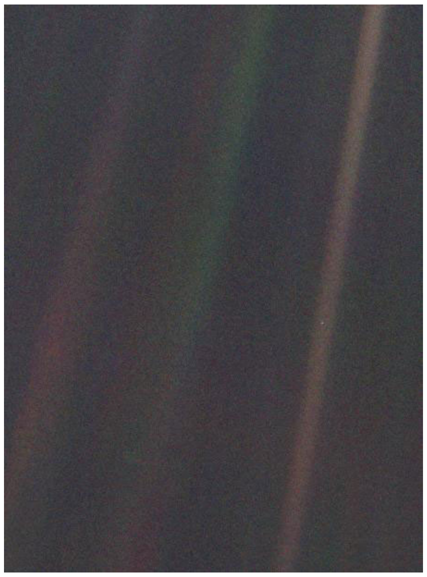
Spring 2012 BMTRY 789-02

Parallel Processing in R

Adrian Michael Nida

2012-04-03

1 Opening



I use this image to represent how high level of a view this lecture will be into the field of Parallel Processing. You are strongly encouraged to learn more about both Carl Sagan's Pale Blue Dot and Parallel Processing. In addition this image is good because it conveys how challenging it will be to effectively communicate these concepts. Even if I'm just slightly off in my projections, it can totally bypass "the mote of dust suspended in a sunbeam."

2 Overview

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*

This is a rough outline of the talk. Each Section will begin with an overview of what that section will cover. Quotes are provided for comic relief.

3 Introduction

Introduction

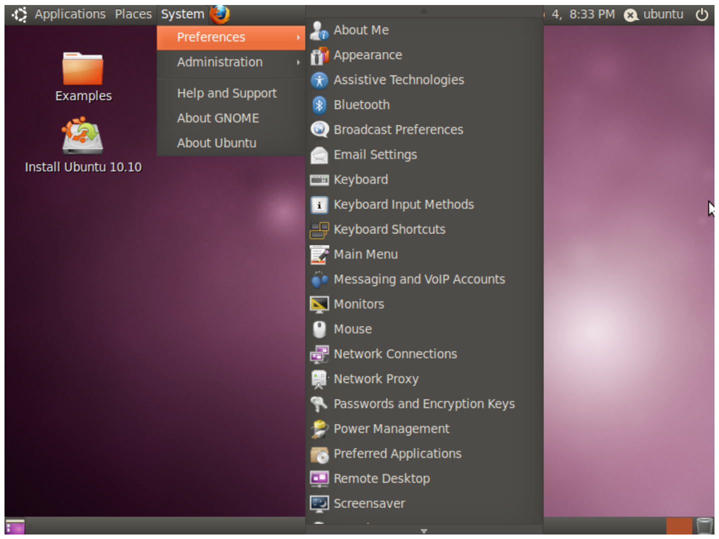
- UNIX != Windows
- History
- Executable Syntax
- Common Commands
- Editing Files
- Secure Shell (ssh)
- Source Control (optional)

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

– Ken Thompson

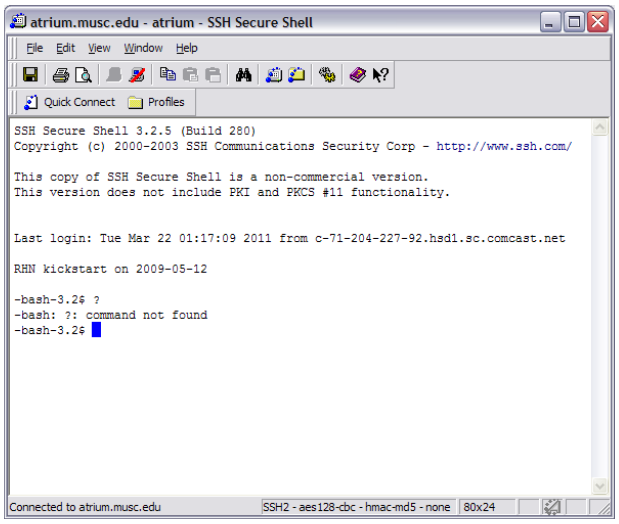
The cluster is built ontop of Linux, which is a clone of UNIX (more later). Therefore we must spend some time learning UNIX before we can talk about our Cluster and what you will need to know to use it.

UNIX != Windows



This is a screenshot of the latest released Ubuntu desktop, 10.10. This is to show you that even the windowing environment differs from the traditional Windows-based desktop.

UNIX != Windows (cont.)


atrium.musc.edu - atrium - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
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$
```

```
Connected to atrium.musc.edu    SSH2 - aes128-cbc - hmac-md5 - none    80x24
```

It takes computer cycles to draw all those pretty pictures you saw earlier. These computing cycles can better be used by other computational intensive tasks (e.g., httpd, yourDissertation.R, etc.). Many *ix based systems are installed without an X server to accommodate these tasks. Most of the way you interact with *ix will be through the Command Line Interpreter (CLI) which is shown above. In this world, folders are called directories, there is no ”C drive” because everything are files/directories under root (’/’), ”My Documents” is now ’ ’, there are other differences.

A History of UNIX

The history

UNICS (as it was called) began life as a way for Ken Thompson and Dennis Ritchie to play ”Space Travel” (a game similar to Star Trek) on a PDP7 back in 1969. It has fragmented and/or been reimplemented many times over since then. For example, the cluster we use is based on Rocks, which is based on CentOS, which is based on RedHat, which is based on Linux, which is based on minix. Minux was a fork of the BSD improved version of ”true UNIX” (a phrase which has spawned millions of dollars in lawsuits).

Executable Syntax

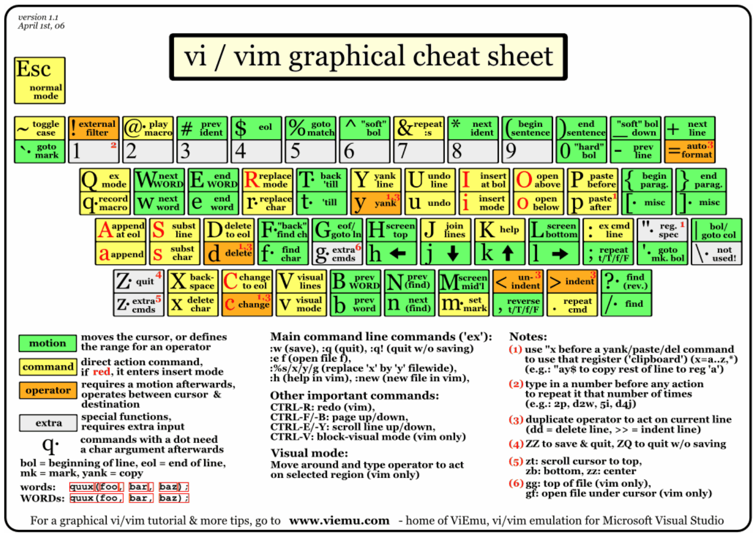
‘/path/to/program [options] [files]‘
where:

- program is the name of the program you wish to rum
  - /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.

This is an overview of how to interact with programs while using the CLI. The items in brackets are optional. These items vary from program to program and many of the options used in one program differ in other programs.

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

A list of the common UNIX commands and what they do. There are others, but I have tried to list the ones you will commonly use. You are encouraged to master these and others.



🟩 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(foo, bar, baz);
WORDs: quux(foo, 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:
(1) 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')
(2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
(3) duplicate operator to act on current line (dd = delete line, >> = indent line)
(4) ZZ to save & quit, ZQ to quit w/o saving
(5) zt: scroll cursor to top, zb: bottom, zz: center
(6) 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: VIemu

Everything in UNIX is a file, so being able to edit files efficiently is the "sink-or-swim" skill. Above is the Cheat Sheet for vi(sual), which is one side in the famous Editor War (the other side in Emacs, which you are also free to learn). Both will require a devotion of time to master, and this short term investment will yield a very productive long term gain. However, if you feel swamped in this, you are free to use pico (which is really nano).

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 (^{TM})

SSH allows you to login to another computer on the network. It replaces rsh and telnet which did everything in clear text. You will need this to connect to the cluster. There are ways to configure "keys" so you don't have to type a password each time you login to a machine.

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://tortoisesvn.tigris.org/ is a well received Windows client.

Source Control is necessary and far more efficient than mailing yourself copies of important_2-2012-04-03.doc. This workflow is simplistic, but it's enough to get you started.

4 Cluster

Cluster

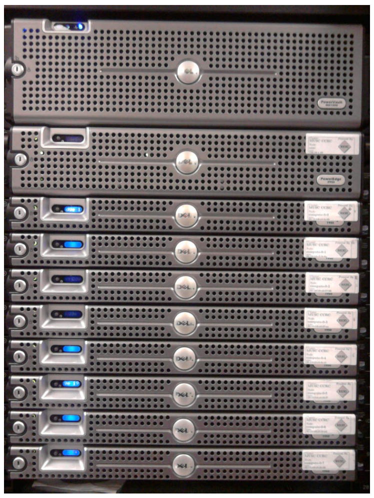
- Hardware capabilities
- User Accounts
- Environment

"Imagine a Beowulf cluster of these!"

– Anonymous (Coward) Slashdot Troll

This is the part of talk that will concentrate on what the cluster you will be using is and what you need to know to use it.

Hardware capabilities



The Cluster's Homepage

The computing resources within the cluster. Each node has 8 Intel(R) Xeon(R) E5345 @ 2.33GHz processing cores. Each node has 16GB of RAM. There are a total of 8 computing notes + one head node. There is also a 1.4TB storage space shared by all nodes.

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)

Not much to say here that isn't already on the slide.

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

Things were laid out originally by Matthew Shotwell and Adam Richards. There has been discussions about changing the way things look. These discussions have also covered upgrades. While there has been much discussions, there is little, if any, monies being spent towards this. That's why all there is at the moment is discussion.

5 Parallel Processing

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

Quad Core Processing explained
8 seconds
32 seconds

Author Unknown

This image shows the major advantage of using Parallel Processing, the time savings. Achieving this performance improvement isn't guaranteed as certain things can make one/more nodes require more time than others.

Problems

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

"With great power comes great responsibility" This stuff isn't easy, don't expect it to be. You will not get it right all the times. Don't get discouraged, feel free to ask for help, that's what friend(s) and consults with Dr. Google are for.

Two Types

- Batch Programming
- Truly Parallel

TIMTOWTDI
'Tim Toady'

Two Types

- Batch Programming
- Truly Parallel

TIMTOWTDIBSCINABTE
'Tim Toady Bicarbonate'
There are two types of Parallel Processing. What is referred to as "Batch Programming" is really just breaking up a large sequential task into many smaller tasks. True Parallel programs are written from the ground up to allow for processing in a parallel environment. 'There's more than one way to do it, but sometimes consistency is not a bad thing either'

5.1 Batch Programming

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
```

This is the command you run to tell R to run a specific R.file in batch mode. '–slave' is an option that will suppress the startup banner and other extraneous output. –args are sent to your script and MUST be included in quotes. If outfile is not specified, my_script.Rout will be used. The example shows how to obtain the arguments you passed in and how to end your R scripts in such a way to determine whether an error occurred.

Bash Scripting Commands

Command	Description
qsub [script.sh]	Submit batch jobs
qsub -I	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

These are the commands you will be using to submit jobs to the cluster through Sun's (now Oracle's) Grid Engine.

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]
```

NameOfYourJob is what will appear in the job queue. EmailAlias is different than your NetID. While NetID@musc.edu will get the mail to you, it is likely to break in the future and shouldn't be used. m specifies when you want to receive mail: beginning, end, abort, suspended. You can also specify no mail. V specifies that you want to pass all the environment variables. cwd means current working directory. What this is doing is calling my_script.R through the cluster. The paradigm here is to code your script in a way where you can divvy up sections of the problem by changing the arguments. Then, create one batch file per argument division. If creating one batch file per division is complicated (i.e., too many to type), feel free to create a program that will do it for you.

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)

This homework will expose you to two things, the batch processing system (Sun Grid Engine) and Rmpi (more on that later). You are to compare the results and time of the single threaded version to two things – the batch version (run with different number of 'slaves') and an Rmpi version. While there is a program to create the batch files that you will need to run through the Sun Grid Engine, you are responsible for creating a program that can 'reduce' the output of each in a way that makes it comparable to the single-threaded version.

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])
```

The single threaded version. ChromosomeLength is defined so we don't have to type the same value more than once. scan is used to read in the file in the correct manner (one chromosome of 3000 nucleotides per line). The total is computed once so we don't compute this non-changing value more than once. A list is created to store our results.

For each line in the file (which equals the number of chromosomes), we first grab the chromosome of interest. The we go through and break it into three character chunks using substr. We check to see if we have seen this amino acid before. If we have, we add one to its value. If not, the amino acid's value is set to one.

When the file has been processed, the Amino Acids are ordered. Then, we go through all of the amino acids we found and print out, the amino acid sequence and the number of times it appeared. paste and cat are used to work around needing both newline and tab characters in the same line (paste doesn't like backslash t as a separator and I didn't want the line to end in a separator). Finally the amount of time needed to process the file is printed.

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
```

An example of the output. The first line comes from scan() the rest is the output we have instructed our program to use. Note the amount of time listed. That is in seconds. How many minutes is it?

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])
```

The batch version. The items in red mark the changed lines between this and the single threaded version.

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 ...

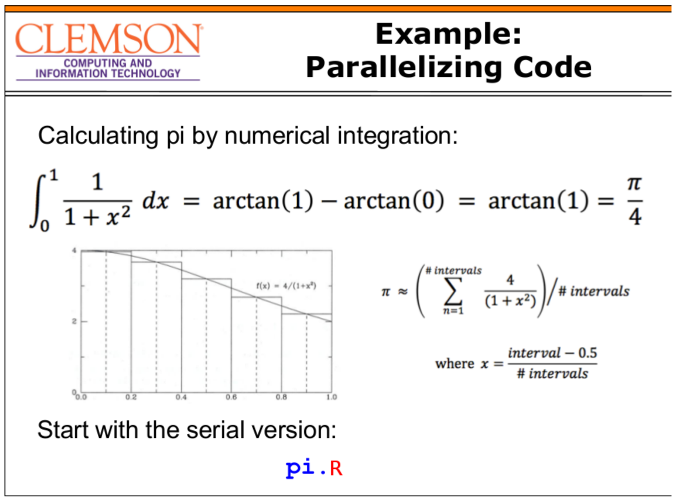
At this point, I will log into the cluster and show you the code examples in action.

5.2 True Parallel Processing

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")
```

The Rmpi library is housed http://cran.r-project.org/web/packages/Rmpi/index.html. However this modified example came from http://math.acadiau.ca/ACMMaC/Rmpi/index.html. This is a very simplified example, but it will be stepped through so you know what each section does.


CLEMSON
COMPUTING AND INFORMATION TECHNOLOGY
f(x) = 4/(1+x²)

Example: Parallelizing Code

Calculating pi by numerical integration:

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

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

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

Start with the serial version:

pi.R

Galen Collier (galen@clemson.edu)

My thanks for Galen for this idea. We will show how to evaluate pi (3.1415926) using this library. This slide, in addition to allowing me to give credit, illustrates the general idea. Thanks to the wonderful world of trigonometry, we know pi is linked to arctangent because the arctangent of one is pi / 4. We also know the derivative of arctan is $\int_0^1 \frac{1}{1+x^2}dx$. This and the trapezoidal rule allows us to estimate pi by the formula seen here. We'll quickly review the single threaded code:

```
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))
```

Simple enough, right? readline takes input from the user. as.integer does what you expect. We have a function that does all the math and returns back what was computed. Then we call our function and print out the results. Let's see what we need to change in order to make this an Rmpi version...

```
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
```

This example estimates PI depending on how the number of intervals are selected. Rmpi makes sure they the question and printouts only happen on the node with rank 0, why? mpi.bcast.Robj2slave broadcasts that variable to all the slave nodes in the cluster. mpi.remote.exec tells Rmpi to run that function on all the slaves. The result comes back as either a matrix/list depending on the result type. apply has to be used to reduce the Nodes will BLOCK at these steps. Also note how the for loop is written. Make sure you understand it.

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!

Feel free to ask me questions. Email (nida@musc.edu) would be best.

Final Thoughts

We're just getting started!

Hadoop!

As mentioned before, this is just the beginning. Hadoop is something we used over at OBIS and uses off-the-shelf commodity hardware to create clusters. It is backed by Google and others. It also uses many of the map/reduce tactics.

6 Questions

Do you have a question(s)?