Matlab

Introduction

Matlab is available in versions R2015a and R2015b. There are always two variants of the release:

- Non commercial or so called EDU variant, which can be used for common research and educational purposes.
- Commercial or so called COM variant, which can used also for commercial activities. The licenses for commercial variant are much more expensive, so usually the commercial variant has only subset of features compared to the EDU available.

To load the latest version of Matlab load the module

```
$ module load MATLAB
```

By default the EDU variant is marked as default. If you need other version or variant, load the particular version. To obtain the list of available versions use

```
$ module avail MATLAB
```

If you need to use the Matlab GUI to prepare your Matlab programs, you can use Matlab directly on the login nodes. But for all computations use Matlab on the compute nodes via PBS Pro scheduler.

If you require the Matlab GUI, please follow the general informations about running graphical applications.

Matlab GUI is quite slow using the X forwarding built in the PBS (qsub -X), so using X11 display redirection either via SSH or directly by xauth (please see the "GUI Applications on Compute Nodes over VNC" part here) is recommended.

To run Matlab with GUI, use

```
$ matlab
```

To run Matlab in text mode, without the Matlab Desktop GUI environment, use

```
$ matlab -nodesktop -nosplash
```

plots, images, etc… will be still available.

Running parallel Matlab using Distributed Computing Toolbox / Engine

Distributed toolbox is available only for the EDU variant

The MPIEXEC mode available in previous versions is no longer available in MATLAB 2015. Also, the programming interface has changed. Refer to Release Notes.

Delete previously used file mpiLibConf.m, we have observed crashes when using Intel MPI.

To use Distributed Computing, you first need to setup a parallel profile. We have provided the profile for you, you can either import it in MATLAB command line:

```
> parallel.importProfile('/apps/all/MATLAB/2015a-EDU/SalomonPBSPro.settings')

ans =

SalomonPBSPro
```

Or in the GUI, go to tab HOME -> Parallel -> Manage Cluster Profiles…, click Import and navigate to :

/apps/all/MATLAB/2015a-EDU/SalomonPBSPro.settings

With the new mode, MATLAB itself launches the workers via PBS, so you can either use interactive mode or a batch mode on one node, but the actual parallel processing will be done in a separate job started by MATLAB itself. Alternatively, you can use "local" mode to run parallel code on just a single node.

The profile is confusingly named Salomon, but you can use it also on Anselm.

Parallel Matlab interactive session

Following example shows how to start interactive session with support for Matlab GUI. For more information about GUI based applications on Anselm see this page.

```
$ xhost +
$ qsub -I -v DISPLAY=$(uname -n):$(echo $DISPLAY | cut -d ':' -f 2) -A NONE-0-0 -q qexp -l select=
-l feature__matlab__MATLAB=1
```

This qsub command example shows how to run Matlab on a single node.

The second part of the command shows how to request all necessary licenses. In this case 1 Matlab-EDU license and 48 Distributed Computing Engines licenses.

Once the access to compute nodes is granted by PBS, user can load following modules and start Matlab:

```
r1i0n17$ module load MATLAB/2015b-EDU
r1i0n17$ matlab &
```

Parallel Matlab batch job in Local mode

To run matlab in batch mode, write an matlab script, then write a bash jobscript and execute via the qsub command. By default, matlab will execute one matlab worker instance per allocated core.

```
#!/bin/bash
#PBS -A PROJECT ID
#PBS -q qprod
#PBS -l select=1:ncpus=16:mpiprocs=16:ompthreads=1

# change to shared scratch directory
SCR=/scratch/work/user/$USER/$PBS_JOBID
mkdir -p $SCR ; cd $SCR || exit

# copy input file to scratch
cp $PBS_O_WORKDIR/matlabcode.m .

# load modules
module load MATLAB/2015a-EDU

# execute the calculation
matlab -nodisplay -r matlabcode > output.out

# copy output file to home
cp output.out $PBS_O_WORKDIR/.
```

This script may be submitted directly to the PBS workload manager via the qsub command. The inputs and matlab script are in matlabcode.m file, outputs in output.out file. Note the missing .m extension in the matlab -r matlabcodefile call, **the .m must not be included**. Note that the **shared /scratch must be used**. Further, it is **important to include quit** statement at the end of the matlabcode.m script.

Submit the jobscript using qsub

```
$ qsub ./jobscript
```

Parallel Matlab Local mode program example

The last part of the configuration is done directly in the user Matlab script before Distributed Computing Toolbox is started.

```
cluster = parcluster('local')
```

This script creates scheduler object “cluster” of type “local” that starts workers locally.

Please note: Every Matlab script that needs to initialize/use matlabpool has to contain these three lines prior to calling parpool(sched, …) function.

The last step is to start matlabpool with "cluster" object and correct number of workers. We have 24 cores per node, so we start 24 workers.

```
parpool(cluster,16);


... parallel code ...


parpool close
```

The complete example showing how to use Distributed Computing Toolbox in local mode is shown here.

```
cluster = parcluster('local');
cluster

parpool(cluster,24);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
             % T and W are both codistributed arrays here.
end
T;
whos         % T and W are both distributed arrays here.

parpool close
quit
```

You can copy and paste the example in a .m file and execute. Note that the parpool size should correspond to **total number of cores** available on allocated nodes.

Parallel Matlab Batch job using PBS mode (workers spawned in a separate job)

This mode uses PBS scheduler to launch the parallel pool. It uses the SalomonPBSPro profile that needs to be imported to Cluster Manager, as mentioned before. This methodod uses MATLAB's PBS Scheduler interface - it spawns the workers in a separate job submitted by MATLAB using qsub.

This is an example of m-script using PBS mode:

```
cluster = parcluster('SalomonPBSPro');
set(cluster, 'SubmitArguments', '-A OPEN-0-0');
set(cluster, 'ResourceTemplate', '-q qprod -l select=10:ncpus=16');
set(cluster, 'NumWorkers', 160);

pool = parpool(cluster, 160);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
             % T and W are both codistributed arrays here.
end
whos         % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Note that we first construct a cluster object using the imported profile, then set some important options, namely : SubmitArguments, where you need to specify accounting id, and ResourceTemplate, where you need to specify number of nodes to run the job.

You can start this script using batch mode the same way as in Local mode example.

Parallel Matlab Batch with direct launch (workers spawned within the existing job)

This method is a "hack" invented by us to emulate the mpiexec functionality found in previous MATLAB versions. We leverage the MATLAB Generic

Scheduler interface, but instead of submitting the workers to PBS, we launch the workers directly within the running job, thus we avoid the issues with master script and workers running in separate jobs (issues with license not available, waiting for the worker's job to spawn etc.)

Please note that this method is experimental.

For this method, you need to use SalomonDirect profile, import it using the same way as SalomonPBSPro

This is an example of m-script using direct mode:

```
parallel.importProfile('/apps/all/MATLAB/2015a-EDU/SalomonDirect.settings')
cluster = parcluster('SalomonDirect');
set(cluster, 'NumWorkers', 48);

pool = parpool(cluster, 48);

n=2000;

W = rand(n,n);
W = distributed(W);
x = (1:n)';
x = distributed(x);
spmd
[~, name] = system('hostname')

    T = W*x; % Calculation performed on labs, in parallel.
             % T and W are both codistributed arrays here.
end
whos         % T and W are both distributed arrays here.

% shut down parallel pool
delete(pool)
```

Non-interactive Session and Licenses

If you want to run batch jobs with Matlab, be sure to request appropriate license features with the PBS Pro scheduler, at least the " -l __feature__matlab__MATLAB=1" for EDU variant of Matlab. More information about how to check the license features states and how to request them with PBS Pro, please look here.

In case of non-interactive session please read the following information on how to modify the qsub command to test for available licenses prior getting the resource allocation.

Matlab Distributed Computing Engines start up time

Starting Matlab workers is an expensive process that requires certain amount of time. For your information please see the following table:

compute nodes	number of workers	start-up time[s]
16	384	831
8	192	807
4	96	483
2	48	16

MATLAB on UV2000

UV2000 machine available in queue "qfat" can be used for MATLAB computations. This is a SMP NUMA machine with large amount of RAM, which can be beneficial for certain types of MATLAB jobs. CPU cores are allocated in chunks of 8 for this machine.

You can use MATLAB on UV2000 in two parallel modes :

Threaded mode

Since this is a SMP machine, you can completely avoid using Parallel Toolbox and use only MATLAB's threading. MATLAB will automatically detect the number of cores you have allocated and will set maxNumCompThreads accordingly and certain operations, such as fft, , eig, svd, etc. will be automatically run in threads. The advantage of this mode is that you don't need to modify your existing sequential codes.

Local cluster mode

You can also use Parallel Toolbox on UV2000. Use local cluster mode, "SalomonPBSPro" profile will not work.