Statistical and Mathematical Software on HPC systems

Jefferson Davis
Research Analytics
Plan of Attack

• Look at three packages on Karst: 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 matlab
module load r
module load sas

For all these most recent version of the package 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

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

The csv file faithful.csv has 272 values.

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

The file lineExample.sas

data faithful;
  infile "faithful.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
```

Parameter Estimates

| Variable | DF | Parameter Estimate | Standard Error | t Value | Pr > |t| |
|----------|----|--------------------|----------------|---------|------|------|
| Intercept| 1  | 33.47440           | 1.15487        | 28.99   | <.0001|
| x        | 1  | 10.72964           | 0.31475        | 34.09   | <.0001|

Sure, $y = 10.72964 \times x + 33.47440$ seems okay.
SAS: producing graphics

ods listing close;
ods html style=money
file="CanadaGraph.html" ;
proc gchart
  data=sashelp.prdsale;
vbar Product / sumvar=actual;
title1 "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

```sas
%let NObs = 500000000;
data Unif(keep=u);
call streaminit(123);
do i = 1 to &NObs;
   u = rand("Uniform"); /* U[0,1] */
   output;
end;
run;

proc means data=unif;
var u ;
run;

NOTE: PROCEDURE MEANS used
(Total process time):
   real time  13.45 seconds
   cpu time   28.80 seconds
```

```sas
%let NObs = 500000000;
data Unif(keep=u);
call streaminit(123);
do i = 1 to &NObs;
   u = rand("Uniform"); /* U[0,1] */
   output;
end;
run;

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

NOTE: PROCEDURE MEANS used
(Total process time):
   real time  24.91 seconds
   cpu time   23.91 seconds
```
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

~> module load r
~> R

One Karst node is running an Rstudio server

https://rstudio.iu.edu/
R: some useful tidbits

• The question mark will display a function’s help text. This is a shortcut for the help() function
  ? sin

• The command \texttt{R CMD BATCH --no-save R_input.R} runs in batch

• The up arrow key will go back to previous commands

• The command \texttt{system()} is used for shell commands
  \texttt{system("rm core")}

• The \texttt{rm()} command clears variables
  \texttt{rm(list=ls(all=TRUE))} \ #Clear all variables

• The hash tag is used for comments
  \#This is an R comment
Let’s load the file `<faithful.csv>` again and rerun the earlier regression.

```r
faith <- read.csv("faithful.csv", header=FALSE)
f
```

```
  V1  V2
V1 3.600 79
V2 1.800 54
```

```r
lm(faith$V2 ~ faith$V1)
```

```
Call:
lm(formula = faith$V2 ~ faith$V1)

Coefficients:
(Intercept)    test2$V1
33.47         10.73
```
R: plotting data

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

Default name is “Rplot001.png”.

Multiple plots are saved in multiple files.
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 `mclapply()` 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(     
|   lapply(rep(100,1000000), rnorm))   
| st[3]                 | library(parallel) 
|                       | stm <- system.time(     
|                       |   mclapply(rep(100,1000000), rnorm, mc.cores = 15)) 
|                      | stm[3]               |
| elapsed               | elapsed          |
| 18.733                | 2.927             |
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

• Decent support for parallelism
Matlab: starting it up

~> module load matlab
MATLAB numerical calculation framework version 2015a loaded.
~> matlab

To run myInput.m in batch
~> matlab -r myInput
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

faith = csvread('faithful.csv');
x = faith(:,1);
y = faith(:,2);
fit = fitlm(x, y, 'linear')
fit =
Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:

<table>
<thead>
<tr>
<th></th>
<th>Estimate</th>
<th>SE</th>
<th>tStat</th>
<th>pValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>33.474</td>
<td>1.1549</td>
<td>28.985</td>
<td>7.136e-85</td>
</tr>
<tr>
<td>x1</td>
<td>10.73</td>
<td>0.31475</td>
<td>34.089</td>
<td>8.13e-100</td>
</tr>
</tbody>
</table>
Matlab: plotting data

```
fit.plot
print( gcf,'-dpng',...
     'MatlabPlot' )
```

This saves the plot as MatlabPlot.png.
Matlab: implicit parallelism in svd

Many functions will recognize the multicore environment and create an appropriate number of threads. A good example is singular value decomposition (SVD), rewriting a matrix as the product of “nice” matrices.

<table>
<thead>
<tr>
<th>&gt; matlab</th>
<th>&gt; matlab -singleCompThread</th>
</tr>
</thead>
<tbody>
<tr>
<td>tic</td>
<td>tic</td>
</tr>
<tr>
<td>svd(rand(5000))</td>
<td>svd(rand(5000))</td>
</tr>
<tr>
<td>toc</td>
<td>toc</td>
</tr>
</tbody>
</table>

Elapsed time is 15.343513 seconds.  
Elapsed time is 157.930566 seconds.
Matlab: svd using the gpu

<table>
<thead>
<tr>
<th>Code</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; qsub -I -q debug_gpu -lgres=ccm &gt; ccmlogin &gt; matlab</td>
<td></td>
</tr>
<tr>
<td>tic</td>
<td></td>
</tr>
<tr>
<td>svd(rand(5000))</td>
<td></td>
</tr>
<tr>
<td>toc</td>
<td></td>
</tr>
<tr>
<td>Elapsed time is 62.310409 seconds.</td>
<td></td>
</tr>
<tr>
<td>&gt; qsub -I -q debug_gpu -lgres=ccm &gt; ccmlogin &gt; matlab</td>
<td></td>
</tr>
<tr>
<td>tic</td>
<td></td>
</tr>
<tr>
<td>svd(rand(5000,'gpuArray'));</td>
<td></td>
</tr>
<tr>
<td>toc</td>
<td></td>
</tr>
<tr>
<td>Elapsed time is 16.616624 seconds.</td>
<td></td>
</tr>
</tbody>
</table>
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 \( x_1^2 + 4 \sin(5 x_2) \) subject to the constraint 
\[ (x_1 - 1)^2 + (x_2 - 1)^2 = 25 \]

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

```matlab
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.

<table>
<thead>
<tr>
<th>ms.UseParallel = false; tic [x,f] = run(ms,problem, 2000); toc</th>
<th>ms.UseParallel = true; tic [x,f] = run(ms,problem, 2000); toc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed time is 52.175676 seconds.</td>
<td>Elapsed time is 5.533175 seconds.</td>
</tr>
</tbody>
</table>
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

On the right Matlab tries to check all $2^{100}$ nearby points at each step. Yikes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>matlab</code> x0=ones(100,1); tic patternsearch(@(x)myFun(x),... x0,[],[],[],[],0<em>x0,1+0</em>x0,[]); toc</td>
<td>Elapsed time is 29.419132 seconds.</td>
</tr>
<tr>
<td><code>matlab</code> parpool(15) options = psoptimset('UseParallel',true,'CompletePoll', 'on', 'Vectorized', 'off'); x0=ones(100,1); tic patternsearch(@(x)myFun(x),... x0,[],[],[],[],0<em>x0,1+0</em>x0,[],options); toc</td>
<td>Elapsed time is 160.692717 seconds.</td>
</tr>
</tbody>
</table>

delete(gcp)
Matlab: parallel for loops

If you have a pool of parallel workers you use them to run a for-loop with parfor.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tic</td>
<td></td>
</tr>
<tr>
<td>for i=1:5000000</td>
<td></td>
</tr>
<tr>
<td>a(i)=max(rand(100,1));</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>toc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Elapsed time is 21.712727 seconds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tic</td>
<td></td>
</tr>
<tr>
<td>parpool(15)</td>
<td></td>
</tr>
<tr>
<td>tic</td>
<td></td>
</tr>
<tr>
<td>parfor i=1:5000000</td>
<td></td>
</tr>
<tr>
<td>a(i)=max(rand(100,1));</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>toc</td>
<td>delete(gcp)</td>
</tr>
<tr>
<td></td>
<td>Elapsed time is 2.548504 seconds.</td>
</tr>
</tbody>
</table>
Matlab: parallel for loops.

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

```
%Fibonacci failure
a(1)=1;a(2)=1;
parpool(2)
parfor i=3:100
    a(i)=a(i-1)+a(i-2);
end
delete(gcp)
```
Contact info

majdavis@iu.edu
researchanalytics@iu.edu