Running Matlab jobs

Running serial Matlab jobs in single thread

Matlab jobs can be run in interactive session or the batch mode. In either way, load the module as it's needed:

```
$ module purge
$ module load matlab/2016b
$ matlab
```

However there are caveats worth being mentioned here.

- Matlab exit code
  By default, Matlab returns 0 even when it stops due to an error. The batch system (and Unix generally) interprets a 0 exit code as "completed successfully", any other return code means "failed with error". This could cause problems e.g. if job dependencies are used for a workflow. To discover failed Matlab jobs, one workaround is to wrap the Matlab code in a "try ... catch" block.

- Preference directory

In the example job script below one CPU core is requested for the task (by default one core per task).
#!/bin/bash
#SBATCH --job-name=Matlab
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=2GB
#SBATCH --time=01:00:00

module purge
module load matlab/2016b

if [ "${SLURM_JOBTMP} == " ]; then
    export SLURM_JOBTMP=/state/partition1/${USER}/$$
    mkdir -p ${SLURM_JOBTMP}
fi

export MATLAB_PREFDIR=$(mktemp -d $SLURM_JOBTMP/matlab-XXXXXX)

cd /scratch/${USER}/test_slurm/matlab/basic

echo "Job starts: $(date)"
echo "Hostname: $(hostname)"

while read line
do
cat<<EOF | matlab -nodisplay
try
    basic
catch err
    fprintf('\nTime: %s
', datestr(datetime('now')));
    fprintf('Matlab error: %s\n', err.message);
    exit(1);
end
EOF

matlab_status=$?
echo "Job ends: $(date)"
rm -rf $MATLAB_PREFDIR
exit $matlab_status

In the "try ... catch" block, the "basic" represents a Matlab script file named "basic.m" in the same directory. Its content is simply 1+2 for demo purpose.

Running parallel Matlab jobs on one node

Because of a license limitation on the NYU HPC clusters, running parallel Matlab jobs is possible only on one node. We can not do that across multiple compute nodes.

The following example is taken from the directory /share/apps/examples/matlab/parfor-prefdir. There are two things worth noting here:

- MATLAB_PREFDIR
   This is similar as the serial Matlab example above. It should be set in every Matlab submission, so that jobs from the same user running on the same node will not step on each other, i.e. they will write (intermediate) data out to their own dedicated directories.
- parpool('local', ${SLURM_CPUS_PER_TASK})
   This creates and returns a parallel pool on the local node where the job lands, with the specified number of workers.
#!/bin/bash
#
#SBATCH --job-name=Matlab
#SBATCH --nodes=1
#SBATCH --cpus-per-task=28
#SBATCH --mem=15GB
#SBATCH --time=12:00:00

module purge
module load matlab/2016b

cd /scratch/$USER/examples/matlab/parfor-prefdir

if [ "$SLURM_JOBTMP" == "" ]; then
  export SLURM_JOBTMP=/state/partition1/$USER/$$
mkdir -p $SLURM_JOBTMP
fi

export MATLAB_PREFDIR=$(mktemp -d $SLURM_JOBTMP/matlab-XXXX)

echo
echo "Hostname: $(hostname)"

echo
cat<<EOF | srun matlab -nodisplay
parpool('local', $SLURM_CPUS_PER_TASK)
pi_parallel
exit
EOF

rmdir $SLURM_JOBTMP/*

In the job script above, "pi_parallel" is a function defined in the matlab script file named "pi_parallel.m" in the same example directory as given above.

The following issue is noted for older version of Matlab for the Mercer cluster. It still exists on Prince - the program will hang after printing the first line (lineA) if you run this script,

#!/bin/bash -f
/share/apps/matlab/2016b/bin/matlab -nodisplay -r "
fprintf('lineA\n');
fprintf('lineB\n');
fprintf('lineC\n');
exit
"

This may also prevent you from executing any further Matlab commands. As an alternative, you can change your script to this format,
#!/bin/bash -f

cat <<EOF | /share/apps/matlab/2016b/bin/matlab -nodisplay
fprintf('lineA\n');
fprintf('lineB\n');
fprintf('lineC\n');
exit
EOF

which can finish correctly with an output,

lineA
lineB
lineC