Parallel Computing using MATLAB Distributed Compute Server

ZRORRO HPC
Goals of the session

• Overview of parallel MATLAB
  - Why parallel MATLAB?

• Multiprocessing in MATLAB

• Parallel MATLAB using the Parallel Computing Toolbox (PCT)
  - Running a serial job
  - Running interactive parallel job
  - Running a batch job on the OSC cluster
Parallel Computing

- **Goals:**
  - Speed up computations by using multiple processors
  - Utilize more memory than available on single machine

- **How?**
  - Using MPI: Message Passing Interface, a library that is used to exchange data and control information between the processors.

- **Used in distributed memory environments**
  - Using OpenMP: A set of compiler directives that is used to run threads in parallel in a shared memory environment

- **Reality**
  - Parallel programming using C/C++/FORTRAN and MPI is hard
  - Creating parallel code in C/FORTRAN and MPI takes a long time
Why Parallel MATLAB

- MATLAB is widely used for developing/prototyping algorithms

- The High Level Language and Integrated
  - Development/Visualization environment leads to productive code development

- By parallelizing MATLAB code
  - Algorithm can be run with different/larger data sets
  - Algorithm can be run with larger parameter sweeps
  - Compute times may be reduced
Multiprocessing in MATLAB

- MATLAB R2011a supports implicit and explicit multi-processing

- Implicit multi-processing
  - Built-in multithreading
  - Speeds up many linear algebra routines, matrix operations
  - Leverages multiple cores on processor

- Explicit multi-processing
  - Parallel computing using the Parallel Computing Toolbox and MATLAB Distributed Computing Server
  - Leverages multiple processors on clusters
Implicit Multiprocessing: Multithreading in MATLAB

- MATLAB runs computations on multiple threads on your machine
- No changes to MATLAB code required
- Users can change behavior via preferences
- Maximum gain in element-wise operations and BLAS routines
Explicit Multiprocessing:
The Parallel Computing Toolbox

- Explicit multiprocessing is enabled through the use of the following two products
  
  - The Parallel Computing Toolbox (PCT)
  
  - MATLAB Distributed Computing Server (MDCS)
The Parallel Computing Toolbox

- Provides parallel constructs in the MATLAB language, such as parallel for loops, distributed arrays and message passing

- Enables rapid prototyping of parallel code through an interactive parallel MATLAB session

- Provides the ability to scale the problem by harnessing resources in a remote cluster
The Parallel Computing Toolbox

Language enhancements include:

• Ability to create and use distributed arrays

• Over 150 parallel functions for use on distributed arrays
  - cos, sin, log, find, isempty, etc.

• ScaLAPACK based parallel linear algebra routines
  - svd, lu

• Global, collective operations such as global addition, reduction, etc.

• Explicit, fine grained parallelism via MPI functions
The Parallel Computing Toolbox

- The Parallel Computing Toolbox supports the following types of schedulers for job submission
  - Local scheduler
    - Can run up to 4 Workers simultaneously
    - Useful for debugging parallel code locally
  - Job Manager
  - Supported Third-Party scheduler (PBS, LSF, SGE)
  - Generic Scheduler
    - Generic interface that allows use with a third-party scheduler

- Additionally the PCT supports the use of configurations
  - Configurations are a convenient way to store scheduler parameters
MATLAB Distributed Computing Server - MDCS

- The MATLAB Distributed Computing Server enables scaling of parallel MATLAB code on clusters
- It includes a basic scheduler and also supports LSF, PBS and TORQUE
Setup

• Start interactive MATLAB session on AU-HPC
  mathlab

* Note include -Y flag in your ssh command to set X11 forwarding

• Start MATLAB in batch mode
  mathlab -nodisplay

• Submit MATLAB job
  mathlab -nodisplay <myjob.m> saveMyjob.out &
LSF - Running Matlab in batch

- To submit a batch Matlab job, first prepare a file with your Matlab commands, say “program_file.m”. Then issue the commands:

```bash
bsub -q normal matlab -nodisplay -nojvm -nosplash -r program_file -logfile output_file.txt
```

*NB: if you intend to use java programs do not include the flag -nojvm.*

Note that the suffix ".m" is omitted from the command file name. This submits a batch job to the batch queue taking input from the file program_file.m, and placing text output in output_file.txt.
Sample LSF script
Matlab Job

#!/bin/bash

# enable your environment, which will use .bashrc configuration in your home directory
#BSUB -L /bin/bash

# the name of your job showing on the queue system
#BSUB -J MatlabJob

# the following BSUB line specify the queue that you will use,
#BSUB -q normal

# the system output and error message output, %J will show as your jobID
#BSUB -o %J.out
#BSUB -e %J.err

#the CPU number that you will collect (Attention: each node has 2 CPU)
#BSUB -n 1

#when job finish that you will get email notification
#BSUB -u user@american.edu
#BSUB -N

# your matlab code
matlab -nodisplay -r myplot

#enter your working directory
cd /home/username/matlab

bsub < matlab.lsf
Running on a cluster using a Generic Scheduler

• Creating a job

```matlab
job = createParallelJob(sched);
set(job, 'MaximumNumberOfWorkers', 4);
set(job, 'MinimumNumberOfWorkers', 4);
```

• Some important properties of jobs
  - FileDependencies: List of user m-files that the job needs
  - PathDependencies: List of directories to be added to the MATLAB path

• Output is retrieved using the function
  ```matlab
  getAllOutputArguments
  ```
Running on a cluster using a Generic Scheduler

Creating tasks

task = createParallelJob(job);
set(task, 'CaptureCommandWindowOutput', 1);

- Some important properties of Tasks
  - CommandWindowOutput
    - Returns the messages printed to the screen
  - Error
    - Returns the error stack (if an error occurs on the Worker)
Running on a cluster using a Generic Scheduler

• Finally, the job is submitted using the command submit(job)