Parallel MATLAB jobs on Biowulf

Dave Godlove, HPC @ NIH
david.godlove@nih.gov
February 17, 2016

While waiting for the class to begin, log onto Helix and execute the following command

\$ cp -r /data/classes/matlab/swarm_example /data/$USER
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)
  $ cp -r /data/classes/matlab/swarm_example /data/$USER
Quick review of a few MATLAB tricks

```
>> name='Mary'; adjective='little'; noun='lamb'

>> fprintf('%s had a %s %s. \n',name,adjective,noun)

Mary had a little lamb.
>>
```
Quick review of a few MATLAB tricks

```matlab
>> name='Mary'; adjective='little'; noun='lamb'

>> sentence = ...
    sprintf('%s had a %s %s. \n',name,adjective,noun);

>> sentence

sentence =

Mary had a little lamb.
```
Quick review of a few MATLAB tricks

eval

>> part1='6'; part2='7';

>> eval(['ultimate_answer = ' part1 ' * ' part2])

ultimate_answer =

        42
Quick review of a few MATLAB tricks

```matlab
>> part1='6'; part2='7';

>> eval(sprintf('ultimate_answer = %s * %s',part1,part2))

ultimate_answer =

    42
```
Quick review of a few MATLAB tricks

evalc

>> part1='6'; part2='7';

>> my_string = ...
evalc(sprintf('ultimate_answer = %s * %s',part1,part2));

>> my_string

my_string =

ultimate_answer =

    42
Quick review of a few MATLAB tricks

! (pronounced bang, similar to system)

>> whoami
Undefined function or variable 'whoami'.

>> !whoami
godlovedc

>> user = evalc('!whoami'), host = evalc('!hostname')

user =
godlovedc

host =
       cn1653
Quick review of a few MATLAB tricks

fprintf

sprintf

eval

evalc

!
Outline

- Steps to running parallel MATLAB jobs
  - Developing code interactively
  - Compiling code
  - Writing swarm files and calling swarm
  - Monitoring jobs

- A concrete example (processing image files)

```bash
$ cp -r /data/classes/matlab/swarm_example /data/$USER
```
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)
  
```bash
$ cp -r /data/classes/matlab/swarm_example /data/$USER
```
Quick overview of the cluster

Biowulf login node
Quick overview of the cluster

Biowulf login node

2 CPUs
(1 hyperthreaded core)

4 GB memory
(2 GB per CPU)
Quick overview of the cluster

Biowulf login node

Biowulf compute node
Quick overview of the cluster

Biowulf login node

Biowulf partition
Quick overview of the cluster

Biowulf login node

Biowulf cluster
Quick overview of the cluster

Biowulf cluster

Storage (mounted on all nodes)
Developing code interactively

The steps to starting an interactive MATLAB session on a Biowulf compute node

$ ssh -Y user@biowulf.nih.gov

$ sinteractive -c 4 --mem=4g -L matlab,matlab-image,matlab-stat,matlab-compiler

$ module load matlab/<ver>

Two options, GUI or command prompt

$ matlab&

Or

$ matlab -nojvm
Developing code interactively

Using ssh to start a secure shell session on biowulf

$ ssh -Y user@biowulf.nih.gov

This step my differ depending on the client
-putty
-X-Win32
-XQuartz
-NoMachine (preferred client / available for Windows, Mac and Linux, )

https://hpc.nih.gov/docs/connect.html
Developing code interactively

Using sinteractive to allocate resources

$ sinteractive -c 4 --mem=4g -L matlab,matlab-image,matlab-stat,matlab-compiler
Developing code interactively

Loading modules

$ module load matlab

Or

$ module load matlab/2015b

$ module load matlab/2014b

etc...
Developing code interactively

GUI vs. command line MATLAB

$ matlab&

Or

$ matlab -nojvm
Steps to running parallel MATLAB jobs

- Developing code interactively
- Compiling code
- Writing swarm files and calling swarm
- Monitoring jobs

A concrete example (processing image files)

```bash
$ cp -r /data/classes/matlab/swarm_example /data/$USER
```
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)
  $ cp -r /data/classes/matlab/swarm_example /data/$USER
Compiling code

Overview

• Removes the need for MATLAB licenses

• Allows you to run hundreds or thousands of instances of your code in parallel
Compiling code

Overview

Using a command like the following:

$ mcc2 -m my_function.m

or

>> mcc2 -m my_function.m

or

>> mcc2('-m','my_function.m')
Compiling code

Overview

produces several new files
(from my_function.m)

my_function ← binary
run_my_function.sh ← shell wrapper used to invoke my_function
readme.txt ← some useful info (version of MATLAB)
mccExcludedFiles.log ← lists some files that cannot be compiled
requiredMCRProducts.txt ← not human readable?
Compiling code

Overview

Execute compiled code using the shell script:

$ run_my_function.sh /usr/local/matlab-compiler/v90 input1 input2 inputN
Compiling code

Before you begin

Consider suppressing non-diagnostic output and figures

Will not be seen by user during execution and may complicate output (.o) files.
Compiling code

Before you begin

Can use the isdeployed flag:

```matlab
% isdeployed evaluates to “true” if code is
% compiled here we say if not deployed plot a
% figure and pause
if ~isdeployed
    figure
    plot(x,y)
    pause
end
```
Compiling code

Before you begin

Every function must be on your path at compile time.
Compiling code

Before you begin

Every function must be on your path \textbf{at compile time}!

\textbf{No} \texttt{addpath('new/path')} statements!

\textbf{Avoid} \texttt{cd('new/path')} statements too!
Compiling code

Before you begin

Compiled code will only accept strings as input

$ run_my_function.sh /usr/local/matlab-compiler/v90 input1 input2 inputN

For example:

$ run_my_function.sh /usr/local/matlab-compiler/v90 3 [1 2 5] {1,2,5}

Will be interpreted as:

>> my_function('3','[1 2 5]','{1,2,5}')
Compiling code

Before you begin

Code must convert strings back to their intended data types

input1 = char2double(input1);

eval(sprintf('input1 = %s;',input1))
Compiling code

Before you begin

Code must convert strings back to their intended data types

% this will check to see if the variable (var) is string and will convert it to strings value
if ischar(input1)
    eval(sprintf('input1 = %s;',input1))
end
Compiling code

When ready to compile, `mcc` is the proper command.

```bash
>> mcc -m my_function.m
```

But don’t use it!

It will tie the compiler license up for as long as your MATLAB session is active!
Compiling code

Instead, use `mcc2` (wrapper to `mcc`)

```matlab
>> mcc2 -m my_function.m
```

Opens a new instance of MATLAB, calls `mcc` to compile code, then closes new instance of MATLAB to release compiler license.

Available from MATLAB interactive sessions or from the shell (after loading MATLAB module).
Compiling code

Use the following command if `mcc2` does not appear on your search path.

```bash
>> rehash toolbox
```
Compiling code

Runtime flags

- singleCompThread
- nodisplay
- nojvm

For example:

```matlab
>> mcc2  -m  -R  -nodisplay  -R  -singleCompThread  my_function.m
```
Compiling code

Using the correct component runtime

$ run_my_function.sh /usr/local/matlab-compiler/v90 input1 input2 inputN

<table>
<thead>
<tr>
<th>MATLAB version</th>
<th>runtime library</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015b</td>
<td>v90</td>
</tr>
<tr>
<td>2015a</td>
<td>v85</td>
</tr>
<tr>
<td>2014b</td>
<td>v84</td>
</tr>
<tr>
<td>2013a</td>
<td>v81</td>
</tr>
<tr>
<td>2012b</td>
<td>v80</td>
</tr>
</tbody>
</table>

More info at:
https://hpc.nih.gov/apps/Matlab_compiler.html
Compiling code

Using a local mcr cache

```
$ ls -al | grep mcr
drwxr-x---  10 godlovedc godlovedc  4096 Nov 10 11:04 .mcrCache8.0
drwxr-xr-x   6 godlovedc godlovedc  32768 Jan 27 10:51 .mcrCache8.1
drwxr-x---   5 godlovedc godlovedc  73728 Jan 27 11:25 .mcrCache8.4
drwxr-x---  13 godlovedc godlovedc  8192 Oct 28 12:06 .mcrCache8.5
drwxr-x---  14 godlovedc godlovedc 24576 Feb 16 01:28 .mcrCache9.0
```
Compiling code

Using a local mcr cache

```bash
$ export MCR_CACHE_ROOT=/tmp/$USER/mcr_cache

or

>> user = deblank(evalc('!whoami'));
>> setenv('MCR_CACHE_ROOT',fullfile('/tmp',user,'mcr_cache'))
```
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)

```bash
$ cp -r /data/classes/matlab/swarm_example /data/$USER
```
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)

```bash
$ cp -r /data/classes/matlab/swarm_example /data/$USER
```
Writing swarm files and calling swarm

• Swarm is a wrapper for sbatch

• Creates 1 job per line (in a job array)

• Greatly simplifies job submission

• Tons of options

• Can capture sbatch commands if useful
Writing swarm files and calling swarm

Overview: two step process

• Write a swarm file that has a single command (job) on each line

• Invoke the swarm program using the name of the swarm file as an argument
Writing swarm files and calling swarm

Example swarm file:
myjobs.swarm

```
run_my_function.sh /usr/local/matlab-compiler/v90 param1
run_my_function.sh /usr/local/matlab-compiler/v90 param2
run_my_function.sh /usr/local/matlab-compiler/v90 param3
run_my_function.sh /usr/local/matlab-compiler/v90 paramN

$ swarm -f myjobs.swarm
```
% stick all the parameters in an array
param_list = {'param1' 'param2' 'param3' 'paramN'};

% make a command on a new line for each parameter
command_list = [];
for ii = 1:length(param_list)
    command_list = [command_list ...
                    'run_my_function.sh '...
                    '/usr/local/matlab-compiler/v90 '...
                    param_list{ii}...
                    '/n'];
end

% write the commands into a swarm file
file_handle = fopen('myjobs.swarm','w+');
fprintf(file_handle,command_list);
fclose(file_handle);
Writing swarm files and calling swarm

Example swarm file:
myjobs.swarm

```
run_my_function.sh /usr/local/matlab-compiler/v90 param1
run_my_function.sh /usr/local/matlab-compiler/v90 param2
run_my_function.sh /usr/local/matlab-compiler/v90 param3
run_my_function.sh /usr/local/matlab-compiler/v90 paramN
```
Writing swarm files and calling swarm

Remember:
Inputs are strings

run_my_function.sh /usr/local/matlab-compiler/v90 1
run_my_function.sh /usr/local/matlab-compiler/v90 2
run_my_function.sh /usr/local/matlab-compiler/v90 3
run_my_function.sh /usr/local/matlab-compiler/v90 4
Writing swarm files and calling swarm 

Arrays as input to compiled functions

```matlab
param1 =

    4   1   0   2
    4   4   0   5
    2   0   4   0
    3   1   3   2
```

```matlab
g >> param1 = [''' mat2str(param1) ''']
```

```matlab
param1 =

"[4 1 0 2;4 4 0 5;2 0 4 0;3 1 3 2]"
```
Writing swarm files and calling swarm

Arrays as input to compiled functions

run_my_function.sh /usr/local/matlab-compiler/v90 "[4 1 0 2;4 4 0 5;2 0 4 0;3 1 3 2]"
run_my_function.sh /usr/local/matlab-compiler/v90 "[2 2 4 1;4 2 1 1;4 3 3 2;1 4 3 5]"
run_my_function.sh /usr/local/matlab-compiler/v90 "[2 1 5 1;3 3 3 4;1 3 1 1;4 4 1 4]"
...
run_my_function.sh /usr/local/matlab-compiler/v90 "[1 1 4 1;5 3 3 4;2 2 3 4;1 2 5 2]"
Arrays as input to compiled functions

\[
\begin{bmatrix}
4 & 1 & 0 & 2 \\
4 & 4 & 0 & 5 \\
2 & 0 & 4 & 0 \\
3 & 1 & 3 & 2
\end{bmatrix}
\]

\[
\text{param1} = \text{eval(sprintf('param1 = %s', param1))}
\]

\[
\begin{array}{cccc}
4 & 1 & 0 & 2 \\
4 & 4 & 0 & 5 \\
2 & 0 & 4 & 0 \\
3 & 1 & 3 & 2
\end{array}
\]
Writing swarm files and calling swarm

Cell arrays as input to compiled functions

```
>> param1 = {'Bill','Steve','Linus','Cleve'}

param1 =

    'Bill'    'Steve'    'Linus'    'Cleve'
```
param1 = {'Bill','Steve','Linus','Cleve'}

param_str = []; for ii = 1:length(param1)
    param_str = [param_str '''' param1{ii} '''','];
end

param1 = ['''{'' param_str(1:end-1) '}''''];

>> param1

param1 =

"{"Bill","Steve","Linus","Cleve"}"
Writing swarm files and calling swarm

Cell arrays as input to compiled functions

```
run_my_function.sh /usr/local/matlab-compiler/v90  "{"Bill","Steve","Linus","Cleve"}"
run_my_function.sh /usr/local/matlab-compiler/v90  "{"Hank","Dean","Doc","Brock"}"
run_my_function.sh /usr/local/matlab-compiler/v90  "{"Sheila","Triana","Kim","Sally"}"
...
run_my_function.sh /usr/local/matlab-compiler/v90  "{"Walt","Jesse","Gus","Mike"}"
```
Writing swarm files and calling swarm

Cell arrays as input to compiled functions

```matlab
param1 =
{
'Bill','Steve','Linus','Cleve'
}
>> eval(sprintf('param1 = %s',param1))

param1 =

    'Bill'    'Steve'    'Linus'    'Cleve'
```
Writing swarm files and calling swarm.mat files as input

```matlab
>> param = round(rand(4)*10)

param =

4  1  0  2
4  4  0  5
2  0  4  0
3  1  3  2
```
Writing swarm files and calling swarm

.mat files as input

% how many files?
fileN = 4;

% make a directory for .mat files
mat_dir = '~/mat_dir';
if ~isdir(mat_dir)
    mkdir(mat_dir)
end

% generate arrays and save in .mat files
for ii = 1:fileN
    param = round(rand(4)*10);
    filename = sprintf('%s.mat',num2str(ii));
    save(fullfile(mat_dir,filename),'param');
end
% list all the files in a directory
mat_dir = '~/mat_dir';
file_list = what(mat_dir);
file_list = file_list.mat;

% make a command on a new line for each file
command_list = [];
for ii = 1:length(file_list)
    command_list = [command_list ...
                    'run_my_function.sh ...
                    '/usr/local/matlab-comiler/v90 ...
                    file_list{ii}...
                    '\n'];
end

% write the commands into a swarm file
file_handle = fopen('myjobs.swarm','w+');
fprintf(file_handle,command_list);
fclose(file_handle);
Writing swarm files and calling swarm
.mat files as input

run_my_function.sh /usr/local/matlab-compiler/v90 1.mat
run_my_function.sh /usr/local/matlab-compiler/v90 2.mat
run_my_function.sh /usr/local/matlab-compiler/v90 3.mat
run_my_function.sh /usr/local/matlab-compiler/v90 4.mat
function my_function(filename)

load(filename, 'param')

% param =
% 4 1 0 2
% 4 4 0 5
% 2 0 4 0
% 3 1 3 2

% code that uses input1 below...
% ...
Writing swarm files and calling swarm

Calling swarm to run file in terminal

$ swarm -f myjobs.swarm
10258332

From within MATLAB

>> !swarm -f myjobs.swarm
10258333
Writing swarm files and calling swarm

Even better from within MATLAB (capture the job id)!

```matlab
>> jobid = evalc('!swarm -f myjobs.swarm')

jobid =

10259035
```
Writing swarm files and calling swarm

Setting up dependencies with job ids

$ swarm -f myjobs.swarm
10258332

$ swarm -f anotherjob.swarm -dependency=afterany:10258332
Writing swarm files and calling swarm

Setting up dynamic dependencies with captured job ids in MATLAB

```matlab
>> jobid = evalc('!swarm -f myjobs.swarm');

>> eval(sprintf('!swarm -f anotherjob.swarm dependency=afterany:%s', jobid))

10258333
```
Writing swarm files and calling swarm

Setting up dynamic dependencies with captured job ids in MATLAB

```
>> jobid = evalc('!swarm -f job1.swarm');

>> jobid = evalc(sprintf('!swarm -f job2.swarm dependency=afterany:%s',... jobid));

>> jobid = evalc(sprintf('!swarm -f job3.swarm dependency=afterany:%s',... jobid));

>> jobid = evalc(sprintf('!swarm -f jobN.swarm dependency=afterany:%s',... jobid));

Etc...
```
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)
  $ cp -r /data/classes/matlab/swarm_example /data/$USER
Outline

• Steps to running parallel MATLAB jobs
  • Developing code interactively
  • Compiling code
  • Writing swarm files and calling swarm
  • Monitoring jobs

• A concrete example (processing image files)
  $ cp -r /data/classes/matlab/swarm_example /data/$USER
Monitoring jobs

$ squeue -j 10258333
$ squeue -u username

$ jobload -j 10258333
$ jobload -u username

$ sjobs -j 10258333
$ sjobs -u username

$ jobhist jobid
Monitoring jobs dynamically

```python
>>> eval(sprintf('!squeue -j %s', jobid))
>>> eval('!squeue $USER')

>>> eval(sprintf('!jobload -j %s', jobid))
>>> eval('!jobload $USER')

>>> eval(sprintf('!sjobs -j %s', jobid))
>>> eval('!sjobs $USER')

>>> eval(sprintf('!jobhist %s', jobid))
```
Monitoring jobs

myjobs.o
myjobs.e
Time for a break