The University of Queensland

Research Computing Centre

How to use MATLAB on UQ HPC Systems

Batch System Upgrade to PBSPro

In late 2017, all three clusters (Awoonga, FlashLite and Tinaroo) changed their batch system from Torque to PBSPro.

This was necessary to obtain better performance of the batch system under extreme loads and also to provide a number of diagnostic tools to help us, to help users.

A "wrapper script" has been deployed so that, in most cases, existing Torque job submissions will be understood and translated into PBSPro systax as the job is submitted.
You should work to migrate your job scripts to PBSPro syntax, and can use the `qstat -f JOB_NUMBER` command to learn about the PBSPro syntax corresponding to your submitted torque syntax.

A PDF copy of the PBSPro User Guide can be obtained from the Altair PBSWorks website for a copy of the user guide.
The user guides and documentation modules and other information about using the clusters currently contain examples of Torque syntax or output.

The following section or document may contain torque syntax or sample torque output.
It will be updated as soon as possible.
When a page or section has been updated, this notice will be removed from it.

Document Status

This update: July 29 2020 by David.Green @uq.edu.au

Introduction

It is no longer a contrary to the end user license agreement to submit batch jobs on UQ HPC systems that call the MATLAB software directly.

MATLAB (http://www.mathworks.com.au) is a powerful technical language and runtime environment for technical and scientific computation. Matlab is available on the HPCU grid computing environment, but there are a number of differences between the way you run Matlab on a workstation compared to a HPC cluster environment. The nuances of these procedures can have a significant effect on the performance and correctness of your code, as well as the performance and licence availability for other users. This guide will help your Matlab programs run well in our environment.

For a more general introduction to creating and running MATLAB code, see MATLAB's online help system (type "help" into the Matlab prompt) or consult the documentation on the MathWorks website. This guide assumes you are familiar with the basic concepts of creating and running Matlab programs, and are also familiar with the basics of logging in to HPC and submitting jobs to PBS. There is an extensive help system available on HPC, available with the 'man' command. For example, to find help with the PBS system, type

```
user@node:~> man pbs
```

This manual page will also contain references to the PBS commands available, like the job submission command 'qsub'. For specific help with qsub, run 'man qsub'. For help with using the manual system, run 'man man'.

Running MATLAB via the Batch System

Running Matlab in an Interactive PBS session

Under most circumstances, our licence agreement requires users to run Matlab interactively, i.e. with the qsub -I option. Begin by requesting a compute node on the cluster:

```
user@node:~> qsub -I -A groupAccount -l select=1:ncpus=4:mem=10GB -l walltime=02:00:00
```

When your job has started, you will be presented with a different prompt. Instead of the hostname "node", your prompt will display the hostname of the node your job is running on. If your prompt still looks like the login node, something went wrong with your interactive session. **Do not run MATLAB until you have successfully started an interactive job running on an allocated cluster node!** You must load the "matlab" module to gain access to MATLAB:

```
user@compute:~> module load matlab  (best practice is to call a specific version so you know which one you actually
```

Once the module is loaded, you may start Matlab by typing:

```
user@compute:~> matlab
```

If you have a valid display environment (see section 3.1 for help with displaying graphics), you will see the Matlab GUI windows, as if you had run Matlab on your local machine. Otherwise, Matlab will present a warning about displaying graphics then load the text console. This console is exactly like the console window in the GUI version (i.e. the window with the '>> ' command prompt). You can run Matlab commands, .m files or type code directly into the console as you would in the GUI console.

To exit MATLAB again, type 'exit' or 'quit'. Type 'exit' again to complete your interactive PBS job and return to the HPC cluster login node.

Interactive jobs are restricted to one compute node. Due to performance considerations (see section 1.2), you are strongly advised to test your code on a regular node first, and with short walltime requests (less than a few hours) to ensure that its performance actually improves when you increase the number of cpus.

While you are running your interactive job, you will continue to have access to your file areas e.g. /home/username. However, you will also have access to the node's local storage, $TMPDIR, which will be faster than the network attached storage areas like /home and /30days. This is strongly recommended if your program frequently reads from or writes to other files. Files in TMPDIR will be automatically deleted upon completion of your interactive job. Copy these files from /scratch to another area before you exit the job.

To end your interactive session, just type

```
user@compute:~> exit
```

and you will return to the head node from which you originally ran qsub.

If you want to capture the text output of your program, you will need to redirect MATLAB's output to a file on the command line. For example:

```
user@compute:~> matlab < my_program.m > my_output.txt
```

This command takes the Matlab program contained in my_program.m, runs it and saves the output into my_output.txt.

Running MATLAB in a Regular PBS job

It is no longer a contrary to the end user license agreement to submit batch jobs on UQ HPC systems that call the MATLAB software directly.

Create a job submission script file called myJob.pbs

```
#!/bin/bash
#
#Bare bones script to load Matlab and run a non-graphical program for upto one week
#
#PBS -m abe
#PBS -A groupAccount            #<< REPLACE THIS WITH YOUR CORRECT account string
#PBS -l select=1:ncpus=8:mem=10gb
#PBS -l walltime=168:00:00

module load matlab/2015b
echo
which matlab
echo
#
cd $TMPDIR

########
#EITHER#
########
#Redirect the m file
matlab -nodisplay -nojvm -nosplash < $PBS_O_WORKDIR/my_program.m

####
#OR#
####
#invoke the script this way (DCG thinks that the main program needs to have the same name as the m-file
#please check this yourself
matlab -nodisplay -nojvm -nosplash -r $PBS_O_WORKDIR/my_program

#DO NOT FORGET TO COPY ANY OUTPUT FILES YOU GENERATE IN $TMPDIR BACK TO $PBS_O_WORKDIR (or WHERE EVER THEY SHOULD BE
```

then `qsub myJob.pbs` from the directory that contains the my_program.m file

Note you may need to manually set your search path if you are incorporating functions that you have written.
See http://au.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html

Running MATLAB in a Job Array for Different Parameter Values

It is no longer a contrary to the end user license agreement to submit batch jobs on UQ HPC systems that call the MATLAB software directly.

Scenario:
You have a MATLAB script that operates on a single integer parameter.
The MATLAB script is located in the directory from where you will submit the batch job.
You want to run the script with a sequence of values of the input parameter.

Use a job submission script file like this

```
#!/bin/bash
#
#PBS -m abe
#PBS -A groupAccount                  #<< REPLACE THIS WITH YOUR CORRECT account string (use the groups command)
#PBS -l select=1:ncpus=8:mem=10GB     #<< Modify to suit your cluster and needs
#PBS -l walltime=168:00:00
#PBS -J 1-5                           #<< Modify to suit your parameter values

#Load the MATLAB software module
```

```
module load matlab/2019b

#Copy the matlab code into the TMPDIR
cp $PBS_O_WORKDIR/parameter.m $TMPDIR

#Change into the TMPDIR
cd $TMPDIR

# Pass a single argument to the main function in an m-file called parameter.m
matlab  -nodisplay -nojvm -nosplash -r "parameter($PBS_ARRAY_INDEX)"

#DO NOT FORGET TO COPY ANY OUTPUT FILES YOU GENERATE IN $TMPDIR BACK TO $PBS_O_WORKDIR (or WHERE EVER THEY SHOULD BE
```

Running MATLAB in a Job Array for Different Input Directories

It is no longer a contrary to the end user license agreement to submit batch jobs on UQ HPC systems that call the MATLAB software directly.

Scenario:
You have a MATLAB script that operates on a single directory containing input data file(s).
You want to run the same script across a set of input directories using a job array.
The code and data directories all hang below the same starting point.

Assumes that

- everything associated with this experiment is contained within a directory called Experiment
- the script to run is called myScript.m
- the script to run is located in the directory Experiment/Code
- the input data files are in the Experiment/Data[123]/Inputs
- the job script is submitted from Experiment

```
Experiment/Code/myScript.m
Experiment/Data1/Inputs
Experiment/Data1/Outputs
Experiment/Data2/Inputs
Experiment/Data2/Outputs
Experiment/Data3/Inputs
Experiment/Data3/Outputs
```

```
#!/bin/bash
#
#PBS -m abe
#PBS -A groupAccount              #<< REPLACE THIS WITH YOUR CORRECT account string
#PBS -l select=1:ncpus=1:mem=2gb  #<< Modify to suit your cluster and needs
#PBS -l walltime=168:00:00
#PBS -J 1-3                       #<< Modify to suit your parameter values

module load matlab/2016a
#
cp -rp $PBS_O_WORKDIR/Data${PBS_ARRAY_INDEX}/Inputs $TMPDIR

#Use -r with the name of m file without the m
matlab  -nodisplay -nojvm -nosplash -r Code/myScript

#You should copy your outputs from TMPDIR back to $PBS_O_WORKDIR/Data${$PBS_ARRAY_INDEX}
```

Making your MATLAB Job Scripts More Resilient

Sometimes the MATLAB license server is momentarily unreachable from the HPC.
When this happens your job will almost certainly fail because MATLAB was not able to get a license when it needed it.
To avoid this happening you can check the status of the MATLAB license and only proceed with the rest of your script once the license server responds positively.

The following snippet of code could be added into your job script before the point where you run matlab.
You will need to modify it for different MATLAB versions and for how many times and how long you want to wait each time.

```
module load matlab/2018a

#Will retry up to 12 times with a wait of 15 minutes each time (so 3 hours in total)
for iwait in `seq 1 12`
do
  /sw/Matlab/R2018a/etc/glnxa64/lmutil lmstat -c /sw/Matlab/R2018a/licenses/network.lic > /dev/null
  #The special shell variable $? will equal 0 if good, not 0 if bad.
  if [ $? -eq 0 ]; then
```

```
    echo "License is OK"
    break
  else
    echo "License status failed on attempt $iwait"
    #wait 15 minutes each time
    sleep 15m
  fi
done
```

Running Matlab in text-only mode

Matlab does not need a graphical display to run, and unless it is absolutely necessary, you should run your programs in text mode for performance reasons. See section 1.1 for general help with setting up and using a text mode Matlab session.

Using 'screen'

Barrine contains a command called 'screen' that allows you to create multiple persistent virtual terminal sessions from your actual terminal session. Log in to Barrine and just run the command:

```
user@barrine:~> screen
```

(press enter again to clear the splash page)

You will then be able to create terminal sessions (type "man screen" to see the manual page, or once you're running screen you can type "Ctrl-a ?" to see the list of commands).

Almost all commands are based on typing Ctrl-a then another character. You can type "Ctrl-a c" to create new screens and "Ctrl-a n" to cycle through them (or "Ctrl-a 2" to switch straight to screen 2, screen 0 is the first one).

Once you've created as many screens as you want, you can run the qsub -I etc. in each screen and get independent interactive sessions running on them. Then, you can detach the screen session with "Ctrl-A d", which will take you back to the barrine login node where you originally ran screen. Your jobs will still be running, and you can even log out, log back in later on another computer, and get your screen sessions back by running:

```
user@barrine:~> screen -r
```

and you will get all your screens with their running jobs back! Note that any programs that were running on your screen sessions continue to run while you are disconnected from them.

If you want to remove a screen and leave the others, switch to the one you want to destroy and type "Ctrl-d". You will switch to the previous screen in your session (or exit back to your original barrine session if you destroy the last screen).

The screen command is by far the best way to run terminal sessions, mostly because if you lose the network connection or have to reboot your computer, you won't lose the running jobs. You'll just have to reattach the screens with "screen -r" when you log back in.

Compiled Matlab programs

Why Bother With Compilation?

Matlab has the ability to compile .m files into standalone machine-language applications.

There are several factors to consider about compilation versus executing .m files inside Matlab's interpreter:

1. The first is performance - compiled code generally runs faster than interpreted code, although the difference has been diminishing with recent Matlab versions.
2. The second is licensing - whilst compiled Matlab programs do not require a Matlab licence to run, although you need a license for the compiler to compile them with.
3. The third is freedom - **the legal constraints on using Matlab in batch-mode computing environments do not apply to compiled code**.
   You can have PBS launch your compiled code for you.
4. The fourth is limitations - unfortunately Matlab codes that use some of the toolbox functions may not be able to be compiled and run with MCR.
   Visit http://au.mathworks.com/products/compiler/supported/compiler_support.html for details.
5. The fifth is parallelism - Matlab allows up to twelve workers (cpus) to execute compiled code without the use of the Matlab Distributed Computing Server product which is not currently part of the UQ campus license.

So using the Matlab compiler (to build) and MCR (to run) your Matlab code means that campus license tokens remain available for other UQ users, you are free to run as many instances of your Matlab code as the batch system can

accommodate and you can offload all the running of the code to the batch system with a clear conscience.

Code built with a particular version of the compiler (i.e. release of Matlab) should always be used with the corresponding release version of the MCR.
See http://au.mathworks.com/products/compiler/mcr for table of equivalences.

The barrine environment modules provide information about the equivalences:

```
uqdgree5@b10b10:~>module help mcr/v717

----------- Module Specific Help for 'mcr/v717' ------------------

Sets up the paths you need to the Matlab Compiler Runtime.

MCR Version 7.17 belongs to Matlab Release R2012a.

MCR Version 7.17 resides in /sw/MCR/v717.
```

The Basic Steps

To compile and use Matlab programs, you must undertake several steps.

1. Load the Matlab module so you can access the Matlab compiler
(it is a best practice to specify the exact version and not to rely upon the current and changeable default)

```
user@barrine:~> module load matlab/2012a
```

2. Load the Matlab Compiler Runtime module (although this is not strictly necessary)
(it is a best practice to specify the exact version and not to rely upon the current and changeable default)

```
user@barrine:~> module load mcr/v717
```

3. To invoke the matlab compiler on your main program, use the following command:

```
user@barrine:~> mcc -m myMatlabScript.m
```

The -m filename switch specifies the .m file that contains the start point of your code
(i.e. if you have a main.m which calls a function defined in a separate file like function.m, you need only add the main file to the compiler command).

To ensure that your code only uses one thread when it runs (the default behaviour is quite anti-social!)

```
mcc -v -R -singleCompThread -m test.m
```

or to obtain verbose verbose compiler output, which you may need in order to debug your code.

```
user@barrine:~> mcc -v -m myMatlabScript.m
```

On a linux platform, the compiler will produce a linux shell script and some other executables that you will need to use to run the compiled program.

If the compilation fails, it may be necessary to remove your .matlab cache directory and re-compile (on Barrine this should be ~/.mcrCache7.17 or whatever version you're using).

4. To run your compiled Matlab program you need to call the created shell script with some arguments:

cd into the directory where your program is, then run:

```
user@aNNbNN:~> ./run_myMatlabScript.sh /path/to/MCRroot argument1 argument2 ...
```

The first argument (`/path/to/MCRroot`) is the path to the Matlab runtime installation for your required version.

You can find out this location several ways (after loading the modules)

A module load of the correct MCR module will set an environment variable called MCR

```
uqdgree5@barrine1:~> module load mcr/v717
user@aNNbNN:~> echo $MCR
/sw/MCR/v717
```

Alternatively, the module help for the required MCR module will provide useful information

```
uqdgree5@b10b10:~> module help mcr/v717

----------- Module Specific Help for 'mcr/v717' ------------------

Sets up the paths you need to the Matlab Compiler Runtime.

MCR Version 7.17 belongs to Matlab Release R2012a.

MCR Version 7.17 resides in /sw/MCR/v717.
```

That is, the path you need to pass to run_myMatlabScript.sh is '/sw/MCR/v717' if you compiled with Matlab 2012a.

The remaining arguments are passed to Matlab program as input arguments.

WARNING: Arguments passed to your compiled program are of type "string", and if you wish to pass a number, you must first cast it to a numerical type (in your .m file) either with the str2num() or str2double() functions. For example:

In your script file called myMatlabScript.m you would need to have something like

```
function result = myMatlabScript(input)  % This code takes a number as input
input=str2num(input);  % which must be cast from string to number to work.
result = input+1  % Use the input in some way.
end
```

After compilation, you can run

```
./run_myMatlabScript /sw/MCR/v717 5
```

and Matlab will return "result = 6".

If you wish to pass an array of values, you must surround them with single or double quotes or they will be considered as separate arguments, e.g.

```
./run_myMatlabScript.sh /sw/MCR/v717 "[1 2 3 4]"
```

and Matlab will return "result = 2 3 4 5". The square brackets are not necessary as str2num() will add them if they are missing.

WARNING: If you wish to pass a column vector or a multidimensional array, you must escape the semicolons with backslashes, e.g.

```
./run_myMatlabScript.sh /sw/MCR/v717 "[2 3\; 6 9]"
```

or else the semicolon will be interpreted as a command separator.

Matlab Functions Calling Your Functions

We have noticed that user defined functions that are called from within Matlab functions (eg. the optimization and ODE functions) can misbehave.
In particular, you can get a "MATLAB:UndefinedFunction" error when your program tries to run.

If your code calls a Matlab function that then calls your own function by name, that is,

```
MatlabsOptimiser('MySpecialFunction',arg1,arg2,...)
```

you may experience problems.
One workaround for this is to fake a call of your function before the Matlab function calls it.
This will ensure that your function is included in the namespace when the code is run under MCR.

```
%fake a call to the function that is used to optimise
tossaway=MySpecialFunction(arg1,arg2, ... );
```

A more elegant/correct approach is to declare the function ahead of time by using a Function Handle Constructor.
This involves adding @ on the function name, instead of quotes:

```
MatlabsOptimiser(@MySpecialFunction,arg1,arg2,...)
```

Visit http://au.mathworks.com/help/matlab/matlab_prog/symbol-reference.html for more information.

Embarrassingly Parallel Jobs

You can perform embarrassingly parallel computations with one serial compiled code, perhaps passing different input arguments to each instance of the code you run.
You can then submit several jobs or a job array (using the -t switch), each job requesting one cpu and its own copy of the code.

An example job script that we shall call myJobScript would look like this:

```
#PBS -A groupAccount
#PBS -q workq
#PBS -l select=1:ncpus=1:mem=2GB
#                       ^ We have assumed that you have used the single thread compilation option!!!

#PBS -l walltime=hh:mm:ss
#PBS -J 1-100

#Need to avoid performance related job failures by using local node disk for MCR cache
#Even if your jobs are not failing, I am convinced it will make your jobs start and perhaps run much faster. It will
export MCR_CACHE_ROOT=${TMPDIR}

myCompiledMatlabProgram /sw/MCR/v717 ${PBS_ARRAY_INDEX} > output_${PBS_ARRAY_INDEX}.txt
```

Then you just have to type:

```
user@barrine:~> qsub myJobScript
```

and it will queue then run your code 100 times, each time passing your code a different input argument (taken from the PBS-supplied variable $PBS_ARRAY_INDEX), then using the variable again to place the output of each job into a uniquely named file.
You will need to handle the division of computer labour yourself, somewhere in your Matlab code.

Compiled parallel Matlab programs

To compile Matlab programs to work in parallel, you must add a configuration file to specify the number of available workers.
~~A configuration file for medium nodes (barrinemedium.mat) is available. [* MUST DEPLOY THE .MAT SOMEWHERE *]~~

By far the easiest way to build a compiled Matlab program with this file included is to use the "deploytool" command inside Matlab (in graphical mode).

```
> deploytool
```

This brings up a dialog box asking to create a project and add files to it. Add the .m file that you would run to start your program as the main file, then add any other necessary files as well as the .mat configuration file as additional files. Then click "build" and wait for Matlab to compile your application. Note that this process can take some time and use significant compute resources. Therefore, you should compile your program from within an interactive PBS job, as described in Section 1.1.

Once the program is compiled, you may run it within a PBS job as described above.

You may also compile the program in the same way as described above, from the command line using the 'mcc' command. You don't need to specify the .mat file or other files you need, the Matlab compiler will include them automatically so long as it can find the files, which it will if they're in the same directory as the main .m file in your program.

For more information and a simple example, see the relevant Mathworks page http://www.mathworks.com.au/help/toolbox/compiler/bsl9c8_.html

Compilation Terms and Conditions

See http://www.mathworks.com.au/help/toolbox/compiler/index.html and http://www.google.com/search?q=matlab+mcr+toolboxes and links therein for more information on compiling Matlab code.

Certain Matlab features and toolboxes are not available in compiled mode or with parallel code built with the Parallel Computing Toolkit. You can see the full list of explicitly unavailable products at http://www.mathworks.com.au/products/ineligible_programs/

Graphical output without a graphical display

Figures can be written directly to files from within Matlab programs using the print function:

```
myfig = figure;  % Create a figure object
plot(my_data);   % ... and put something in it
print(myfig,'-dpng','my_data.png');  % Then save it as a PNG file
```

This is the best way to generate figures on the HPCU facility, as it does not actually need a display and saves the figures to files without you having to remember to do that after your program has finished.

Running Matlab with its graphical interface

Creating and updating complex graphics while running heavy computations in MATLAB is not recommended. It is better to create output data, as a mat file, then analyse the data separately.

You will be interacting with the HPC facility through a terminal interface (ssh session). If you want to have access to Matlab's graphics capabilities, you will need to forward an X11 session through your terminal application. Note that this is not needed to create graphics (or save them to a file), only to display them while your code is running.

Connecting HPC graphics to your local display

More detail on this topic is provided in the "Connecting to HPC" User Guide.

For Mac and Unix users, all you need to do is add a switch to your ssh command:

```
user@yourmachine:~> ssh -Y username@hostname
```

You are advised to use one of the physical login nodes not the cluster alias

Windows users will need to install an X window server such as Xming (http://sourceforge.net/projects/xming/) in order to display graphics. Your terminal software will have the ability to enable X forwarding. See your terminal software's documentation for instructions. It may be necessary to use -Y instead of -X, to enable trusted X forwarding. If you are using PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/), the relevant setting is in Settings category: Connection -> SSH -> X11, then check the box marked "Enable X forwarding".

The qsub command can export the DISPLAY environment variable with the use of the -X switch. You will need to do this to display graphics from within the interactive session.

```
user@login_node:~> qsub -I -X ...
```

You may be used to including a lot of graphics processing code in your Matlab programs, perhaps to generate and display figures as your code produces the associated data. This is often unnecessary and can slow your program down significantly. Be especially careful of updating figures a large number of times, e.g. every time the source data changes as part of a loop. If you absolutely must display and update figures, modify the rate at which it updates to reduce the computational cost of doing so, e.g.:

```
for i=1:1000000  % large loop!
  do(something);
  if mod(i,10000)==0  % Only update every 10000 iterations, i.e. 1% of the time
    plot(new_data);
  end
end
```

Compiled Matlab programs with graphics

There is no particular configuration necessary to run compiled Matlab programs with graphics enabled. Refer to section 2.3 regarding compiled Matlab programs, and section 3.1 regarding using graphics. Note that if your compiled program is running in a non-interactive PBS job, you will not be able to display graphics from that job. Refer to Graphical output without a graphical display section for saving graphics to files from within your Matlab program.

Using the Parallel Computing Toolkit

The HPCU facilities are shared among many users, and everyone must be efficient in requesting and monitoring their jobs' resource requirements. Jobs that request more cpus, memory or disk space than they really need run the risk of being cancelled by the system administrators. It is strongly recommended that you test your code with a short walltime to check that the resources you requested are appropriate to the task.

Using more cpus does not necessarily result in a faster execution time. Each program must spend some fraction of its time co-ordinating the resources among the participating compute nodes, which is time not spent executing the more interesting code. More cpus means more co-ordination overhead. You will find that, for your particular program and input, there is an

optimum number of cpus that balances the speed-up of using more cpus with the slowdown of co-ordination overhead. For smaller jobs, this is likely to be in the 2-4 cpu range. Again, test your code while varying the number of cpus you use. Requesting too many will slow your program down as well as make those cpus unavailable to other users!

FYI:
The parallel computing toolkit relies on Java.
You cannot use parallel computing toolkit with the `-nojvm` option to matlab.

Here is a simple example of a parallel for loop, run multiple times on a pool of 1-8 workers, with measurements of the execution time of the parallel part, for each size of workpool.

```
% Execute matlab code in parallel on multiple cores using matlabpool%

workSize = 512;  % Controls the size of the overall task to perform
maxWorkers = 8;  % Max. number of cores (8 is a whole node)
inData = rand(workSize);  % Create some input to work with
outData = zeros(workSize); % Create a matrix to store the output
timeData = zeros(maxWorkers,1);  % A place to store the timing data

for poolSize = 1:maxWorkers

  % Open the pool
  matlabpool(poolSize)  % Warning: you must call matlabpool with brackets
                        % if the number of workers is a variable.
  tic  % Start a timer
  parfor i=1:workSize
    outData(i,:) = inData(i,:) + i;
  end
  timeData(poolSize) = toc;  % Store the elapsed time since 'tic' in timeData

  matlabpool close
end

% inData  % the actual data is pretty big at workSize=512, so suppress output
% outData
timeData
```

The output is:

```
>> >> >> >>
timeData =

    5.1476
    1.4058
    0.1100
    0.1337
    0.1739
    0.1664
    0.2184
    0.2537
```

For this particular program and input data, the optimum number of cpus is 3.

Controlling CPU Utilisation

MATLAB has always had a reputation for being "greedy".
Unless moderated, MATLAB will attempt to use all available CPUs.

Fortunately there are some ways to rein it in ...

Forcing execution to be restricted to a single CPU

```
matlab -singleCompThread ...
```

Using the maxNumCompThread Parameter

Referring to this MathWorks user support page

```
%
LASTN = maxNumCompThreads(8);  %Sets LASTN to the previous setting and sets the current value to 8
THISN = maxNumCompThreads();   %Sets THISN to the new value (that is, 8)
%
disp('Values for maxNumCompThreads before and after setting to 8')
disp(LASTN)
```

```
disp(THISN)
%
```

Using parpool and parfor

See the example elsewhere in this guide of using the Parallel Computing Toolbox.

Calling Python Code from MATLAB

MATLAB code can interoperate with python code.
There is a python API engine for elaborate situations.
You can also use the system function call in simple situations such as is illustrated here.

On the HPC systems, there are three categories of python available:

- the linux distribution python (currently 2.6.6)
- ROCKS installed python (currently 2.7.13)
- Anaconda python (various versions can be invoked)

Anaconda is a tool that helps you to create and use custom python environments.

In this simple demo, we are calling a python script using Python 3.6 provided by an anaconda python environment.

```
uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> module list
Currently Loaded Modulefiles:
  1) matlab/2017a     2) anaconda/4.3.1

uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> conda info --envs
# conda environments:
#
Python-2.7               /sw/RCC/Anaconda/4.3.1/envs/Python-2.7
Python-3.6               /sw/RCC/Anaconda/4.3.1/envs/Python-3.6
ALLMAPS                  /home/uqdgree5/.conda/envs/ALLMAPS
My-Biopython-2.7         /home/uqdgree5/.conda/envs/My-Biopython-2.7
python-2.7.3             /home/uqdgree5/.conda/envs/python-2.7.3
tensorflow               /home/uqdgree5/.conda/envs/tensorflow
root                  *  /sw/RCC/Anaconda/4.3.1

uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> source activate Python-3.6

(Python-3.6) uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> cat matlab+anaconda.m
%MATLAB script that will system call a simple python script to see what happens

%Must cast argument for system command as char
[status,cmdout] = system(char("which python; python test.py"));
%[status,cmdout] = system(char("/bin/date"));

status

cmdout

exit;

(Python-3.6) uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> cat test.py

print("Hello, World!")

(Python-3.6) uqdgree5@awoonga1:~/Demo/MATLAB/WithPython>

(Python-3.6) uqdgree5@awoonga1:~/Demo/MATLAB/WithPython> matlab -nodisplay < matlab+anaconda.m

                            < M A T L A B (R) >
                  Copyright 1984-2017 The MathWorks, Inc.
                   R2017a (9.2.0.538062) 64-bit (glnxa64)
                             February 23, 2017

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> >> >> >> >> >> >>
status =

     0

>> >>
cmdout =

    '/sw/RCC/Anaconda/4.3.1/envs/Python-3.6/bin/python
```

```
    Hello, World!
    '

>> >>
(Python-3.6) uqdgree5@awoonga1:~/Demo/MATLAB/WithPython>
```

License Conditions

The University of Queensland has signed a campus agreement with The Mathworks for the core components as well as a wide collection of toolboxes.

In plain english, this agreement says

1. only staff and students of UQ may access the licensed software from UQ equipment
2. only staff of UQ may access the licensed software from their own equipment (you will not be allowed to install onto your own computer if you are a student ... contact UQ ITS HelpDesk for clarification)
3. no user is permitted to use Matlab in a "cluster, grid, Web server, server farm, or other similar scheduled environment" - that is, you should **not** invoke the matlab command from within a non-interactive batch job script (**this condition is no longer enforced**)
4. we believe that "network servers" would include compute nodes of the HPC cluster PROVIDED you use the product interactively - that is, you need to type the command matlab yourself in an interactive session (**this condition required as the no-batch condition is no longer enforced**)
5. if you use the Matlab compiler to compile your program into a standalone executable, then you can run it where ever and whenever you like without requiring a license token

Legal Information

The following is a synopsis of the Terms and Conditions from the Campus Agreement:

1. Definitions

1.1. Qualified Internal Operations means the installation and use of the Programs and Documentation by Licensed Users, in accordance with the TAH License Option selected, for the purpose of (i) in the case of employees (faculty and academic staff), performing software administration, teaching, and non-commercial, academic research in their ordinary course as Licensee's employees; and ii) in the case of enrolled students, meeting classroom requirements of courses and study offered by the Licensee. Any other use is expressly prohibited by this Agreement or the Software License Agreement. As used herein, "employees" excludes subcontractors and consultants of the Licensee.

1.2. Licensed Users means all enrolled students and employees (faculty and academic staff) of the Licensee (the Total Academic Headcount) who are authorized by Licensee to use the Programs for Internal Operations in accordance with this Agreement as revised from time to time, and to the extent permitted by the TAH License Option selected in the TAH Schedule.

1.5. License. MathWorks' grant to Licensee, pursuant to the terms of The MathWorks, Inc. Software License Agreement, of a nontransferable, limited, and temporary TAH License for the License Term to install and use the Programs and Documentation by Licensed Users for Qualified Internal Operations only. Programs licensed under the TAH License Option are restricted to use solely by degree-granting educational institutions and only for non-commercial, academic use by their faculty, academic staff, and students. The right to use the Programs for any other purpose is expressly prohibited. Research and development divisions and centers of universities, U.S. government agencies and other not-for-profit organizations do not qualify for the TAH License Option. MathWorks shall make the sole determination of Licensee's qualification for any TAH License Option.

3. INSTALLATION, USE and ADMINISTRATION

3.1.1. TAH - Campus Option. During the License Term, Programs may be installed and used by Licensed Users on Licensee-owned or leased individual, standalone computers and network servers, **provided the Programs are not run in a cluster, grid, Web server, server farm, or other similar scheduled environment**. Licensee's faculty and academic staff who are Licensed Users may install and use the Programs on their personally owned computers, on campus, off campus, and via remote access. Under this TAH - Campus Option, students qualify as Licensed Users solely for the purpose of using the Programs in on-campus computing facilities and are expressly prohibited from installing and executing the Programs on their personally owned computers.

Adding MATLAB Distributed Computing Server (MDCS) to the TAH - Campus Option. Licensee may acquire MATLAB Distributed Computing Server "MDCS" for use with the Programs licensed under a TAH Campus License. Your rights and obligations with respect to the MDCS are as set forth in The MathWorks, Inc. Software License Agreement, except for the following:
(a) When a TAH - Campus Option Licensee has paid the fee to include MDCS as part of its TAH Campus Program

configuration, the restriction identified in Paragraph 3.1.1 above which prevents a Licensed User from running a Program in a scheduled environment (see first sentence of Paragraph 3.1.1 above) does not apply to scheduled environments on which the MDCS is installed and licensed.
(b) Only one MDCS can be used per scheduled environment. For scheduled environments that require more than the number of workers provided on the MDCS purchased, Licensee must purchase a separate MDCS License from the then-current Academic Price List. Such separately licensed MDCS License will not be included in Licensee's TAH License Program configuration.

The Licensed Features (as at mid-2016)

UQ users can run this command on Euramoo (similar command can be run on Tinaroo and FlashLite).

Desktop installations would also have an equivalent command available.

```
uqdgree5@eura02-gpu:~> /sw/MATLAB/R2016a/etc/glnxa64/lmutil lmstat -i -c /sw/MATLAB/R2016a/licenses/network.lic
lmutil - Copyright (c) 1989-2014 Flexera Software LLC. All Rights Reserved.
Flexible License Manager status on Sun 8/7/2016 22:25

NOTE: lmstat -i does not give information from the server,
      but only reads the license file.  For this reason,
      lmstat -a is recommended instead.

Feature                         Version     #licenses    Expires      Vendor
_______                         _________   _________    ___________  ______
MATLAB                          35           14822       01-feb-2018  MLM
SIMULINK                        35           14822       01-feb-2018  MLM
Aerospace_Blockset              35           14822       01-feb-2018  MLM
Aerospace_Toolbox               35           14822       01-feb-2018  MLM
Bioinformatics_Toolbox          35           14822       01-feb-2018  MLM
Communication_Toolbox           35           14822       01-feb-2018  MLM
Video_and_Image_Blockset        35           14822       01-feb-2018  MLM
Control_Toolbox                 35           14822       01-feb-2018  MLM
Curve_Fitting_Toolbox           35           14822       01-feb-2018  MLM
Signal_Blocks                   35           14822       01-feb-2018  MLM
Data_Acq_Toolbox                35           14822       01-feb-2018  MLM
Database_Toolbox                35           14822       01-feb-2018  MLM
Datafeed_Toolbox                35           14822       01-feb-2018  MLM
Econometrics_Toolbox            35           14822       01-feb-2018  MLM
RTW_Embedded_Coder              35           14822       01-feb-2018  MLM
Financial_Toolbox               35           14822       01-feb-2018  MLM
Fixed_Point_Toolbox             35           14822       01-feb-2018  MLM
Fuzzy_Toolbox                   35           14822       01-feb-2018  MLM
GADS_Toolbox                    35           14822       01-feb-2018  MLM
Image_Acquisition_Toolbox       35           14822       01-feb-2018  MLM
Image_Toolbox                   35           14822       01-feb-2018  MLM
Instr_Control_Toolbox           35           14822       01-feb-2018  MLM
MATLAB_Coder                    35           14822       01-feb-2018  MLM
MATLAB_Builder_for_Java         35           14822       01-feb-2018  MLM
Compiler                        35           14822       01-feb-2018  MLM
MATLAB_Report_Gen               35           14822       01-feb-2018  MLM
MAP_Toolbox                     35           14822       01-feb-2018  MLM
MPC_Toolbox                     35           14822       01-feb-2018  MLM
Neural_Network_Toolbox          35           14822       01-feb-2018  MLM
Optimization_Toolbox            35           14822       01-feb-2018  MLM
Distrib_Computing_Toolbox       35           14822       01-feb-2018  MLM
PDE_Toolbox                     35           14822       01-feb-2018  MLM
Phased_Array_System_Toolbox     35           14822       01-feb-2018  MLM
RF_Toolbox                      35           14822       01-feb-2018  MLM
Robust_Toolbox                  35           14822       01-feb-2018  MLM
Signal_Toolbox                  35           14822       01-feb-2018  MLM
SimBiology                      35           14822       01-feb-2018  MLM
SimEvents                       35           14822       01-feb-2018  MLM
SimElectronics                  35           14822       01-feb-2018  MLM
SimHydraulics                   35           14822       01-feb-2018  MLM
SimMechanics                    35           14822       01-feb-2018  MLM
Power_System_Blocks             35           14822       01-feb-2018  MLM
Simscape                        35           14822       01-feb-2018  MLM
Real-Time_Workshop              35           14822       01-feb-2018  MLM
Simulink_Control_Design         35           14822       01-feb-2018  MLM
Real-Time_Win_Target            35           14822       01-feb-2018  MLM
XPC_Target                      35           14822       01-feb-2018  MLM
Excel_Link                      35           14822       01-feb-2018  MLM
Stateflow                       35           14822       01-feb-2018  MLM
Statistics_Toolbox              35           14822       01-feb-2018  MLM
Symbolic_Toolbox                35           14822       01-feb-2018  MLM
Identification_Toolbox          35           14822       01-feb-2018  MLM
Trading_Toolbox                 35           14822       01-feb-2018  MLM
Wavelet_Toolbox                 35           14822       01-feb-2018  MLM
uqdgree5@eura02-gpu:~>
```