MATLAB Job Guide

Table of Contents

MATLAB Campus License

Since September 1, 2011, TTU obtained campus license of MATLAB. This license includes the most recent version of the MATLAB software. This version of MATLAB is installed on both Quanah and Red Raider, and is available to all TTU faculty, staff and students.

The following instructions focus on the usage under the campus license. For users who wish to use department or group owned MATLAB licenses, please contact HPCC for assistance.

Prerequisites

Before diving into MATLAB, we strongly recommend you to:

- Read the "Software Environment Setup" to understand how to **load modules** and set up your environment properly.
- Read the "Job Submission Guide" in SLURM to understand how to **submit**, **monitor** your job and set up an **interactive session** with your job.
- (Optional) Have **Xming** or **MobaXterm** set up for interactive sessions. You can refer to this "X Server" guide to set up Xming and log in HPCC with graphical interface, or refer to "Connecting to HPCC Systems" to learn more about MobaXterm and how to connect to our system.

Software Environment Setup for MATLAB

To set up MATLAB, please run the following commands:

```
module load matlab
```

By default, the latest version of MATLAB will be loaded using the above command. To check if MATLAB is currently loaded in your environment, please type:

```
module list
```

```
login-20-25:$ module load matlab
login-20-25:$ module list

Currently Loaded Modules:
  1) nocona/0.15.4   2) matlab/R2020b
```

MATLAB Job Submission

Non-interactive Job

A non-interactive job is recommended when there is no need for a graphical interface interaction between the user and the program for better computational performance. Typically, to run a MATLAB file, the file should be in the ".m" extension format. The ".m" file is where you write series of MATLAB commands that you would like to execute.

Nocona Job Submission Tutorial

Submitting a MATLAB job on Nocona can be done using the following steps:

Step 1: Copy the tutorial job script folder for nocona to your home directory

```
#Copy the folder and all of its contents to your directory

cp -r /lustre/work/examples/nocona/matlab ~/matlab
```

Inside the ~/matlab folder there should be four files available:

`A_matrix.dat`: This file stores the data of the first matrix in a comma-separated format.

`B_matrix.dat`: This file stores the data of the second matrix in a comma-separated format.

`multiplyMatrix.m`: This file contains the MATLAB commands to load the two matrices and perform matrix multiplication on them, then save the result to an ASCII file.

`matlab.sh`: This file contains the SLURM commands used for job submission. The script should look something like this:

```
#!/bin/bash
#SBATCH --job-name=matlab-test
#SBATCH --output=%x.o%j
#SBATCH --error=%x.e%j
#SBATCH --partition nocona
#SBATCH --nodes=1
#SBATCH --ntasks=1

module load matlab
matlab -batch multiplyMatrix
```

This script requests one core/task in a node (`--ntasks=1 and --nodes=1`) to be used for the job. This script also loads the module MATLAB (`module load matlab`) and executes the MATLAB command (`matlab -batch multiplyMatrix`) to run the file "multiplyMatrix.m" without displaying the splash screen and desktop in the "-batch" option for a non-interactive session.

NOTE: The ".m" extension in the "multiplyMatrix.m" file is omitted when running the "`matlab`" command in the job submission script.

NOTE: When you run your MATLAB job on other partitions (Quanah or Matador partition), in your job submission script, you need to change the option "`--partition=nocona`" to "`--partition=quanah`" or "`--partition=matador`". For Matador, make sure you also request appropriate GPU counts.

Step 2: Submit your job by typing in the following command

```
sbatch matlab.sh
```

To check the status of your job, type in the following command

```
squeue -u <username>
```

Step 3: Once the job is finished, you can view the output of your job in the file "matlab-test.o<job_ID>." In our example, the output file is "matlab-test.041439." The output should look similar to this:

```
result =

    84    90    96
   201   216   231
   318   342   366
```

The output saved to the ASCII file "result.txt" should look like this:

```
 8.4000000e+01   9.0000000e+01   9.6000000e+01

 2.0100000e+02   2.1600000e+02   2.3100000e+02

 3.1800000e+02   3.4200000e+02   3.6600000e+02
```

Nocona Single-node Parallel Job Submission Tutorial

Submitting a single-node parallel MATLAB job on Nocona can be done using the following steps:

Step 1: Copy the tutorial job script folder for nocona to your home directory

```
#Copy the folder and all of its contents to your directory

cp -r /lustre/work/examples/nocona/matlabSinglePar
~/matlabSinglePar
```

Inside the ~/matlabSinglePar folder, there is a file named `parTest.m` that looks like the contents displayed below. This file contains MATLAB commands that specify the number of parallel workers to run as well as commands for mathematical computation. Make sure that you only request up to 124 cores in the Matlab file even if you have 128 cores available in a Nocona node.

```
%Create a parallel cluster object
c = parcluster;

%Specify the number of cores for the cluster object
poolObject = parpool(c,124);


%Start timing
tic
n = 100;
A = 500;
a = zeros(n);
parfor i = 1:n*n
    a(i) = max(abs(eig(rand(A))));
end
disp(a)

%End timing
time = toc;

fprintf('The parallel method is executed in %5.2f seconds.  \n',
time);

%Delete the parallel object
delete(poolObject);
```

There is another submission file in SLURM named `matlab.sh`. This script requests 128 cores/tasks in a node (`--ntasks-per-node=128 and --nodes=1`) to be used for the job. The submission script shows that 128 cores will be allocated, but your Matlab file should only request up to 124 cores. This script also loads the module MATLAB (`module load matlab`) and executes the MATLAB command (`matlab -batch parTest`) to run the file "parTest.m" without displaying the splash screen and desktop in the "-batch" option for a non-interactive session. The

```
#!/bin/bash
#SBATCH --job-name=matlab-test
#SBATCH --output=%x.o%j
#SBATCH --error=%x.e%j
```

```
#SBATCH --partition=nocona
#SBATCH --nodes=1
#SBATCH –ntasks-per-node=128
module load matlab
matlab -batch parTest
```

Step 2: Submit your job by typing in the following command

```
sbatch matlab.sh
```

To check the status of your job, type in the following command

```
squeue -u <username>
```

Step 3: Once the job is finished, you can view the output of your job in the file "matlab-test.o<job_ID>." In our example, the output file is "matlab-test.041439." The output should look similar to this:

Nocona Multi-node Parallel Job Submission Tutorial using Matlab Parallel Server

How to Initialize Parallel Setting

In order to run parallel jobs using more than one node, you have to call an integration script to initialize the Matlab Parallel Server. This should only be done one time on each cluster. After that, you can run parallel jobs as usual without initializing this script. Please follow these steps if you want to initialize a parallel environment for your job.

NOTE: The new parallel server is used for running multiple nodes in parallel. If you are happy with running multiple cores in one node and not expanding to multiple nodes, you may skip this and keep using the old single-node parallel method.

Step 1. Request an interactive session, <partition_name> can be quanah or nocona/matador.

```
interactive -c 1 -p <partition_name>
```

Step 2. Load matlab module.

```
module load matlab
```

You can type “module list” to check if the module has been loaded.

Step 3. Load the matlab command line without displaying the user interface.

```
matlab -nodisplay
```

Please allow some time for matlab command terminal to finish loading.

Step 4. Call the configCluster.m script to initialize the parallel environment.

```
configCluster
```

After that, you should be good to go, you can ignore any warnings.

Set up Arguments for Parallel Job

The way multi-node parallel jobs are designed to run is completely different from how serial or single-node jobs run in MATLAB. When you request SLURM to run a job in parallel, you should always request a single core in your submission script. The actual number of cores/nodes that you want to run in parallel need to be requested in your MATLAB “.m” extension file. The single core you get allocated will then spawn the actual nodes and cores you request in your MATLAB file. To specify the number of cores and nodes you want to run in parallel, here is what you should add to your MATLAB “.m” file to enable parallel jobs. You can refer to the parallel job submission tutorial above as a reference for running parallel jobs.

```
%Create a parallel cluster object.

c = parcluster;

%Specify the partition, number of nodes, and save the profile
c.AdditionalProperties.Partition=<partition>;
c.AdditionalProperties.NumberOfNodes=<nodes>;
```

```
c.saveProfile

%OTHER OPTIONAL PROPERTIES FOR YOUR MATLAB JOB
% Specify an account to use for MATLAB jobs, please see this
link to check account name
c.AdditionalProperties.AccountName = 'account-name';

% Specify a reservation to use for MATLAB jobs, please see this
link to check reservation name
c.AdditionalProperties.Reservation = 'reservation-name';

% Specify e-mail address to receive notifications about your job
c.AdditionalProperties.EmailAddress = 'user-id@ttu.edu';

% Specify number of GPUs for Matador only (e.g 2 GPUs)
c.AdditionalProperties.GpusPerNode=2;

% Specify memory to use for MATLAB jobs, per core (e.g. 4000MB),
default memory per core is 4027MB for Nocona
c.AdditionalProperties.MemUsage = '4000';

% Specify the walltime (e.g. 5 hours, less time requested means
higher job priority), the default is 48 hours
c.AdditionalProperties.WallTime = '05:00:00';


%Specify the total number of parallel workers in the nodes
requested
p = c.parpool(<number of cores>);

%This command will display the current properties.
c.AdditionalProperties


%Beginning of your code

%Your code ……

%Ending of your code



%Delete the parallel object when job is finished

p.delete
```

Multi-node Parallel Tutorial

If you have not initialized a parallel environment, your job will not be able to run in parallel with multiple nodes. Please follow "How to Initialize Parallel Setting" to set up your parallel environment first before proceeding with the tutorial.

Submitting a parallel MATLAB job on Nocona can be done using the following steps.

Step 1: Log into login.hpcc.ttu.edu and copy the tutorial job script folder for nocona to your home directory

```
#Copy the folder and all of its contents to your directory

cp -r /lustre/work/examples/nocona/matlabPar ~/matlabPar
```

Inside the ~/matlabPar folder, there is a file named `parTest.m` that looks like the contents displayed below. This file contains MATLAB commands that specify the number of parallel workers and nodes to run as well as commands for mathematical computation.

```
%Create a parallel cluster object
c = parcluster;

%Specify the partition the job will be running on
c.AdditionalProperties.Partition='nocona';

%Specify the number of nodes (-N 2), save the profile, and
display properties
c.AdditionalProperties.NumberOfNodes=2;
c.saveProfile
c.AdditionalProperties

%Specify the total number of parallel workers in 2 nodes
requested
p = c.parpool(256);

%Start timing
tic
```

```
%Calculates spectral radius of each matrix and displays results
n = 100;
A = 500;
a = zeros(n);
parfor i = 1:n*n
    a(i) = max(abs(eig(rand(A))));
end
disp(a)

%End timing
time = toc;


%Display the time of computation
fprintf('The parallel method is executed in %5.2f seconds.  \n',
time);


%Delete parallel workers
p.delete
```

There is another submission file in SLURM named `matlab.sh`. This script requests one core/task in a node (`--ntasks=1 and --nodes=1`) to be used for launching the job. This script also loads the module MATLAB (`module load matlab`) and executes the MATLAB command (`matlab -batch parTest`) to run the file "parTest.m" without displaying the splash screen and desktop in the "-batch" option for a non-interactive session.

```
#!/bin/bash
#SBATCH --job-name=matlab-test
#SBATCH --output=%x.o%j
#SBATCH --error=%x.e%j
#SBATCH --partition=nocona
#SBATCH --nodes=1
#SBATCH --ntasks=1
module load matlab
matlab -batch parTest
```

NOTE: Although the `parTest.m` file requests to run 256 parallel workers on 2 nodes in Nocona, the submission script should only request one core in a node, as this one core will be the master core that spawns additional core workers and additional nodes requested from the MATLAB `parTest.m` file, thus there is no need to request 256 cores and 2 nodes in the SLURM submission script. This means that the `parTest.m` file is what will determine the amount of cores and nodes you want to run parallel jobs on.

NOTE: Make sure that the `--partition` field in your `matlab.sh` file matches the partition name you select in your `parTest.m` file. If your submission file submits to Nocona but your MATLAB job submits to Quanah (or the reverse) you will receive an error and your job will fail to run correctly.

Step 2: Submit your job by typing in the following command

```
sbatch matlab.sh
```

To check the status of your job, type in the following command

```
squeue -u <username>
```

The status of your job will first look like this after submitting your job.

```
  JOBID  PARTITION  PRIORI  ST      USER      NAME        TIME  NODES   CPUS      QOS  NODELIST(REASON)
2026655     nocona    2435   R  ngu29355 matlab-te        0:03      1      1   normal  cpu-23-10
```

The node `cpu-23-10` in this example will be the master node spawning extra workers for your parallel job. After a little while, if the queue is not too busy, the status of your job will look like this as 2 additional nodes `cpu-26-[4,45]` have been allocated based on the 2 nodes and 256 cores you requested in the `parTest.m` file properties. (These images are from Quanah, but they should be similar to Nocona except for showing the "nocona" partition and number of cores as 256)

```
  JOBID  PARTITION  PRIORI  ST      USER      NAME        TIME  NODES   CPUS      QOS  NODELIST(REASON)
2026655     nocona    2435   R  ngu29355 matlab-te        0:32      1      1   normal  cpu-23-10
2026656     nocona    2435   R  ngu29355      Job1        0:03      2    256   normal  cpu-26-[4,45]
```

Step 3: Once the job is finished, you can view the output of your job in the file "matlab-test.o<job_ID>." In our example, the output file is "matlab-test.2026655." The output should look similar to this:

```
ans =

  AdditionalProperties with properties:

             AccountName: ''
    AdditionalSubmitArgs: ''
            EmailAddress: ''
             EnableDebug: 0
             GpusPerNode: 0
                MemUsage: ''
           NumberOfNodes: '2'
               Partition: 'nocona'
            ProcsPerNode: 0
```

```
             Reservation: ''
                 UseSmpd: 0
                WallTime: ''
Starting parallel pool (parpool) using the 'redraider R2020b'
profile ...

additionalSubmitArgs =

    '--ntasks=256 --cpus-per-task=1 --ntasks-per-core=1 -p
nocona -N 2'

Connected to the parallel pool (number of workers: 256).
  Columns 1 through 7

  250.1241  250.1517  249.8877  249.7248  249.5881  249.7216
249.9171
  249.6543  250.0055  249.8429  250.1521  250.2695  249.6712
249.7268
  249.7039  249.8983  249.9581  249.7078  250.0835  249.9869
250.4180
  250.2456  250.1650  249.9261  250.1991  249.9344  250.0130
250.0691
  250.0374  249.9963  250.2695  250.2737  250.0920  250.0398
249.7965

...

  250.1417  249.8015
  249.9628  249.4425
  250.1063  249.8175

The parallel method is executed in 8.49 seconds.
Parallel pool using the 'redraider R2020b' profile is shutting
down.
```

Check if Job is Running in Parallel

It is always helpful to know whether your job is running across multiple cores/nodes or not for debugging purposes. There are two easy steps you can follow to confirm whether your job is running the way you expect it to be.

Step 1: Type "`ssh <node_name>`" to go into the node that your job got allocated to.

Step 2: Once you are in the cpu node, type "`top`" command to look at different running processes in that node.

Take the parallel job example above as an example. The 2 worker nodes allocated for the job are `cpu-26-[4,45]`. We can check one of the nodes to see if it is running in parallel, using node `cpu-26-4` for this example.

Step 3: Type "`ssh cpu-26-4`" to log into the node.

```
login-20-25:/matlabTest/matlabParGuide$ ssh cpu-26-4
```

Step 4: Once you are in, type "`top`" command.

```
login-20-25:/matlabTest/matlabParGuide$ ssh cpu-26-4
Warning: Permanently added 'cpu-26-4,10.100.26.4' (ECDSA) to the list of known hosts.
cpu-26-4:$
```

The command will display all the running processes in that node.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
76058 test      20   0 11.267g 904696 178180 S 106.0  0.5   0:37.59 MATLAB
76071 test      20   0 8416004 873072 178204 S 100.7  0.4   0:33.26 MATLAB
76073 test      20   0 8416004 862220 178160 S 100.7  0.4   0:33.19 MATLAB
76075 test      20   0 8416004 927072 178124 S 100.7  0.5   0:33.31 MATLAB
76063 test      20   0 8416004 875560 178140 S 100.3  0.4   0:33.35 MATLAB
76065 test      20   0 8405236 918224 178220 S 100.3  0.5   0:33.34 MATLAB
76072 test      20   0 8416004 868812 178208 S 100.3  0.4   0:33.65 MATLAB
76074 test      20   0 8416004 859592 178176 S 100.3  0.4   0:32.91 MATLAB
76079 test      20   0 8416004 924860 178252 S 100.3  0.5   0:33.54 MATLAB
76084 test      20   0 8416004 873712 178204 S 100.3  0.4   0:33.73 MATLAB
76085 test      20   0 8416004 927696 178184 S 100.3  0.5   0:32.66 MATLAB
76086 test      20   0 8404212 926300 178212 S 100.3  0.5   0:33.11 MATLAB
76087 test      20   0 8416004 867972 178288 S 100.3  0.4   0:33.41 MATLAB
76088 test      20   0 8416004 927096 178152 S 100.3  0.5   0:33.35 MATLAB
76091 test      20   0 8404212 865000 178172 S 100.3  0.4   0:32.96 MATLAB
76061 test      20   0 8416004 874596 178112 S 100.0  0.4   0:32.91 MATLAB
76067 test      20   0 8404212 899972 178196 S 100.0  0.5   0:33.37 MATLAB
```

You can see that all the cores are currently running MATLAB commands, which indicates they are running in parallel.

You can repeat the same steps for many other nodes you are running parallel on to check if they are all running across multiple cores/nodes.

Memory Request Suggestions for MATLAB

Our Quanah partition consists of 467 nodes, 36 cores per node, 192GB of memory per node, which thus means 5.33GB per core (or per cpu) by default.

Our Nocona partition consists of 240 CPU nodes, 128 cores per node, 512GB of memory per node, which thus means 4GB per core (or per cpu) by default.

When specifying memory request in SLURM, keep in mind that the option "`--mem`" requests total amount of memory for your job and "`--mem-per-cpu`" requests memory for each core (or each cpu) in your job.

When specifying node or core request in SLURM, keep in mind that the option "`--node`" requests total number of nodes, "`--ntasks`" requests total number of cores, and "`--ntasks-per-node`" requests a number of cores per node.

In **SERIAL** jobs, users who choose to run their job on a single core are still able to request extra memory if needed. This means that the limit for memory in your serial jobs does not have to be less than 4GB per core as per the Nocona tutorial above. If users do not specify memory options in their submission script, SLURM will assign the default memory per core, which is 4GB in Nocona. We first suggest that you run your serial job with the default memory, and if the job does not finish due to memory limit, then you can choose to request more. Users can request more memory for their serial jobs as long as the limit does not exceed the total amount of memory available in a single node, meaning it cannot exceed 512GB for Nocona partition. Therefore, users can use the "`--mem-per-cpu`" or "`--mem`" option (both do not make a difference in a one-core job) to specify the total memory for their serial job as long as the limit holds. For instance, a user can request one core and 7GB of memory for his MATLAB job and the SLURM scheduler will still allow it.

Due to the nature of MATLAB being very memory intensive, especially with jobs that compute large matrices, there might be some jobs that fail due to memory limit. If that is the case, we recommend the users to reduce the size of their matrices to allow the job to proceed.

Interactive job

Open MATLAB GUI

NOTE: Make sure you have **Xming** or **MobaXterm** set up if you choose to create an interactive session. Please refer to the prerequisites section in our table contents for user guides on how to set up these softwares.

To take advantage of MATLAB's graphical interface, you can create an interactive session in SLURM to open the software and interact with it following the instructions below.

Step 1. Log onto RedRaider (login.hpcc.ttu.edu) using your eRaider credentials. Please refer to "Connecting to HPCC Systems" to learn how to connect to our system.

Step 2. Request an interactive session using the "interactive" command

```
interactive -p nocona -c 1
```

The command above requests one core (`-c 1`) to be allocated in your Nocona partition (`-p nocona`) for your interactive session. You will be allocated a resource for your interactive job based on your request like the picture below. (Image from Quanah)

```
login-20-25:$ interactive -p nocona -c 1
[CPUs=1 NNodes=1 Name=INTERACTIVE Account=default Partition=nocona X11=YES]

salloc: Pending job allocation 2026657
salloc: job 2026657 queued and waiting for resources
salloc: job 2026657 has been allocated resources
salloc: Granted job allocation 2026657
salloc: Waiting for resource configuration
salloc: Nodes cpu-24-17 are ready for job
cpu-24-17:$
```

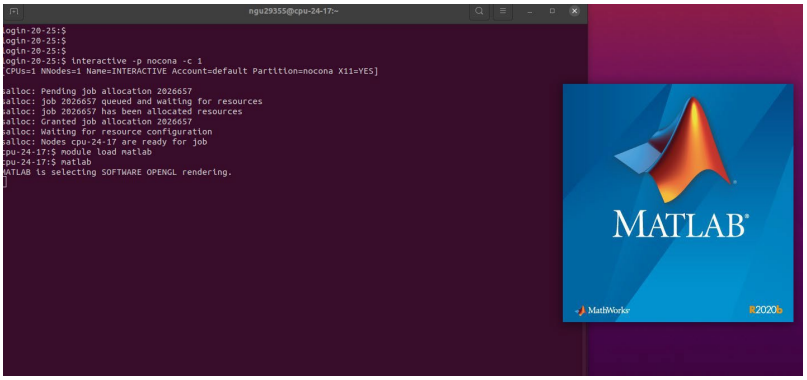
Step 3. Load MATLAB modules by typing in the following command

```
module load matlab
```

Step 4. Open MATLAB using the following command

```
matlab
```

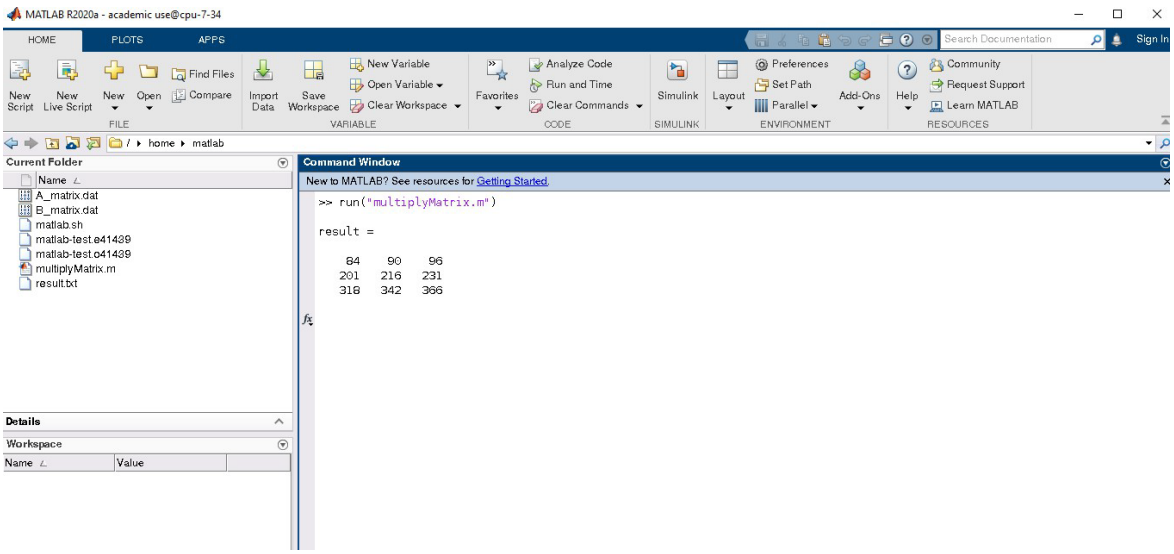
The picture below indicates that the MATLAB software is opening on your screen.



After MATLAB is open, you can test by running the MATLAB file you have available in your "matlab" folder by typing this on your MATLAB command line

```
>> run("multiplyMatrix.m")
```

Make sure that you are in the same folder where the file "multipleMatrix.m" is located. The picture below shows the result of running the file in your GUI MATLAB command line.



Nocona Parallel Job Submission on MATLAB Terminal using Matlab Parallel Server

Typically, we do not recommend running parallel jobs directly on MATLAB terminal without submitting a batch job because this can put pressure on the nodes and cause your job to fail if they are not scheduled correctly. Therefore, submitting batch jobs is still recommended for this purpose, but you can do this directly on the Matlab UI command line if that is the method you prefer.

Example 1: Regular Script

You can first create a script called “parallel_job” by typing in the following.

```
edit parallel_job
```

Make sure to type in the script the following code.

```
%Start timing
tic

%Calculates spectral radius of each matrix and displays results
n = 100;
A = 500;
a = zeros(n);
parfor i = 1:n*n
    a(i) = max(abs(eig(rand(A))));
end
disp(a)
```

```
%End timing
time = toc;
```

Now save the file. After saving, go back to your Terminal on MATLAB and set up the properties. You can refer to "Set up Arguments for Parallel Job" for more optional settings.

```
>> c = parcluster;

>> c.AdditionalProperties.Partition = 'nocona';

>> c.AdditionalProperties.NumberOfNodes = 2;

>> c.saveProfile;
```

Since you already configured the settings in the MATLAB Terminal above, there is no need to specify the settings again inside your "parallel_job" script like the all the other tutorials. Now, you can start submitting a batch job requesting 255 pool workers as an example, since batch will use an additional worker that makes the total workers 256. The "CurrentFolder" should be set to your current working directory or wherever your script is. Setting "AutoAddClientPath" to false will prevent MATLAB from adding folders to the top of your search path.

```
>> job = batch(c,'parallel_job', 'Pool', 255,
'CurrentFolder', '.' , 'AutoAddClientPath', false)
```

The command above will submit your job to the SLURM scheduler and wait for allocated resources just like any other regular job submission. You can check the state using this command.x

```
>> job.State
```

Once finished, load the job variable 'a' in the script and view outputs of that variable using the following command.

```
>> load(job,'a')

>> a
```

Make sure to delete the job when you no longer need it.

```
>> job.delete
```

Example 2: Function Script

You can first create a script called “parallel_example” by typing in the following.

```
edit parallel_example
```

Make sure to type in the script the following function code. This script has a function that takes in an argument as the number of iterations and outputs the time it takes to finish the iterations.

```
function t = parallel_example(iter)

if nargin==0, iter = 8; end

disp('Start iterations')

t0 = tic;
parfor i = 1:iter
     A(i) = i;
     pause(2)
end
t = toc(t0);

disp('Completed')
```

Now save the file. After saving, go back to your Terminal on MATLAB and set up the properties. You can refer to “Set up Arguments for Parallel Jobs” for more optional settings.

```
>> c = parcluster;

>> c.AdditionalProperties.Partition = 'nocona';

>> c.AdditionalProperties.NumberOfNodes = 1;

>> c.saveProfile;
```

Since you already configured the settings in the MATLAB Terminal above, there is no need to specify the settings again inside your “parallel_example” script like the all the other tutorials. @parallel_example refers to the name of your function script. The number of outputs for this function is 1, and the number of inputs is 16. You will submit a batch job requesting 4 pool workers, and since batch will use an additional worker, this will make the total workers 5. The “CurrentFolder” should be set to your current working directory or wherever your function script is. Setting “AutoAddClientPath” to false will prevent MATLAB from adding folders to the top of your search path.

```
>> job = batch(c, @parallel_example, 1, {16},'Pool', 4,
'CurrentFolder', '.' , 'AutoAddClientPath', false)
```

The command above will submit your job to the SLURM scheduler and wait for allocated resources just like any other regular job submission. You can check the state using this command.

```
>> job.State
```

Once finished, you can use this command to view the output.

```
>> job.fetchOutputs{:}
```

Make sure to delete the job when you no longer need it.

```
>> job.delete
```

NOTE: `fetchOutputs` is specifically used to view function outputs, and `load` is typically used to view variables outputs in a regular MATLAB script.

Additional job query commands

If your job is still running and you want to see the ID of the job to check back later, use this command before exiting out of MATLAB.

```
>> id = job.ID
id =
     4
>> clear job
```

Now that you come back to MATLAB, retrieve the job you ran earlier.

```
>> c = parcluster
>> job = c.findJob('ID',4)
```

MATLAB Tutorials for Other Clusters

The above guide goes through general steps to run MATLAB on Nocona partition. However, running MATLAB on Quanah or Matador partition should follow the same ideas, except for some differences in the partition name, memory, and number of nodes request. The list below shows you the path to some examples on different clusters for references. You can just use the given command to copy the tutorial to your home folder to test.

Quanah Job Submission:

```
cp -r /lustre/work/examples/quanah/matlab ~/matlab
```

Quanah Multi-node Parallel Job Submission using Matlab Parallel Server:

```
cp -r /lustre/work/examples/quanah/matlabPar ~/matlabPar
```

Matador Job Submission:

```
cp -r /lustre/work/examples/matador/matlab ~/matlab
```

Matador Multi-node Parallel Job Submission using Matlab Parallel Server:

```
cp -r /lustre/work/examples/matador/matlabPar ~/matlabPar
```