on your system)

Figure 6: Enabling Remote Connection in Windows.




```
sudo vi /etc/sysconfig/iptables
```

This should open up a file that looks something like this in the vi editor (for a tutorial on using vi, check out this website http://staff.washington.edu/rells/R110/):

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
# you will be inputting your new rules here!
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
COMMIT
```

You will need to copy the following chunk of code into the iptables file in the position marked above. This will set your computer to only allow udp and tcp connections from the UMass IP block. The first

four lines set up a nonstandard port number, this can make you even harder to find. Note that this is not supported by OIT, but people who work there do this themselves.

```
-A INPUT -m state --state NEW -m udp -p udp -s 128.119.0.0/16 --dport 32576 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp -s 72.19.64.0/18 --dport 32576 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp -s 128.119.0.0/16 --dport 32576 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp -s 72.19.64.0/18 --dport 32576 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp -s 128.119.0.0/16 --dport 5900 -j ACCEPT
-A INPUT -m state --state NEW -m udp -p udp -s 72.19.64.0/18 --dport 5900 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp -s 128.119.0.0/16 --dport 5900 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp -s 72.19.64.0/18 --dport 5900 -j ACCEPT
```

Once you have saved your changes, type in the following commands into your console to restart your iptables with the new settings enabled. You will want to test that this worked by trying to login from outside the UMass network.

```
service iptables restart
```

You should now be pretty safe from outside intrusion. To login from outside the Umass network you will need to use a VPN that will fool your computer into thinking you are on the UMass network. You can set up a VPN by going to this website: http://www.oit.umass.edu/vpn . You will also need to get in touch with OIT and have VPN added to your account services.

2.3 Command Line

You can connect to a remote computer very easily and extremely securely using SSH (Secure SHell). However, this requires you to know how to use Bash or whatever command line environment your OS uses. This is a hugely valuable skill, but not the focus of this tutorial. There are a ton of resouces for learning Bash and SSH on the internet. A few of them are listed here:

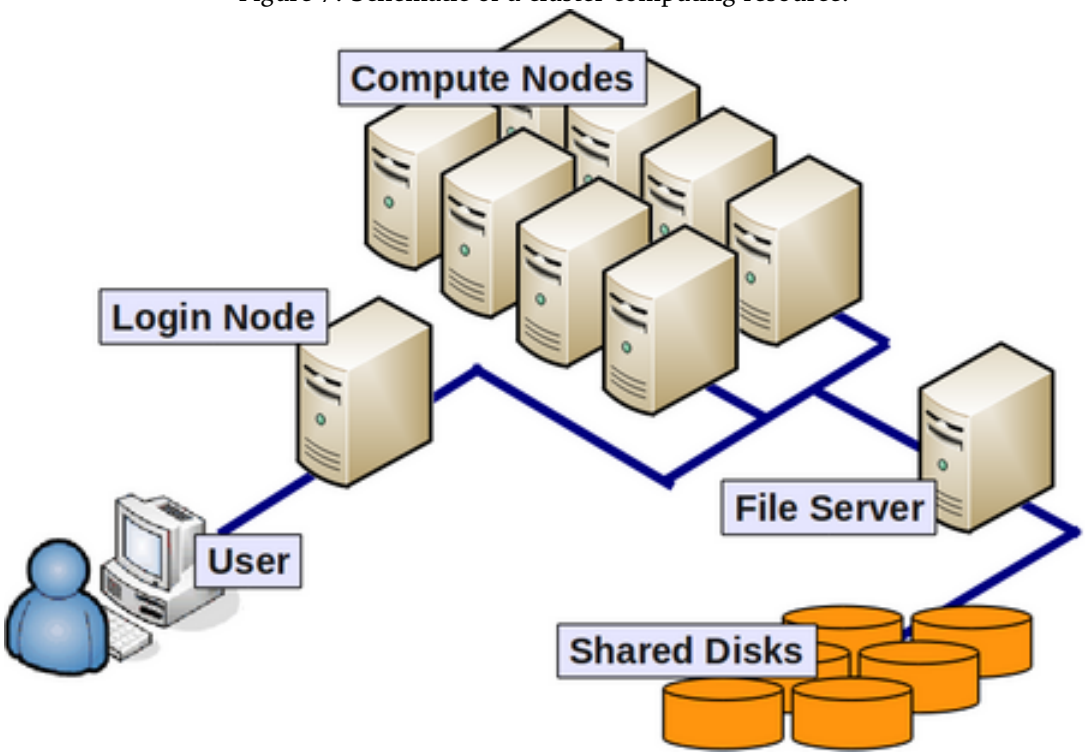
1. A nice tutorial from Wikihow: http://www.wikihow.com/Use-SSH
2. A whole bunch of tutorials in SSH and ways of copying files to and from your remote computer using FTP and SCP from siteground: http://www.siteground.com/tutorials/ssh/
3. An example connecting using SSH from Mediatemple is available Here.
4. Some common pitfalls people run into while programming in Bash and fixes: http://mywiki.wooledge.org/BashPitfalls

Most clusters are only accessible over SSH so this is the way to go in terms of learning a new skill that you can translate into bigger problems in the future.

2.4 RDP and VNC

Remote Desktop Protocol (**RDP**) and Virtual Network Computing (**VNC**) are two protocols for connecting to your computer that will give you a window that looks just like you were sitting in front of the computer screen. It makes remote computing dead easy but requires a faster internet connection so is not always ideal when traveling. That said it is good for many quick tasks. Windows and Linux both have free clients for accessing another computer but I really think a paid client is worth the $20 or less you spend on it. I use JumpDesktop for logging in to my personal cluster and have found it to be fast and easy to set up. It is also free for windows and is available here: http://jumpdesktop.com/. If you go this route there is not much more to explain. You simply use the remote computer as if you were sitting right in front of it and do whatever you need to do. A note of caution though – this does not build any skills and will mean that if you need to run your analysis on a bigger cluster, you will have to learn SSH anyway.

Figure 7: Schematic of a cluster computing resource.



2.5 Setting Up Access to Your Campus Cluster

The first thing you need to do to get access to a cluster is determine whether your university/organization even has a cluster resource available. You will want to find the associated web page and contact someone listed as an administrator before you go any further to ask them if you can have access and how you would go about doing that. You may have to get faculty sponsorship, pay a fee, or just fill out a form. You will also have to do a fair bit of research to learn about the norms and rules of using your local cluster. Additionally, each cluster will have its own job scheduling software. This software provides the administrators with a way to ensure that resources are distributed fairly across users and that users can request an appropriate amount of resources. You will need to consult the help page for the job scheduler/cluster you are using as there are more different languages than I can possibly go into here. What follows is just an illustrative example (with helpful links) for logging in to the UMass cluster resource to get you started with how you might go about doing the same at your home university.

UMass Amherst currently owns and maintains a 5312 core cluster with 30+ TB RAM and 400+ TB Shared high performance storage as part of the Massachusetts Green High Performance Computing Center (**MGHPCC**), located in Holyoke, MA http://www.mghpcc.org/. This cluster is available to UMass Faculty and Graduate students for HPC use and has a fairshare job queue that allows researchers who have not purchased their own cluster resources to use it as well. Current costs for buy-in are around $100k for a 512 core AMD rack. What follows are instructions for getting started with setting up an account and submitting simple jobs in **R**.

1. If you are a Principal Investigator (UMass Faculty), simply go to this page: https://www.umassrc.org/hpc/ and fill out the form, you should be granted an account within a week. If you are a graduate student, you need to get your adviser to make an account first (or find a faculty member who is willing to be your PI), have them create an account and then list them as your PI when you make an account request.

2. MGHPCC only allows connections from the UMass campus network so if you want to connect from your laptop off campus, you will need to do so through a virtual private network (**VPN**) that

essentially routes all of your internet traffic through an encrypted tunnel to the UMass network. You can set up a VPN by going to this website: http://www.oit.umass.edu/vpn . You will also need to get in touch with OIT and have VPN added to your account services. If you are not approved for some reason, you will just be limited to accessing the cluster while you are physically on campus.

3. Once you have an account set up, you will have to use **ssh** to login to the cluster remotely. If you have a Mac or Linux machine you will have this available by default, but if you are on Windows you will need to install PUTTY by clicking on the following link: http://www.chiark.greenend.org.uk/ sgtatham/putty/. Once you have Putty installed (if you are working in Windows) you can click on the icon to open a PUTTY terminal. If you are using a Mac/Linux computer, simply open a terminal. In either case, you can now type the following into your terminal, which should then prompt you for a password if you are successful:

   ```
   ssh username@ghpcc06.umassrc.org
   ```

   Once you have successfully logged in, you can change directory to your home directory:

   ```
   cd /home/username
   ```

   You can now copy files over using scp and then submit jobs to the cluster using LSF. A detailed tutorial for this entire process is linked to as the next item on this list.

4. For a nice tutorial on how to login and submit jobs to the MGHPCC presented by Andrea S Foulkes, Gregory J Matthews, Nicholas G Reich as part of an ICB3 Big Data workshop, follow this link: http://www.beyondeconomy.net/Files/ICB3_MGHPCC_Login_Slides.pdf

3 Hardware

One of the big questions you will have to answer as you get into HPC is whether or not to make an investment in your own dedicated hardware. This is an important consideration and will be different depending on your use needs and financial resources. This section will outline some considerations for whether owning your own is a good idea and paths to upgrading a current desktop or purchasing your own desktop/workstation/cluster.

In general, if you only plan to work sporadically on projects requiring HPC, purchasing your own hardware is a bad idea. It may look cool in your office but it will keep you from doing more professionally rewarding things like attending conferences. It is also very inefficient. If you plan to do more HPC work, but not for another couple of years, it is also a very bad idea to purchase hardware now. Moore's Law states that we get about double the computing power (for about the same cost) every two years (http://en.wikipedia.org/wiki/Moore's_law). You should not buy hardware unless you plan to use it NOW and use it pretty consistently for the next several years.

If you meet these criteria then you need to ask yourself at what scale and time-frame you need to do your computing. MGHPCC is free and will give you access to more computing power than you can ever hope to own yourself. If you can do it in R, then you should go this route first. If you absolutely need Stata, SAS or Matlab, then you will need to look at your own hardware. The first thing you should definitely consider is upgrading an existing Desktop if you have one. This will usually do the trick for most HPC problems and is WAY cheaper than buying new hardware. What follows are a series of section on purchasing hardware for different needs and price ranges. If you think you need something more than an expensive workstation, then you should not be buying it yourself. This is a job for the institution and you will likely need a grant to buy in. See about MGHPCC options before even considering this route. Do not get more than you need, it will cost you in the long run.

3.1 Upgrades you Should Make To Your Desktop (Under $1000)

If you own and desktop, you are probably in luck! There are a bunch of cheap upgrades you can make that will often let you work on the problem you want to without the need to buy a new machine. These are discussed below:

1. Buy more RAM. This is the single most common limiting factor in conducting your analysis. This will let you work with more data, faster. Check your motherboard specs and see how many DIMM slots you have available. RAM is easy to install yourself so this is the simplest upgrade route. Most computer motherboards have 2 or 4 slots for RAM. You can currently get 8GB sticks for a total of 16 or 32GB of RAM total for under $300. This will let you work with up to about 1,000,000,000 total observations (32 GB) in R (a 1,000,000 x 1,000 data frame), which should be enough for most tasks. This will also let you work with moderately large data sets in parallel.
2. Buy a faster processor. You will need to take your computer to an actual shop for this, but for under $400, you should be able to get a fast 4 core Intel processor. You will have to check with the shop as to what you can upgrade to, but this can sometimes double or triple the speed with which you can process things. Do not take this route if your computer has an AMD processor or if the processor is older than a Sandy Bridge (first core i7) Intel processor (see this link for details: http://en.wikipedia.org/wiki/List_of_Intel_CPU_microarchitectures.
3. Buy a Solid State hard drive (SSD). These are getting much cheaper now and will provide a huge boost to the general speed of your system. This will only have a real tangible effect on the performance of R if you read and write lots of data, but will improve overall system performance quite a lot.

3.2 Desktop (Under $2,500)

Get help from somebody with more experience than you before you even think about buying a new desktop. There are a lot of factors to consider when buying a new desktop for personal HPC but you can really screw this up. Here are some general guidelines.

1. Only buy a computer with an Intel processor. AMD processors are so much slower per core that they are not worth it . Intel processors tend to have less physical cores but each can run tow threads simultaneously, effectively doubling their performance. AMD systems look cheap for what you get but are a really bad idea, just trust me. Try to go for a 4 core Intel at a minimum but a 6 or 8 core is better if you can afford it.
2. Look for a motherboard with 6 or 8 RAM DIMM slots. This will let you go up to 48-64 GB of RAM and will give you room to grow. This is definitely the sweet spot.
3. Get an SSD with atleast 240GB capacity – this is just a no-brainer at this point. Get a 4TB HDD as a data drive if you want more storage.
4. Do not skimp on the power supply if you are building a system yourself. You want as much power headroom as possible.

These are the vendors I would recommend (not an exclusive list). Don't go with a major vendor as they will rip you off for memory and processor upgrades.

1. IBuyPower is who I went through for my own desktop. They are well priced and the system has held up well: http://www.ibuypower.com/
2. Puget is a bit more expensive but has nice quality stuff: http://www.pugetsystems.com/
3. Origin systems look nice and have a high build quality but cost a pretty penny: http://www.originpc.com/

3.3 Workstation (Under $20,000)

Do not make choices about a workstation on your own. Consult somebody who owns one and ask a ton of questions to the sales rep you talk to. If you do not take this seriously, you can waste a biblical amount of money. You should only purchase an HPC workstation if you can consistently put it to work on intensive problems. They are mostly good for high-RAM applications where a cluster administrator might get mad at you for hogging resources (up to 512GB RAM per thread) and constantly running and testing lots of mid-size simulations (20-40 cores). Plan to spend at least $7,500 if you go this route but $12-15k tends to get you the best value.

1. Microway is the system integrator I went with for my HPC workstation. They are absolutely phenomenal and based in Massachusetts. They offer a ridiculous amount of support and can be purchased at a steep discount through a UMass agreement with the company: http://www.microway.com/
2. Titanus would have been my next choice for a workstation, they are based in Florida and have very competitive pricing: http://www.titanuscomputers.com/

4 General Resources

R Resources for HPC:

1. If you have not visited the QuickR site already, this should be your first stop: http://www.statmethods.net/
2. If you are interested in using R for HPC, this is the best place to start: http://cran.r-project.org/web/views/HighPerformanceComputing.html
3. A nice tutorial introducing the functionality in SnowFall: http://www.imbi.uni-freiburg.de/parallel/docs/Reisensburg2009_TutParallelComputing_Knaus_Porzelius.pdf
4. A free online e-book with lots of examples for conducting time-series analysis in R: http://a-little-book-of-r-for-time-series.readthedocs.org/en/latest/src/timeseries.html
5. The **biglm** package is a memory efficient implementation of glm that only uses $O(n^2)$ memory. http://cran.r-project.org/web/packages/biglm/biglm.pdf
6. The StatsTeachR package (maintained by ICB3 affiliates at UMass Amherst) has a nice set of tutorials for advanced computing in R available here: http://www.statsteachr.org/

General HPC Resources:

1. A nice set of slides and resources for statistical data mining developed by Adrew Moore at CMU: http://www.autonlab.org/tutorials/
2. UC Berkeley AMPCamp big data analytics tutorial: http://ampcamp.berkeley.edu/big-data-mini-course-home/
3. The SNAP lab at Stanford makes available serveral nice pieces of software and Python and C++ libraries here: http://snap.stanford.edu/
4. The wiki page for the UMass Amherst portion of the Massachusetts Green High Performance Computing Center cluster resource can be found here: http://wiki.umassrc.org/wiki/index.php/Main_Page

5. If you are adventerous and have a bunch of old computers lying around, you may find some traction in building a Beowulf cluster linking them together. Check out these links to see if this option makes sense for you. The main problem with a Beowulf cluster with old hardware is that each processor can be so slow that you do not really get any performance gain.
   http://en.wikipedia.org/wiki/Beowulf_cluster
   http://fscked.org/writings/clusters/cluster-1.html
   http://en.wikibooks.org/wiki/Building_a_Beowulf_Cluster