

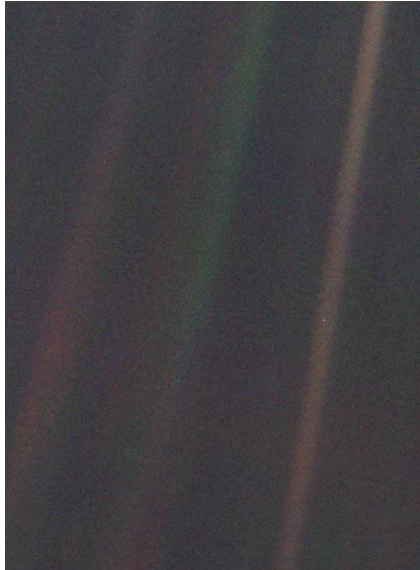
Spring 2012 BMTRY 789-02

Parallel Processing in R

Adrian Michael Nida

DBE

2012-04-03



Outline of Talk

- Introduction
- Cluster
- Parallel Processing

"The time has come," the Walrus said, "To talk of many things:..."

– Lewis Carroll *Through the Looking-Glass and What Alice Found There*

Introduction

- ▣ UNIX != Windows
- ▣ History
- ▣ Executable Syntax
- ▣ Common Commands
- ▣ Editing Files
- ▣ Secure Shell (ssh)
- ▣ Source Control (optional)

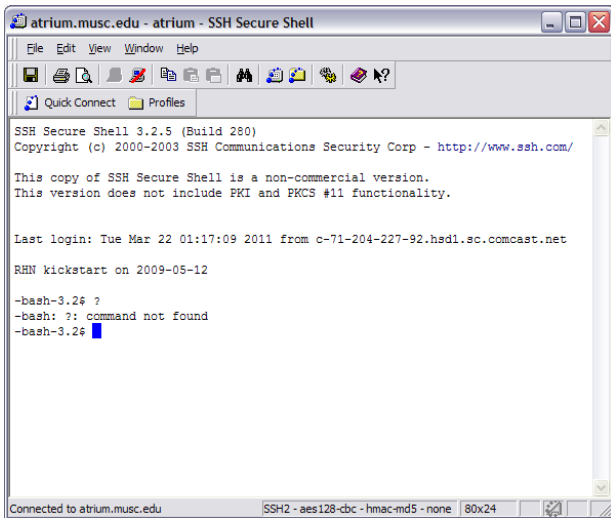
"Sure, Unix is a user-friendly operating system. It's just picky with whom it chooses to be friends."

– Ken Thompson

UNIX != Windows



UNIX != Windows (cont.)



The screenshot shows a terminal window titled "atrium.musc.edu - atrium - SSH Secure Shell". The window has a menu bar with "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with various icons for file operations and system functions. The main area of the window displays the following text:

```
SSH Secure Shell 3.2.5 (Build 280)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.

Last login: Tue Mar 22 01:17:09 2011 from c-71-204-227-92.hsd1.sc.comcast.net

RHN kickstart on 2009-05-12

-bash-3.2$ ?
-bash: ?: command not found
-bash-3.2$ █
```

At the bottom of the window, there is a status bar that reads "Connected to atrium.musc.edu" on the left, "SSH2 - aes128-cbc - hmac-md5 - none" in the center, and "80x24" on the right.

A History of UNIX



The history

Executable Syntax

`'/path/to/program [options] [files]'`

where:

- ▢ program is the name of the program you wish to run
 - ▢ `/path/to` is used to specify where on the filesystem program is located (Hint: If this location is in your `$PATH`, you won't need to type it) (Another Hint: The current directory `'.'` is **NOT** in your path, so to execute things there you must type `'./program'`)
- ▢ options are "switches" passed into the program to alter its code flow.
 - ▢ They can start with `'-'`, `'- -'`, or nothing at all.
- ▢ files are the files your program requires to run. This can be none at all.

man [program]	Displays help for a command (try 'man man', 'man hier')
cd [directory]	Change to directory
mkdir [newdir]	Make a directory in the current directory
ls [-lha] [directory]	(Li)st contents of directory
cp [-ra] SOURCE DEST	copy SOURCE to DEST
mv SOURCE DEST	copy and then delete SOURCE to DEST
rm [-rf] file(s)	REMOVE file(s)
chmod [-R] ugo file	Change mode (permissions) of a file (x=1, w=2, r=4)
chown [-R] owner:group file	Change Owner (and group)
find [directory] -option PATTERN	Search for files matching option's PATTERN
head tail [-n lines] [file]	print first last lines of file
grep [-inrv] PATTERN file(s)	Search for pattern in file(s)
sed [-i] 's/FIND/REPLACE/[g]' [file]	find & replace in 'stream'
awk 'FS=":"' print \$1, \$6' [file]	print 1st & 6th fields of file
exit	End CLI session
> >> 2& > 1	piping and STD[IO ERR] redirection

version 1.1
April 1st, 06

vi / vim graphical cheat sheet

Esc
normal mode

~ toggle case	* external filter	@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	% next ident	(begin sentence) end sentence	_ "soft" bol down	+ next line
* goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto format
Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.
q record macro	w next word	e end word	r replace char	t 'till	y yank	u undo	i insert mode	o open below	p paste after	* misc		* misc
A append at eol	S subst line	D delete to eol	F "back" find ch	G eof/ goto ln	H screen top	J join lines	K help	L screen bottom	. ex cmd line	^ reg. spec		bol/ goto eol
a append	s subst char	d delete	f find char	g extra cmds	h ←	j ↓	k ↑	l →	* repeat t/T//F	' goto mk. bol		\ not used!
Z quit	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid?	< un-indent	> indent	? find (rev.)			
z extra cmds	x delete char	c change	v visual mode	b prev word	n next (find)	m set mark	reverse t/T//F	* repeat cmd	/ find			

- motion** moves the cursor, or defines the range for an operator
 - command** direct action command, if red, it enters insert mode
 - operator** requires a motion afterwards, operates between cursor & destination
 - extra** special functions, requires extra input
 - q** commands with a dot need a char argument afterwards
- bol = beginning of line, eol = end of line, mk = mark, yank = copy
- words: `quux(foe, bar, baz)`
 WORDS: `quux(foe, bar, baz)`

Main command line commands ('ex'):
 :w (save), :q (quit), :q! (quit w/o saving)
 :e f (open file f),
 :%s/x/y/g (replace 'x' by 'y' filewide),
 :h (help in vim), :new (new file in vim),

Other important commands:
 CTRL-R: redo (vim),
 CTRL-F/-B: page up/down,
 CTRL-E/-Y: scroll line up/down,
 CTRL-V: block-visual mode (vim only)

Visual mode:
 Move around and type operator to act on selected region (vim only)

- Notes:**
- use 'x' before a yank/paste/del command to use that register ('clipboard') (x=a..z.) (e.g.: "ay\$ to copy rest of line to reg 'a')
 - type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
 - duplicate operator to act on current line (dd = delete line, >> = indent line)
 - ZZ to save & quit, ZQ to quit w/o saving
 - zt: scroll cursor to top, zb: bottom, zc: center
 - gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

Taken from: Vlemu

Secure Shell (ssh)

- 📖 To connect to another computer, you will need to use this program from the OpenSSL group.

```
ssh [-1246AaCfgkMNnqsTtVvXxY] [-b bind_address] [-c  
cipher_spec] [-D [bind_address:]port] [-e escape_char] [-F configfile]  
[-i identity_file] [-L [bind_address:]port:host:hostport] [-l  
login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port] [-R  
[bind_address:]port:host:hostport] [-S ctl_path] [-w tunnel:tunnel]  
[user@]hostname [command]
```

- 📖 There are Windows alternatives
 - 📖 PuTTY
 - 📖 SSH Secure Shell (™)

Source Control

- When working between many computers, you will eventually have to organize your documents so changes get passed correctly.
- Source Control allows one to "check [in|out]" versions of documents in ways that allow a revisionist history.
- Subversion was the SCM used by DBE
 - svn co <https://projects.dbbe.musc.edu/nida/School/>
 - This server is DOWN at the moment :(
 - svn status
 - svn up
 - Make Changes
 - svn diff
 - svn add [file]
 - svn ci -m 'Message'
- <http://tortoisetsvn.tigris.org/> is a well received Windows client.

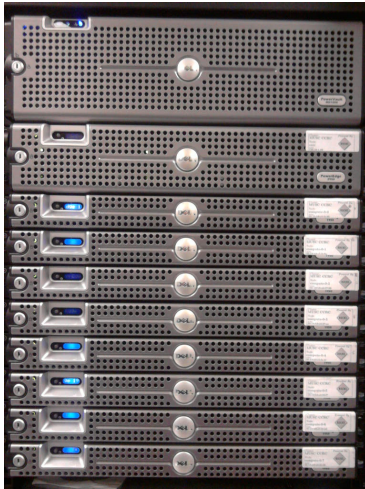
Cluster

- ▣ Hardware capabilities
- ▣ User Accounts
- ▣ Environment

"Imagine a Beowulf cluster of these!"

– Anonymous (Coward) Slashdot Troll

Hardware capabilities



The Cluster's Homepage

User Accounts

- Accounts (should) have been created for all of you
- Synched with University's Lightweight Directory Access Protocol (i.e., same NetID/Password combo you already know)
- Very few have the keys to the kingdom (i.e., sudo access)

Environment

/export (mounted from all nodes)

- 📁 apps
 - 📁 ...
 - 📁 R
 - 📁 R-2.1.0
 - 📁 R-2.10.1
 - 📁 R-2.12.2
 - 📁 **R-2.8.1**
 - 📁 resources
 - 📁 ...
- 📁 bio
 - 📁 hmmer
 - 📁 ncbi

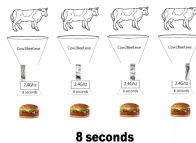
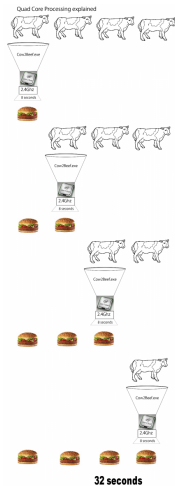
Parallel Processing

- ▣ Advantages
- ▣ Problems
- ▣ The two types

"There are 3 rules to follow when parallelizing large codes. Unfortunately, no one knows what these rules are."

– W. Somerset Maugham, Gary Montry

Advantages



Author Unknown

Problems

- ▮ Hard to implement
 - ▮ Critical Regions
 - ▮ Race Conditions
- ▮ Knowing what you can parallelize.

Two Types

- Batch Programming
- Truly Parallel

TIMTOWTDI

Two Types

- ▣ Batch Programming
- ▣ Truly Parallel

TIMTOWTDIBSCINABTE

Batch Programming

```
R CMD BATCH [options] ["--args arg1 ..."] my_script.R [outfile]
```

where my_script.R is in the form:

```
args <- commandArgs(TRUE) #Specifies only trailing args  
print(args) #Print args character vector  
...  
q(status=0) #Any other number signifies error
```

Bash Scripting Commands

Command	Description
qsub [script.sh]	Submit batch jobs
qsub -l	Submit an interactive job
qstat -u [userid]	Check status of all of your jobs
qhold [jobID]	Put a job on hold (before it starts)
qrls [jobID]	Release a job from hold status
qdel [jobID]	Delete a job, running or not

Batch Script

Very simple example:

```
#!/bin/sh  
#$ -N NameOfYourJob  
#$ -M EmailAlias@musc.edu  
#$ -m beas  
#$ -S /bin/bash  
#$ -V  
#$ -cwd
```

```
cd /path/to/where/my_script/is
```

```
R CMD BATCH [options] ["--args arg1 ..."] my_script.R [outfile]
```


An Intro to Homework

- On the class website, you will find five files.
 - Assignment (the PDF of this portion of the talk)
 - Genome input file – 50000 'Chromosome' file with 3000 'nucleotides' / 'Chromosome' (144MB)
 - mineAminos.R (the single threaded version – shown on next slide)
 - mineAminos.batch.R (the batch script version of the above file)
 - create.batchfile.R (a program that will create the batch files you will need to process through the Sun Grid Engine)

mineAminos.R (single-threaded)

```
ChromosomeLength = 3000
genome <- scan("genome.txt", what=character(ChromosomeLength))
total <- length(genome)
AminoAcids <- list()
for (i in 1:total) {
  chromosome <- genome[i]
  for(j in seq(1, ChromosomeLength, 3)) {
    amino <- substr(chromosome, j, j+2)
    if (!is.null(AminoAcids[[amino]])) {
      numAminos <- AminoAcids[[amino]]
      AminoAcids[[amino]] <- (1 + as.integer(numAminos))
    } else {
      AminoAcids[[amino]] <- 1
    }
  }
}
Names <- sort(names(AminoAcids))
for (i in 1:length(Names)) {
  cat(Names[i], paste(AminoAcids[[Names[i]]], "\n", sep=''), sep="\t")
}
print(proc.time()[3])
```

Output

```
> source("mineAminos.R")  
Read 50000 items  
aaa      780293  
aac      781510  
aag      781449  
aat      779933  
aca      779984  
...  
ttc      781373  
ttg      780609  
ttt      782149  
elapsed  
2017.413
```

mineAminos.batch.R

```
ChromosomeLength = 3000
genome <- scan("genome.txt", what=character(ChromosomeLength))
total <- length(genome)
AminoAcids <- list()

Args <- commandArgs(TRUE)
Beginning <- as.integer(Args[1])
Ending <- as.integer(Args[2])
for (i in Beginning:Ending) {

  chromosome <- genome[i]
  for(j in seq(1, ChromosomeLength, 3)) {
    amino <- substr(chromosome, j, j+2)
    if (!is.null(AminoAcids[[amino]])) {
      numAminos <- AminoAcids[[amino]]
      AminoAcids[[amino]] <- (1 + as.integer(numAminos))
    } else {
      AminoAcids[[amino]] <- 1
    }
  }
}
Names <- sort(names(AminoAcids))
for (i in 1:length(Names)) {
  cat(Names[i], paste(AminoAcids[[Names[i]]], "\n", sep=''), sep="\t")
}
print(proc.time()[3])
```

create.batchfile.R

Feel free to review this file. It is not coded efficiently, but it gets the job done. This is an example of how you should run it:

```
R CMD BATCH --vanilla --slave '--args $NumSlaves $Name $EmailAlias' create.batchfile.R
```

You will have to run it with at least three different NumSlaves so you can compare the times to the single threaded version. You will also have to sum the outputs from each run to compare them to the single-threaded version.

Let's try it ...

library("Rmpi")

```
# Load the R MPI package if it is not already loaded.
if (!is.loaded("mpi_initialize")) {
  library("Rmpi")
}

# Spawn as many slaves as possible
mpi.spawn.Rslaves()

# In case R exits unexpectedly, have it automatically clean up
# resources taken up by Rmpi (slaves, memory, etc...)
.Last <- function(){
  if (is.loaded("mpi_initialize")){
    if (mpi.comm.size(1) > 0){
      print("Please use mpi.close.Rslaves() to close slaves.")
      mpi.close.Rslaves()
    }
    print("Please use mpi.quit() to quit R")
    .Call("mpi_finalize")
  }
}

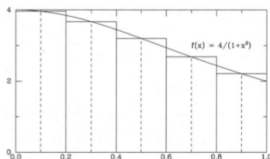
# Tell all slaves to return a message identifying themselves
Result <- mpi.remote.exec(paste(mpi.get.processor.name(),"is",mpi.comm.rank(),"of",mpi.comm.size()))
print(Result)

# Tell all slaves to close down, and exit the program
mpi.close.Rslaves()
mpi.quit(save="no")
```

Example: Parallelizing Code

Calculating pi by numerical integration:

$$\int_0^1 \frac{1}{1+x^2} dx = \arctan(1) - \arctan(0) = \arctan(1) = \frac{\pi}{4}$$



$$\pi \approx \left(\sum_{n=1}^{\# \text{ intervals}} \frac{4}{(1+x^2)} \right) / \# \text{ intervals}$$

$$\text{where } x = \frac{\text{interval} - 0.5}{\# \text{ intervals}}$$

Start with the serial version:

`pi.R`

Galen Collier (galen@clemson.edu)

```
intervals <- as.integer(readline("Please enter the number of intervals: "))

computeInterval <- function(intervals) {
  ysum <- 0.0;
  for (i in 1:intervals) {
    xi <- (1.0/intervals)*(i+0.5)
    ysum <- ysum + 4.0/(1.0+xi*xi)
  }
  myarea <- ysum*(1.0/intervals)
  return(myarea)
}

Result <- computeInterval(intervals)
print(paste("Area is", Result))
```



```
if (!is.loaded("mpi_initialize")) {      #Added
  library("Rmpi")                       #Added
}                                        #Added

mpi.spawn.Rslaves()                    #Added
intervals <- as.integer(readline("Please enter the number of intervals: "))

computeInterval <- function(intervals) {
  rank <- mpi.comm.rank()               #Added
  size <- mpi.comm.size()               #Added
  size <- size - 1                      #Added WHY IS THIS NEEDED?
  ysum <- 0.0;
  for (i in seq(rank, intervals, by=size)) { #Changed WHY???
    xi <- (1.0/intervals)*(i+0.5)
    ysum <- ysum + 4.0/(1.0+xi*xi)
  }
  myarea <- ysum*(1.0/intervals)
  return(myarea)
}

mpi.bcast.Robj2slave(intervals)        #Added
mpi.bcast.Robj2slave(computeInterval) #Added
Result <- mpi.remote.exec(computeInterval(intervals)) #Changed




area <- apply(Result, 1, sum)           #Added
print(paste("Area is", area))          #Changed (slightly)

mpi.close.Rslaves()                   #Added
mpi.quit(save="no")                   #Added
```

Homework (cont.)

BONUS! You are asked to take the single threaded version of mineAminos and convert it to an Rmpi version.

 Hints:

-  Run a different 'Chromosome' on a different slave. (Compare 'i' to 'rank')
-  The results returned by `mpi.remote.exec` will be a 'list-of-lists' use `as.matrix(as.numeric(Results[i]))` to convert to matrix columns
-  Get started early!

 GOOD LUCK!

Final Thoughts

We're just getting started!

Hadoop!



Do you have a question(s)?