Statistical and Mathematical Software on HPC systems

Jefferson Davis
Research Analytics
Research Analytics

- Formerly the Stat/Math Center
- A subgroup of Research Technology
- Support and consulting for software in statistical, mathematical, and geographic analysis.
“Research Analytics-y” questions

• “Can I import my data into Stata?”
• “How do I run a Kolmogorov–Smirnov test in Matlab?”
• “My optimization takes forever to run. Why?”
• “How can I export an ArcGIS attribute table to Excel?”
Plan of Attack

• Look at three packages on Big Red II: SAS, R, Matlab.
• Look at running a common task in all three.
• Discuss a bit on how the different packages take advantage of multicore or parallel functionality.
What modules am I using?

module load ccm
module load matlab
module load r
module load sas
module load cudatoolkit

With the current naming conventions you don’t need to specify the version number. The most recent version is the default.
Three roads to parallelism

• Implicit parallelism (my favorite)
• Small modifications to existing code
• Fiddling around with mpi
SAS: background

• First developed in 1966 at North Carolina State for regression and analysis of variance.
• Commercialized in 1976 when the SAS Institute incorporates.
• Solid workhorse statistical package
• Still has not been used as a New York Times crossword clue, but both “Special Air Service” and “Scandinavian Airlines System” have.
SAS: starting it up

majdavis@login1:~> module load sas
SAS data analysis and management system version 9.4 loaded.

~> sas –nodms
More common to write the sas program and run it as a script

~> sas lineExample.sas
A SAS program has two steps: a data step and a proc step.
SAS: loading data and regressing

Look at the example of fitting a line through some data.

The csv file test2.csv has twenty values.

We use SAS to find the line of best fit.
SAS: loading data and regressing

The file lineExample.sas

data test2;
  infile "test2.csv" delimiter="",";"
  input x y;
run;
proc reg;
  model y = x;
run;
~>sas lineExample.sas

The output goes to lineExample.lst
SAS: loading data and regressing

```bash
~> tail -n 7 lineExample.lst
```

<table>
<thead>
<tr>
<th>Parameter Estimates</th>
<th>Parameter</th>
<th>Standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>DF</td>
<td>Estimate</td>
</tr>
<tr>
<td>Intercept</td>
<td>1</td>
<td>3.54368</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>2.19113</td>
</tr>
</tbody>
</table>

Sure, $y = 2.19113 \times x + 3.54368$ seems okay.
Producing graphics requires an X11 connection. Running canada.sas will save create an HTML file and a graphic in gchart.png

```sas
ods listing close;
ods html style=money file="CanadaGraph.html";
proc gchart data=sashelp.prdsale;
vbar Product / sumvar=actual;
ttitle1 "First Quarter Sales in Canada";
where Quarter=1 and  
  Country="CANADA";
run;
quit;
ods html close;
ods listing;

~>sas canada.sas
```
• SAS will create threads to run faster on multicore environments.
• This happens by default but we can see the improvement if we force SAS to run without threads.
• The next example involves taking the mean of a set of 50,000,000 random numbers.
### SAS: implicit parallelism example

<table>
<thead>
<tr>
<th>%let NObs = 50000000;</th>
<th>%let NObs = 50000000;</th>
</tr>
</thead>
<tbody>
<tr>
<td>data Unif(keep=u);</td>
<td>data Unif(keep=u);</td>
</tr>
<tr>
<td>call streaminit(123);</td>
<td>call streaminit(123);</td>
</tr>
<tr>
<td>do i = 1 to &amp;NObs;</td>
<td>do i = 1 to &amp;NObs;</td>
</tr>
<tr>
<td>u = rand(&quot;Uniform&quot;); /* U[0,1] */ output;</td>
<td>u = rand(&quot;Uniform&quot;); /* U[0,1] */ output;</td>
</tr>
<tr>
<td>end;</td>
<td>end;</td>
</tr>
<tr>
<td>run;</td>
<td>run;</td>
</tr>
</tbody>
</table>

```
proc means data=unif;
var u;
run;
```

```
proc means data=unif;
var u;
options nothreads;
run;
```

---

**NOTE:** PROCEDURE MEANS used (Total process time):

<table>
<thead>
<tr>
<th>Time Type</th>
<th>Time Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>real time</td>
<td>1.76 seconds</td>
</tr>
<tr>
<td>cpu time</td>
<td>4.37 seconds</td>
</tr>
</tbody>
</table>

---

**NOTE:** PROCEDURE MEANS used (Total process time):

<table>
<thead>
<tr>
<th>Time Type</th>
<th>Time Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>real time</td>
<td>3.63 seconds</td>
</tr>
<tr>
<td>cpu time</td>
<td>3.57 seconds</td>
</tr>
</tbody>
</table>
R: background

- First created in the early 1990s by Ross Ihaka and Robert Gentleman as an implementation of S.
- Development soon shifted to a larger core group.
- Distributed under the GNU General Public License.
R: starting it up

```bash
~> module load R
~> R
```
R: some useful tidbits

- The question mark will display a function’s help text. This is a shortcut for the help() function
  ```r
  help(sin)
  ? sin
  ```
- The command `invisible()` suppresses output
- The up arrow key will go back to previous commands
- The command `system()` is used for shell commands
  ```r
  system("rm core")
  ```
- The `rm()` command clears variables
  ```r
  rm(list=ls(all=TRUE))  #Clear all variables
  ```
- The hash tag `#` is used for comments
  ```r
  #This is an R comment
  ```
- Goggling for `[R]` will save a lot of frustration (This one might not be true anymore)
Let’s load the file test2.csv again and run the same regression we ran in SAS.

```r
> test2 <- read.csv("test2.csv", header = FALSE)
> test2
   1 1 5.5216
   2 2 6.4930
> lm(test2$V2 ~ test2$V1)

Call:
  lm(formula = test2$V2 ~ test2$V1)
Coefficients:
(Intercept)  test2$V1
  3.544      2.191
```
R: plotting data

You don’t need an X11 connection to generate graphics

```r
fit<-lm(test2$V2~test2$V1)
png()
plot(test2$V1,test2$V2)
abline(fit$coefficients)
dev.off()
```

Default name is “Rplot001.png”.
R: multicore versions of apply()

R has many mapping functions that apply a function to the elements of a list, vector, what-have-you.
The function lapply() applies a function to the elements of a list. The function mcapply() is a multicore version of lapply.
The function rep(m,n) repeats m for n time. So lapply(rep(100,1000000),rnorm) returns 1000000 lists of 100 random numbers from a normal distribution.

```
st<-system.time(invisible(lapply(rep(100,1000000),rnorm))))
st[3]

library(parallel)
stm<-system.time(invisible(mclapply(rep(100,1000000),rnorm,mc.cores=32)))
stm[3]
```

<table>
<thead>
<tr>
<th>elapsed</th>
<th>elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.634</td>
<td>4.066</td>
</tr>
</tbody>
</table>
Matlab: background

• Developed by Cleve Moler in the 1970s to give students easier access to numerical libraries for linear algebra (Matrix Laboratory)
• MathWorks company founded in 1984 for commercial development
• About 1900 IU network users 2013-14 academic year
• Somewhat decent support for parallelism
Matlab: starting it up

majdavis@login1:~> module load matlab
SAS data analysis and management system version 9.4 loaded.

~> matlab
Matlab: some useful tidbits

- The help command will display a function’s help text. The doc command brings up more information
  
  help sin
  
  doc sin

- The semi-colon (;) will suppress output

- The up arrow key will go back to previous commands

- Typing and then using the up arrow key goes back to previous commands that start with that text

- The exclamation point is used for shell commands
  
  ! rm matlab_crash_dump.*

- The percent sign is used for comments
  
  %This is a Matlab comment
Matlab: loading data and regressing

```matlab
>> test2 = csvread('test2.csv');
>> x = test2(:,1);
>> y = test2(:,2);
>> coeffs = [x ones(size(x))]
y;
coeffs =
    2.1911
    3.5437
>> plot(x,y,'rx')
>> hold on
>> plot(coeffs(1)*x+coeffs(2),'k')
>> print(gcf, '-dpng', 'MatlabPlot')
```
Matlab: implicit parallelism in svd

Many functions will recognize that the multicore environment and create an appropriate number of threads.

Look at an example of two singular value decompositions.
(Rewriting a matrix as the product of “nice” matrices.)
Matlab: implicit parallelism in svd

```
> matlab

 tic
 svd(rand(5000))
 toc

Elapsed time is 67.720321 seconds.
```

```
> matlab -singleCompThread

 tic
 svd(rand(5000))
 toc

Elapsed time is 157.032804 seconds.
```

Older versions of Matlab allowed you to set the maximum number of threads with `maxNumCompThreads`, but this is being deprecated.
Matlab: svd using the gpu

<table>
<thead>
<tr>
<th>&gt; qsub –l -q debug_gpu -lgres=ccm</th>
<th>&gt; qsub –l -q debug_gpu -lgres=ccm</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; ccmlogin</td>
<td>&gt; ccmlogin</td>
</tr>
<tr>
<td>&gt; matlab</td>
<td>&gt; matlab</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tic</th>
<th>tic</th>
</tr>
</thead>
<tbody>
<tr>
<td>svd(rand(5000))</td>
<td>svd(rand(5000,'gpuArray'));</td>
</tr>
<tr>
<td>toc</td>
<td>toc</td>
</tr>
</tbody>
</table>

Elapsed time is 62.310409 seconds.  Elapsed time is 16.616624 seconds.
Matlab: parallel-enabled functions

Many Matlab functions can use a pool of worker processes if you explicitly create them and tell the function to use them.

Sample problem: maximize the function $x_1^2 + 4\sin(5x_2)$ subject to the constraint $(x_1-1)^2 + (x_2-1)^2 = 25$

We first write a function to define the constraint mycon.m.

```
function [c,ceq] = mycon(x)
    c = (x(1)-1)^2 + (x(2)-1)^2 - 25;
    ceq = [];
```
Matlab: parallel-enabled functions

Then we set up the problem
```matlab
opts = optimset('Algorithm','sqp');
problem = createOptimProblem('fmincon','objective', ... @(x) x(1)^2 + 4*sin(5*x(2)),'x0',[3 3],'
lb',[-5 -5], ... 'ub',[5 5],'nonlcon',@mycon,'options',opts);
ms = MultiStart;
```

The Matlab Multistart solver runs an optimizer from multiple start points. It’s natural to want to run it in parallel.

|_ms.UseParallel = false; tic [x,f] = run(ms,problem,2000); toc Elapsed time is 48.502433 seconds. |
|---|---|---|
|ms.UseParallel = true; tic [x,f] = run(ms,problem,2000); toc Elapsed time is 4.673513 seconds. |
Matlab: parallel-enabled functions

Matlab will more more than happy to let you run things in parallel even is it’s a really bad idea.

The optimizer patternsearch is an example. At each step patternsearch checks the values of the objective function at near the current point. The first point with a lower value becomes the current point for the next step.

You can, however, use a pool of workers and check them in parallel. This means checking all the nearby points.
**Matlab: parallel-enabled functions**

The code on the right is checking a full 1024 points at each step. Running it in parallel doesn’t make up for that.

<table>
<thead>
<tr>
<th>Code 1</th>
<th>Code 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x0 = .4*ones(10,1); tic patternsearch(@(x)myFun(x,params), ... x0,[],[],[],[],0*x0,1+0*x0,[]); toc</code></td>
<td><code>parpool(32) options = psoptimset('UseParallel', true,... 'CompletePoll', 'on', 'Vectorized', 'off'); x0 = .4*ones(10,1); tic patternsearch(@(x)myFun(x,params), ... x0,[],[],[],[],0*x0,1+0*x0,[],options); toc</code></td>
</tr>
<tr>
<td>Elapsed time is 0.476223 seconds.</td>
<td>Elapsed time is 4.673513 seconds.</td>
</tr>
</tbody>
</table>
Matlab: parallel for loops

If you have a pool of parallel workers you use them to run a for-loop with parfor. The are some restrictions on the loop, but the main one is that the order of evaluation can’t matter.

So the code below fails

```
parpool(2)
parfor i=3:100
a(i)=a(i-1)+a(i-2);
end
delete(gcp)
```
Matlab: parallel for loops

An example that doesn’t fail

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tic</code></td>
<td><code>tic</code></td>
</tr>
<tr>
<td><code>for i=1:50000</code></td>
<td><code>for i=1:50000</code></td>
</tr>
<tr>
<td><code>    max(rand(100));</code></td>
<td><code>    max(rand(100));</code></td>
</tr>
<tr>
<td><code>end</code></td>
<td><code>end</code></td>
</tr>
<tr>
<td><code>toc</code></td>
<td><code>toc</code></td>
</tr>
<tr>
<td></td>
<td><code>delete(gcp)</code></td>
</tr>
<tr>
<td>Elapsed time is 11.202063 seconds.</td>
<td>Elapsed time is 2.629504 seconds.</td>
</tr>
</tbody>
</table>
Thanks for coming
Contact info

majdavis@iu.edu
statmath@rtinfo.indiana.edu