Parallel Computing with Matlab
UVACSE Short Course

E Hall[1]

[1]University of Virginia Advanced Computing Services and Engagement
uvacse@virginia.edu

October 6, 2014

Outline

Starting and Configuring NX Client

Once logged into `fir.itc.virginia.edu` through NX

- Open a terminal from Applications/Accessories/Terminal menu
  - Select and right-click on Terminal to add to launcher
- Create a `Matlab` directory with `mkdir` command
- Start web browser from icon at top of desktop

Download Short Course Examples

Download the short-course materials from
http://www.uvacse.virginia.edu/software/Matlab-at-uva/

Follow the links,
→High Performance Computing
→Parallel Computing Toolbox
→Parallel Computing Short Course
and download 3 files to `Matlab` directory you create with `mkdir` command

- ClassExamples_Fa14.zip
- matlab_parallel_Fa14.pdf

Solving Big Technical Problems

Computationally intensive, long-running codes

- Run tasks in parallel on many processors
- Task parallel

Large Data sets

- Load data across multiple machines that work in parallel
- Data parallel

Parallel Computing Toolbox Features

Support for **data-parallel** and **task-parallel** application development

Ability to annotate code segments

- **`parfor`** (parallel for-loops) for task-parallel algorithms
- **`spmd`** (single program multiple data) for data-parallel algorithms

These **high-level programming constructs** convert serial Matlab code to run in parallel on several workers

The parallel programming constructs function even in the absence of workers so that a single version of code can run for both serial and parallel execution

Programming Parallel Applications in Matlab

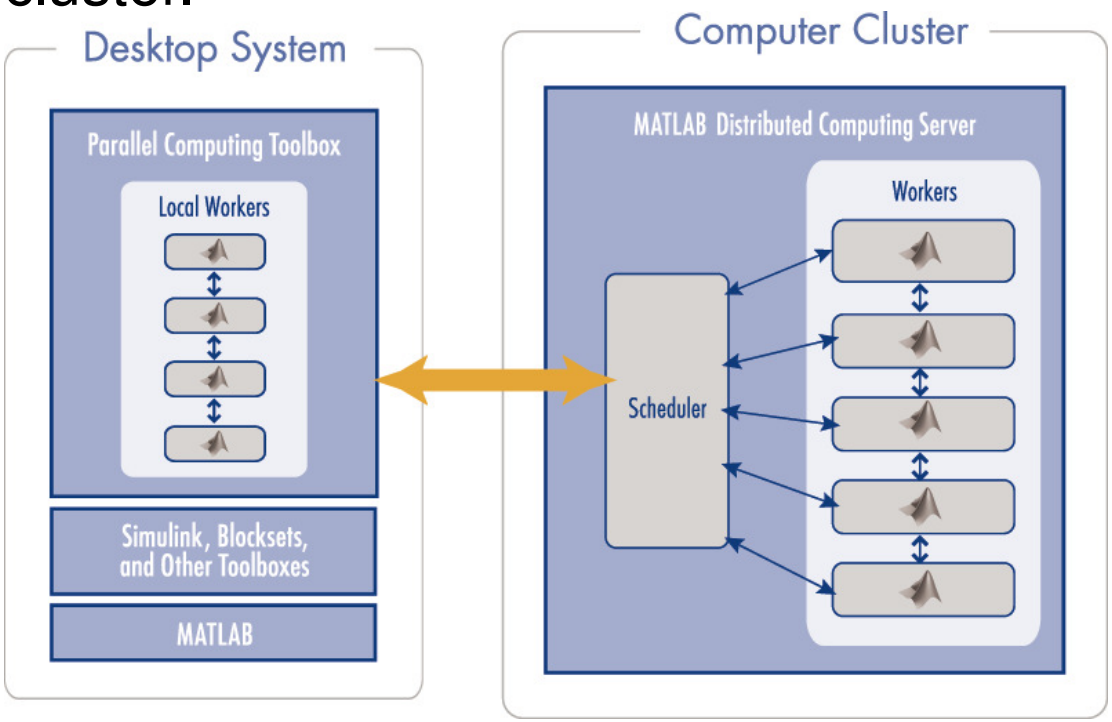
Simplifies parallel code development by abstracting away complexity of managing coordination and distribution of computations and data between a Matlab client and workers.

Run 12 workers locally on a multicore desktop

Integration with Matlab Distributed Computing Server for **cluster-based applications** that can use up to **256 workers**

Matlab Parallel Workflow

The toolbox enables application prototyping on the desktop with up to eight local workers (left), and with Matlab Distributed Computing Server (right), applications can be scaled to multiple computers on a cluster.



UseParallel for Optimization Algorithms

The Optimization Toolbox solvers **`fmincon`**, **`fgoalattain`**, and **`fminimax`** can automatically distribute the numerical estimation of gradients of objective functions and nonlinear constraint functions to multiple processors.

- Parallel computing is enabled with **`matlabpool`**, a Parallel Computing Toolbox function
- The option **`UseParallel`** is set to **`'Always'`**. The default value of this option is 'Never'.

UseParallel for Optimization Algorithms

Optimization Toolbox

- **`fmincon`**
- **`fminimax`**
- **`fgoalattain`**

Genetic Algorithm and Direct Search Toolbox

- **`ga`**
- **`gamultiobj`**
- **`patternsearch`**

UseParallel for Optimization Algorithms

Caveats:

- The built-in parallel support in Optimization Toolbox is **beneficial for problems that have objective/constraint functions with execution times greater than network overhead** associated with distributing computations across multiple workers
- However, **parallelizing the objective/constraint function itself can be a better approach** if it is the most expensive step in the optimization problem and can be accelerated by parallelizing the objective function

Documentation:

- Improving Optimization Performance with Parallel Computing
- Minimizing an Expensive Optimization Problem Using Parallel Computing Toolbox

Example Code: Parallel Optimization with `fmincon`

```
% Serial optimization using fmincon
startPoint = [1 -2 0 5];
options = optimset('Display','iter','Algorithm','active-set');
startTime = tic;
fmincon(@expensive_objfun,startPoint,[],[],[],[],[],[], ...
    @expensive_confun,options);
time_fmincon_sequential = toc(startTime);
fprintf('Serial FMINCON optimization takes %g seconds.\n', ...
    time_fmincon_sequential);


% Parallel optimization using fmincon
matlabpool open 2  % create 2 Matlab workers

options = optimset(options,'UseParallel','always');
startTime = tic;
fmincon(@expensive_objfun,startPoint,[],[],[],[],[],[],...
    @expensive_confun,options);
time_fmincon_parallel = toc(startTime);
fprintf('Parallel FMINCON optimization takes %g seconds.\n',...
    time_fmincon_parallel);
```

Task-parallelism using `parfor`

Parallel `for`-Loops - `parfor`

- Code Annotation

```
parfor i = 1 : n
    % do something with i
end
```

- Mix task-parallel and serial code in the same function
- Run loops on a pool of Matlab resources
- Iterations must be order-independent

Task-parallelism using `parfor`

```
function  [a]=pcalc(nloop)
% Example using the parfor construct
% to calculate the maximum eignevalue
% of a random 300x300 matrix nloops times

N=nloop;
a=zeros(N, 1);
% TIME CONSUMING LOOP
tic;
parfor i=1:N
    a(i)=iFunctionTakesLongTime();
end
toc
end

function max_eig=iFunctionTakesLongTime()
% Computation intensive calculation dependent
% on matrix size
max_eig=max(abs(eig(rand(300))));
end
```

Task-parallelism using `parfor`

```
>> % Run pcalc() as serial code
>> pcalc(100);
Elapsed time is 8.702374 seconds.
>> pcalc(100);
Elapsed time is 8.790408 seconds.

>> % Run pcalc() as parallel code
>> matlabpool open 2
Starting matlabpool using the 'local' profile
>> pcalc(100);
Elapsed time is 6.039350 seconds.
>> pcalc(100);
Elapsed time is 5.733905 seconds.
```

Task-parallelism using `parfor`

Parallel for-loops (`parfor`) automatically distribute a set of independent tasks over a set of workers.

Work distribution across worker is dynamic for load balancing.

The `matlabpool` command sets up the interactive execution environment for parallel constructs such as `parfor` or `spmd`

The running code automatically detects the presence of workers and reverts to serial behavior if none are present.

Using the `batch` command allows parallel code to run in batch mode across the compute nodes of a cluster.

Applications: Monte Carlo simulations, Parameter sweeps

Setting Up the Matlab Workers with `matlabpool`

The `matlabpool` command allocates a set of dedicated computational resources by reserving a number of Matlab workers for executing parallel Matlab code.

Parallel Computing Toolbox provides the ability to use up to 12 local workers on a multicore or multiprocessor computer using a single toolbox license.

Within this environment, `parfor` and `spmd` constructs can set up data and Matlab code exchange between a Matlab client.

`spmd` for Data-Parallel Processing

For Matlab algorithms that require large data set processing, the Parallel Computing Toolbox provides,

- distributed arrays, parallel functions
- the `spmd` keyword to annotate sections of your code for parallel execution on several workers.

These parallel constructs handle the inter-worker communication and coordinate parallel computations behind the scenes.

`spmd` for Data-Parallel Processing

Using distributed arrays, you can allocate matrices of any data type across all workers participating in the parallel computation.

Parallel functions let you perform mathematical operations such as indexing, matrix multiplication, decomposition, and transforms directly on distributed arrays.

The toolbox also provides more than150 parallel functions for distributed arrays, including linear algebra routines based on ScaLAPACK.

`spmd` for Data-Parallel Processing

Single Program Multiple Data - `spmd`

- Code Annotation

```
spmd
    % data parallel operation
end
```

- Single program
  - Runs simultaneously across all workers
  - Enables easy writing and debugging
- Multiple Data
  - Data spread across workers
- Runs serially if no workers are available

`spmd` for Data-Parallel Processing

Distributed Arrays

- Distributed arrays are special arrays that store segments of data on Matlab workers that are participating in a parallel computation.
  - can handle larger data sets than you can on a single Matlab session.
- You can construct distributed arrays in several ways:
  - Using constructor functions such as `rand`, `ones`, and `zeros`
  - Concatenating arrays with same name but different data on different labs
  - Distributing a large matrix

spmd for Data-Parallel Processing: Example Code

Solving a Large Linear System

```
function [ x2, errChk ] = solver_large1( N )
% This function is a simple test of a LU linear solver
% of Ax=b using a distributed A matrix. Since b is sum
% of columns, x should always be a vector of ones.
tic;
spmd
    dist=codistributor();
    A = rand(N, N, dist);
    b = sum(A, 2);

    % solve Ax=b
    x = mldivide(A,b);
    x2 = gather(x);
    % Check error
    errChk = normest(A * x - b);
end
toc
x2=x2{:};
errChk=errChk{:};
```

spmd for Data-Parallel Processing: Example

Numerical Estimation of Pi Using Message Passing

- Use the fact that

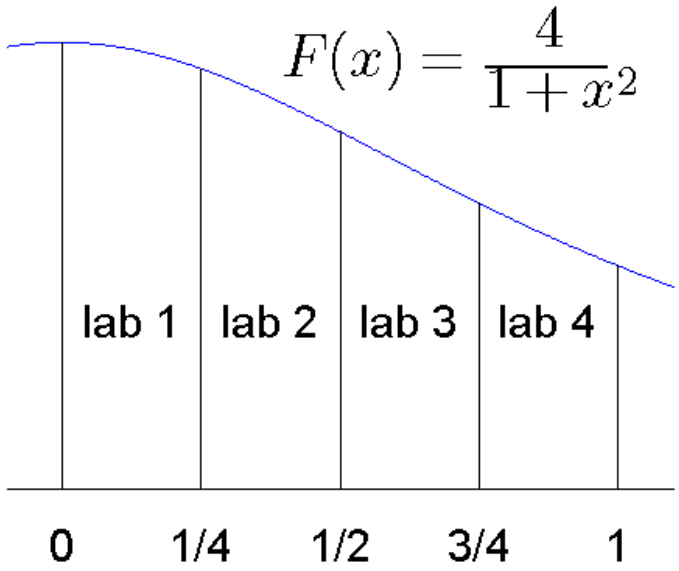
$$\int_0^1 \frac{4}{1+x^2} dx = 4(atan(1) - atan(0)) = \pi$$

  to approximate pi by approximating the integral on the left

- use the spmd keyword to mark the parallel blocks of code and the Matlab worker pool performs the calculations in parallel

`spmd` for Data-Parallel Processing: Example

Divide the work between four labs by having each lab calculate the integral of the function over a subinterval of [0, 1]



Define the variables `a` and `b` on each lab using the `labindex` so that the intervals [`a`, `b`] correspond to the subintervals above.

spmd for Data-Parallel Processing: Example Code

Parallel Esimation of Pi:

```
clear  % clear workspace
F = @(x) 4./(1 + x.^2);
matlabpool open 4
spmd
    % Define integration interval
    % Use implicit messaging using labindex
    a = (labindex - 1)/numlabs;
    b = labindex/numlabs;
    fprintf('Subinterval: [%-4g, %-4g]\n', a, b);

    % Use Matlab quadrature method to approximate integral
    myIntegral = quadl(F, a, b);
    fprintf('Subinterval: [%-4g, %-4g]   Integral: %4g\n', ...
            a, b, myIntegral);

    % We use the gplus function to add myIntegral across all
    % the labs and return the sum on all the labs.
    piApprox = gplus(myIntegral);

end
```

Matlab MPI for Message Passing

Use when a high degree of control over parallel algorithm is required

- High-level abstractions of MPI message-passing routines based on the MPI standard (MPICH2)
  - `labSendReceive`, `labBroadcast`, and others
- Send, receive, and broadcast any data type in Matlab including structures and cell arrays, without any special setup.
- Use any MPI implementation that is binary-compatible with MPICH-2

Message Passing for Data Parallel Algorithms

Message passing functions can be used with `spmd` statements

```
matlabpool open 4
spmd
    source = 1;
    destination = [2, 4];
    if labindex == source
        % Send data from source lab
        testData.rpm = 1000; % Set up structure
        testData.speed = 35;
        otherData = rand(1000); % A random array

        labSend(testData, destination);
        labSend(otherData, destination);
    elseif any(labindex == destination)
        % Receive on destination labs
        recvdata{1} = labReceive(source);
        recvdata{2} = labReceive(source);
    end
end
v=recvdata{2} % Data received in lab 2
v{1}
```

`pmode` for Interactive Parallel Computing

Parallel Command Window

- The Parallel Command Window provides an extension to the Matlab command window for executing data-parallel Matlab code directly on workers participating in the interactive parallel session
- Commands issued at the `pmode` prompt are executed immediately on all the labs and results are returned immediately to the client session.
- You can use distributed arrays, associated parallel functions, as well as message-passing functions in this mode.
- This tool facilitates the debugging process, as it allows you to watch the results and the interaction between labs at each step.

`pmode` for Interactive Parallel Computing

Interactive Prototyping / Development Using Parallel Command Window

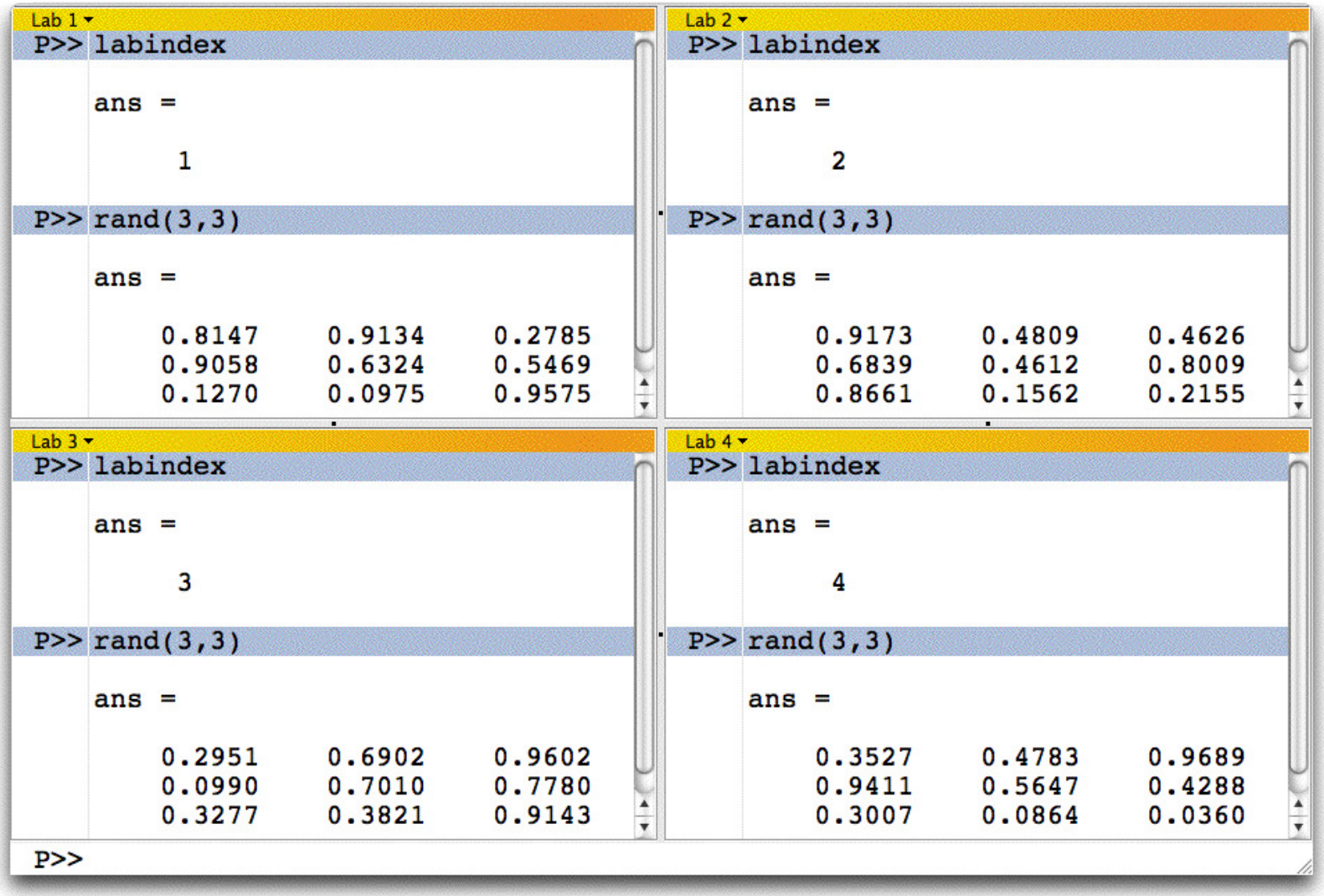
- Parallel Command Window similar to regular command window
- Execute commands and observe behavior on each worker of the cluster
- Can run up to 8 workers on Matlab desktop
- Workers are processes, not tied to cores or processors as such

pmode for Interactive Parallel Computing

Example Code

```
>>pmode start
P>>a=1
P>>labindex
P>>numlabs
P>> A=rand(2000, 2000, codistributor());
P>>whos
P>>size(localPart(A))
```

pmode for Interactive Parallel Computing



Scaling Up from the Desktop

Parallel Computing Toolbox provides the ability to use up to 12 local workers on a multicore or multiprocessor computer using a single toolbox license.

When used together with MATLAB Distributed Computing Server, you can scale up your application to use any number of workers running on any number of computers.

ITS Linux cluster has MDCS licenses for 256 workers.

Alternatively, you can run up to 12 workers on a single multi-core compute node of the cluster.

Using Parallel Configurations with PBS Pro

Parallel Configurations - Where and How the Code is Executed

Maintain named configurations

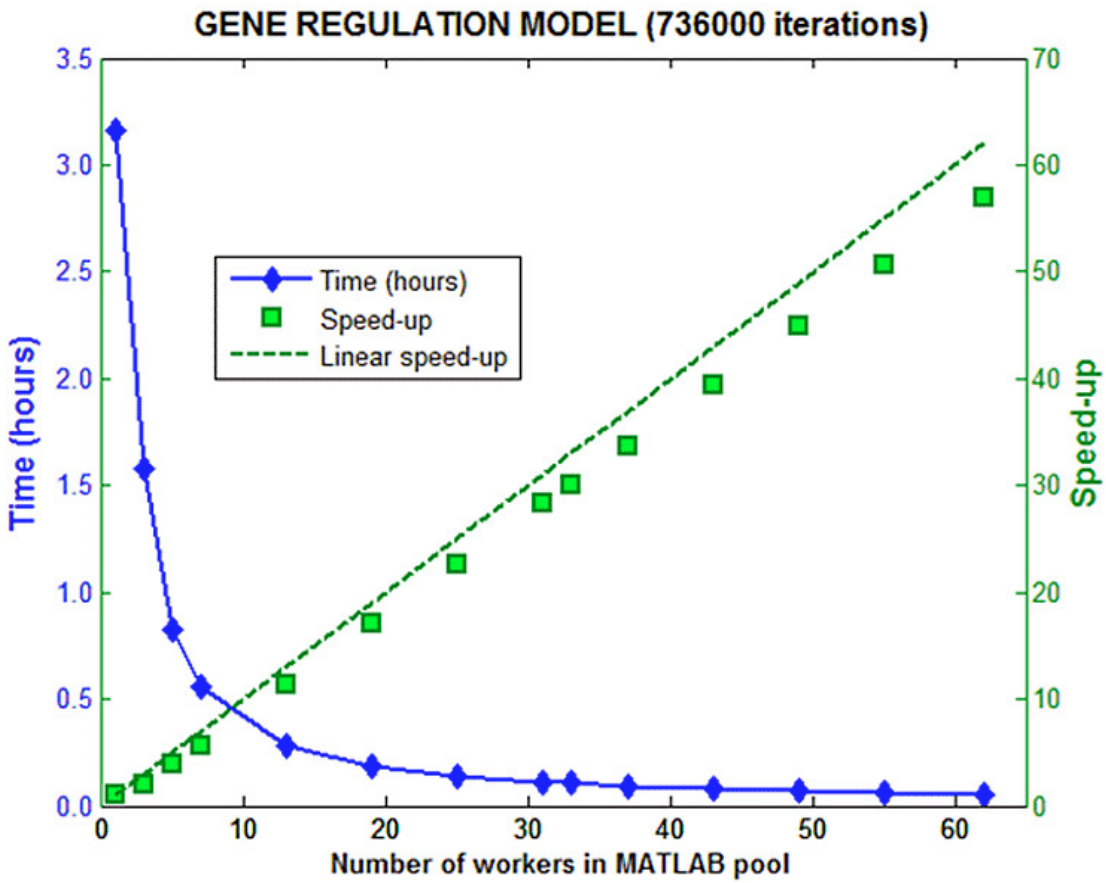
- Predefine cluster information and environment-specific parameters
- No code changes required
- Set once, use many times

Useful for both interactive and batch workflows

Toolbox provides GUI to manage configurations

Scaling Up from the Desktop

Example: Running a gene regulation model on a cluster using MATLAB Distributed Computing Server.



Parallel Matlab on ITC Linux Clusters

- ITC Linux Clusters
- Distriubted Computing Server Licenses for 256 workers. Users encouraged not to use more than 16 at one time.
- Matlab Configurations interface to PBS Pro for submitting jobs to the cluster
- Distributed Computing Server Licenses available on any Linux cluster that mounts /common for ITC servers

Interactive Parallel Matlab Jobs on Cluster

The `matlabpool` command can be used with the cluster configuration file to launch an interactive job to the cluster from within the Desktop interface on the cluster front-end.

```
>> matlabpool 'pbsproconfig_standard_2011b' 4
Starting matlabpool using the 'pbsproconfig_standard_2011b' configuration
>> [ x2, errChk ] = solver_large1(5000);
errChk =
   9.1320e-10
time =
   16.5533
>> matlabpool close
Sending a stop signal to all the labs ... stopped.
```

Interactive Parallel Matlab Jobs on Cluster

After execution of the `matlabpool` command with the cluster configuration file, the PBS command `qstat` command shows the compute nodes serving as workers have been allocated.

```
fir-s$qstat -nu teh1m

lc5.itc.virginia.edu:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1831790.lc5.itc teh1m    standard Job4        30138   4   4   16gb 02:00 R 00:00
   lc5-compute-4-33/8+lc5-compute-4-48/1+lc5-compute-3-32/13+lc5-compute-3-38/8
```

After the `matlabpool close` command, the job is exiting.

```
fir-s$qstat -nu teh1m

lc5.itc.virginia.edu:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1831790.lc5.itc teh1m    standard Job4        30138   4   4   16gb 02:00 E 00:07
   lc5-compute-4-33/8+lc5-compute-4-48/1+lc5-compute-3-32/13+lc5-compute-3-38/8
```

Additional Computational Resources

The Cross Campus Grid Project
http://www.uvacse.virginia.edu/cross-campus-grid-xcg/

XSEDE: Extreme Science and Engineering Discovery Environment
https://www.xsede.org/

MATLAB GPU Computing with NVIDIA CUDA-Enabled GPUs
A Tesla GPU card is available for testing Matlab GPU code in the CS department.

References

1. Mathworks Parallel Computing Toolbox Documentation
   http://www.mathworks.com/products/parallel-computing/
2. Mathworks Parallel Computing Toolbox Demos and Webinars
   http://www.mathworks.com/products/parallel-computing/demos.html
3. *Parallel Matlab for Multicore and Multinode Computers*, by Jeremy Kepner, SIAM Press.

Need further help? Email uvacse@virginia.edu.