Parallel Computing with MATLAB

Jemmy Hu

SHARCNET HPC Consultant
University of Waterloo

May 24, 2012

https://www.sharcnet.ca/~jemmyhu/tutorials/uwo_2012
Content

- **MATLAB:** UWO site license on goblin
- **MATLAB:** Parallel Computing Toolbox (PCT)
  - PART I: PCT on PC
  - PART II: PCT on remote cluster

Examples and Demos
Run UWO MATLAB site license on SHARCNET

- Installed on cluster ‘goblin’, version R2010b
- You have the same ‘sharcnetID’ and UWO ‘userID’
- Usage: walkthrough the software page https://www.sharcnet.ca/my/software/show/54
- Test my in /work/j8hu/matlab (UW case)
Use MATLAB compiler ‘mcc’

- Motivation: run multiple data/instances with the same code, serial farming

- Using ‘mcc’ to compile MATLAB code into standard binary code (license check once for compiling)

- Run the compiled binary as serial jobs without further license check

- Detail info online
  https://www.sharcnet.ca/my/software/show/54
Parallel Computing Toolbox

• PART 1
  - Parallel Computing Toolbox
  - Task Parallelization
  - Data Parallelization
  - Batch mode on local PC
  - Interactive with pmode
Parallel Computing Toolbox **Key Features**

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

- Ability to annotate code segments with `parfor` (parallel for-loops) and `spmd` (single program multiple data) for implementing task- and data-parallel algorithms

- High-level constructs such as **distributed arrays**, **parallel algorithms**, and **message-passing** functions for processing large data sets on multiple processors

- Ability to run 12 workers locally on a multi-core desktop (R2012a) (default to the number of cores available on a PC (not account for hyper-threading))

- Integration with MATLAB Distributed Computing Server for **cluster-based applications** that use any scheduler or any number of workers

- **Interactive** and **batch** execution modes
PCT Architecture (client-server)
• **Parallel mode on a MATLAB Pool**

  `matlabpool` Open or close pool of MATLAB sessions for parallel computation
  `parfor` Execute code loop in parallel
  `spmd` Execute code in parallel on MATLAB pool
  `batch` Run MATLAB script as batch job

• **Interactive Functions**

  `help` Help for toolbox functions in Command Window
  `pmode` Interactive Parallel Command Window
Key Function List

• **Job Creation**
  - `createJob` Create job object in scheduler and client
  - `createTask` Create new task in job
  - `dfeval` Evaluate function using cluster

• **Interlab Communication Within a Parallel Job**
  - `labBarrier` Block execution until all labs reach this call
  - `labBroadcast` Send data to all labs or receive data sent to all labs
  - `labIndex` Index of this lab
  - `labReceive` Receive data from another lab
  - `labSend` Send data to another lab
  - `numlabs` Total number of labs operating in parallel on current job

• **Job Management**
  - `cancel` Cancel job or task
  - `destroy` Remove job or task object from parent and memory
  - `getAllOutputArguments` Output arguments from evaluation of all tasks in job object
  - `submit` Queue job in scheduler
  - `wait` Wait for job to finish or change states
Typical Use Cases

- **Parallel for-Loops (parfor)**
  allowing several MATLAB workers to execute individual loop iterations simultaneously
  restriction on parallel loops is that no iterations be allowed to depend on any other iterations.

- **Large Data Sets**
  allows you to distribute that array among multiple MATLAB workers, so that each worker contains only a part of the array
  Each worker operates only on its part of the array, and workers automatically transfer data between themselves when necessary

- **Batch Jobs**
  offload work to a MATLAB worker session to run as a batch job.
  the MATLAB worker can run either on the same machine as the client, or if using MATLAB Distributed Computing Server, on a remote cluster machine.
Parallel mode-I: matlabpool

- Open or close a pool of MATLAB sessions for parallel computation

- Syntax:
  
  MATLABPOOL
  MATLABPOOL OPEN
  MATLABPOOL OPEN <poolsize>
  MATLABPOOL CLOSE
  MATLABPOOL CLOSE FORCE
  
- Work on local client PC

- Without open matlabpool, parallel code will still run but runs sequentially
Starting matlabpool using the 'local' profile ... connected to 1 labs.

Starting matlabpool using the 'local' profile ... stopped.

Error using *matlabpool* (line 144)
Failed to open matlabpool. (For information in addition to the causing error, validate the profile 'local' in the Cluster Profile Manager.)

Caused by:

Error using *distcomp.interactiveclient/start* (line 88)
Failed to start matlabpool.
This is caused by:
You requested a minimum of 2 workers, but the cluster "local" has the NumWorkers property set to allow a maximum of 1 workers. To run a communicating job on more workers than this (up to a maximum of 12 for the Local cluster), increase the value of the NumWorkers property for the cluster. The default value of NumWorkers for a Local cluster is the number of cores on the local machine.

Starting matlabpool using the 'local' profile ... connected to 2 labs.

Sending a stop signal to all the labs ... stopped.

>>
Task Parallel applications

- **parallel problems by organizing them into independent tasks (units of work)**
  - parallelize Monte Carlo simulations

- **Parallel for-Loops (parfor)**

  ```matlab
  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
Iterations run in parallel in the MATLAB pool (local workers)
Run a .m file - Demo
New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```matlab
>> matlabpool
Starting matlabpool using the 'local' profile ... connected to 2 labs.
>> matlabpool close
Sending a stop signal to all the labs ... stopped.
 >>
>> run('C:\Documents and Settings\JemmyHu\My Documents\MATLAB\pcalc.m')
Elapsed time is 15.157231 seconds.
>> matlabpool
Starting matlabpool using the 'local' profile ... connected to 2 labs.
>> run('C:\Documents and Settings\JemmyHu\My Documents\MATLAB\pcalc.m')
Elapsed time is 14.430279 seconds.
>> matlabpool close
Sending a stop signal to all the labs ... stopped.
>> run('C:\Documents and Settings\JemmyHu\My Documents\MATLAB\pcalc.m')
Elapsed time is 150.823344 seconds.
>> matlabpool
Starting matlabpool using the 'local' profile ... connected to 2 labs.
>> run('C:\Documents and Settings\JemmyHu\My Documents\MATLAB\pcalc.m')
Elapsed time is 133.301549 seconds.
```
Data Parallel applications

• Single Program Multiple Data (spmd)

```matlab
spmd (n)
    <statements>
end
```

For example, create a random matrix on four labs:

```matlab
matlabpool open
spmd (2)
    R = rand(4,4);
end
matlabpool close
```

create different sized arrays depending on `labindex`:

```matlab
matlabpool open
spmd (2)
    if labindex==1
        R = rand(4,4);
    else
        R = rand(2,2);
    end
end
matlabpool close
```
Demo

MATLAB R2012a

Command Window

>> matpool
Starting matpool using the 'local' profile ... connected to 2 labs.

>> spmd(2)
R=rand(4,4);
end

>> R(:)

ans =

[4x4 double]
[4x4 double]

>> R(1)

ans =

[4x4 double]

>> R(1)

ans =

0.3248 0.1471 0.8364 0.6218
0.6681 0.1767 0.5562 0.4399
0.6349 0.0691 0.0084 0.9658
0.6497 0.2843 0.0048 0.3897

>> R(2)

ans =

0.2646 0.0349 0.6492 0.1985
0.0968 0.8823 0.5871 0.7901
0.5082 0.0404 0.1233 0.1068
0.4666 0.6907 0.8173 0.4139
>> spmd(2)
    if labindex==1
        R=rand(4,4);
    else
        R=rand(2,2);
    end
end
>> R(:)
ans =
    [4x4 double]
    [2x2 double]
>> R(1)
ans =
    0.2700   0.3453   0.0536   0.7744
    0.8468   0.4466   0.1996   0.3975
    0.8046   0.6926   0.4874   0.6586
    0.6691   0.8541   0.6766   0.4859
>> R(2)
ans =
    0.1938   0.0858
    0.1083   0.2949
}
Distributed arrays and operations (matlabpool mode)
codistributor()
>> spmd(2)
a=rand(500,500);
b=codistributed(a)
end

Lab 1:

This lab stores b(:,1:250).

LocalPart: [500x250 double]
Codistributor: [1x1 codistributor1d]

Lab 2:

This lab stores b(:,251:500).

LocalPart: [500x250 double]
Codistributor: [1x1 codistributor1d]
Batch mode

• Name a .m file as ‘mybatch’ with
  for i=1:1024
    A(i) = sin(i*2*pi/1024);
  end
• run in batch mode
  job = batch('mybatch')

• The batch command does not block MATLAB, so you must wait for the job to finish before you can retrieve and view its results:
  wait(job)
• The load command transfers variables from the workspace of the worker to the workspace of the client, where you can view the results:
  load(job, 'A')
  plot(A)
• When the job is complete, permanently remove its data:
  destroy(job)
>> job=batch('mybatch')

job =

Job ID 13 Information

-----------------------
UserName: jemmyhu
State: running
SubmitTime: Thu Jun 02 11:01:55 EDT 2011
StartTime:
Running Duration:

- Data Dependencies

  FileDependencies: C:\Users\jemmyhu\Documents\MATLAB\mybatch.m
  PathDependencies: {}

- Associated Task(s)

  Number Pending: 1
  Number Running: 0
  Number Finished: 0
  TaskID of errors:

>> wait(job)
>> load(job, 'A')
>> plot(A)
A batch parallel loop

% mybatch
parfor i=1:1024
    A(i) = sin(i*2*pi/1024);
end

% run job in batch
job = batch('mybatch', 'configuration', 'local', 'matlabpool', 1)

% To view the results:
wait(job)
load(job, 'A')
plot(A)

% remove its data:
destroy(job)
job=batch('mybatch_par', 'configuration', 'local', 'matlabpool', 1)

job =
MatlabPool Job ID 17 Information

UserName : jemmyhu
State : queued
SubmitTime : Thu Jun 02 11:09:39 EDT 2011
StartTime :
Running Duration :

- Data Dependencies

FileDependencies : C:\Users\jemmyhu\Documents\MATLAB\mybatch_par.m
PathDependencies : {}

- Associated Task(s)

   Number Pending : 2
   Number Running : 0
   Number Finished : 0
   TaskID of errors :

- Scheduler Dependent (MatlabPool Job)

   MaximumNumberOfWorkers : 2
   MinimumNumberOfWorkers : 2

>> wait(job)
>> load(job, 'A')
>> plot(A)
>> destroy(job)
function interactiveToBatch()
    % interactive to batch

    %% for loop
    A = cell(1,4);
    for i=1:4
        A(i) = magic(i);
    end
    A(:)

    %% interactive parallel for loop
    A = cell(1,4);
    parfor i=1:4
        A(i) = magic(i);
    end
    A(:)

    %% batch job
    jm = findResource('scheduler', 'configuration', 'local');
    job = createJob(jm);
    for i=1:4
        createTask(job, @magic, 1, {i});
    end
    submit(job);
    waitForState(job);
    A = getAllOutputArguments(job);
    A(:)
end
ans =

   1

ans =

   1   3
   4   2

ans =

   8   1   6
   3   5   7
   4   9   2

ans =

   16   2   3   13
   5  11  10   8
   9   7   6  12
   4  14  15   1
Parallel mode-II: pmode

>> pmode start

P>> pmode exit
pmode demo

P>> help magic  % ask for help on a function
P>> PI = pi    % set a variable on all the labs
P>> myid = labindex  % lab ID
P>> all = numlabs  % total No. of labs
P>> segment = [1 2; 3 4; 5 6]  % create a replicated array on all the labs
P>> segment = segment + 10*labindex  % perform on different labs
P>> x = magic(4)  % replicated on every lab
P>> y=codistributed(x)  % partitioned among the lab
P>> z = y + 10*labindex  % operate on the distributed array whole
P>> combined = gather(y)  % entire array in one workspace

The combined is now a 4-by-4 array in the client workspace.
whos combined
To see the array, type its name.
combined
Demo: distributed array operations (repeat)
Lab 1

```matlab
P>> b = rand(2,8,codistributor());
This lab stores b(:,1:4).

LocalPart: [2x4 double]
Codistributor: [1x1 codistributor1d]
```

```
P>> getLocalPart(b)
ans =
```
```
0.4267   0.8379   0.6662   0.4703
0.5679   0.5489   0.7862   0.5058
```

Lab 2

```matlab
P>> b = rand(2,8,codistributor());
This lab stores b(:,5:8).

LocalPart: [2x4 double]
Codistributor: [1x1 codistributor1d]
```

```
P>> getLocalPart(b)
ans =
```
```
0.9787   0.1247   0.0250   0.1717
0.4276   0.9818   0.6535   0.4670
```

P>>
Lab 1

P>> gather(b)

ans =

Columns 1 through 7

0.4267    0.8379    0.6662    0.4703    0.9787
0.5679    0.5489    0.7862    0.5058    0.4276

Column 8

0.1717
0.4670

Lab 2

P>> gather(b)

ans =

Columns 1 through 7

0.4267    0.8379    0.6662    0.4703    0.9787
0.5679    0.5489    0.7862    0.5058    0.4276

Column 8

0.1717
0.4670
This lab stores c(1:4,:).

LocalPart: [4x2 double]
Codistributor: [1x1 codistributor1d]

```
P>> c=b'
ans =
0.4267   0.5679
0.8379   0.5489
0.6662   0.7862
0.4703   0.5058
```
Parallel pi in pmode

use the fact that

\[ \int_0^1 \frac{4}{1 + x^2} \, dx = 4(\tan(1) - \tan(0)) = \pi \]

to approximate pi by approximating the integral on the left.

divide the work between the labs by having each lab calculate the integral the function over a subinterval of [0, 1] as shown in the picture.
Steps

- All labs/workers will compute the same function: \( F = \frac{4}{1+x^2} \)

- Each worker/lab will calculate over a subinterval \([a,b]\) of \([0, 1]\), for 2 labs, the subinterval will be:
  - \([0, 0.50]\)
  - \([0.50, 1.0]\)
  
  \[a = (\text{labindex}-1)/\text{numlabs}\]
  \[b = \text{labindex}/\text{numlabs}\]

- Use a MATLAB quadrature method to compute the integral
  \[\text{myIntegral} = \text{quadl}(F, a, b)\]

- Add together to form the entire integral over \([0,1]\)
  \[\pi\text{Approx} = \text{gplus}(\text{myIntegral})\]
clear

\[ \int_0^{\infty} \frac{1}{1 + x^2} \, dx \]

\[
\begin{align*}
F &= \int_0^{\infty} \frac{1}{1 + x^2} \\
\text{ans} &= 0 \quad 0.5000 \\
\text{piApprox} &= \text{gplus}(', F', ', a, b') \\
\text{piApprox} &= 3.1416 \\
\text{abs}(\text{pi}-\text{piApprox}) &= 2.4865 \times 10^{-10}
\end{align*}
\]
function pctdemo()

% run Pi is spmd and matlabpool mode

%% open matlabpool
matlabpool open

%% define the interval
spmd
    a = (labindex - 1)/numlabs;
    b = labindex/numlabs;
    fprintf('Subinterval: [%-4g, %-4g]\n', a, b);
end

%% labs use a quadrature method to approximate each integral
spmd
    myIntegral = quadl(@pctdemo_quadpi, a, b);
    fprintf('Subinterval: [%-4g, %-4g] Integral: %g\n', a, b, myIntegral);
end

%% add the results together
spmd
    piApprox = gplus(myIntegral);
    fprintf('piApprox: %-4g\n', piApprox);
end

%% results in client
approx1 = piApprox(1);  % 1st element holds value on lab 1.
fprintf('pi     : %.18f\n', pi);
fprintf('Approximation: %.18f\n', approx1);
fprintf('Error    : %g\n', abs(pi - approx1));

%% close matlabpool
matlabpool close
>> pmode start
Starting pmode using the 'local' profile ... connected to 2 labs.
>> pmode exit
Sending a stop signal to all the labs ... stopped.
>> run('E:\Demos\pctdemo.m')
Starting matlabpool using the 'local' profile ... connected to 2 labs.
Lab 1:
    Subinterval: [0 , 0.5 ]
Lab 2:
    Subinterval: [0.5 , 1 ]
Lab 1:
    Subinterval: [0 , 0.5 ] Integral: 1.85459
Lab 2:
    Subinterval: [0.5 , 1 ] Integral: 1.287
Lab 1:
    piApprox: 3.14159
Lab 2:
    piApprox: 3.14159
pi : 3.141592653589793100
Approximation: 3.141592653838448400
Error : 2.48655e-10
Sending a stop signal to all the labs ... stopped.
>>
PART-2

- Configure MATLAB and PCT on PC
- Batch script
- Run PCT on remote clusters (hound)
- Examples and Demos
Where is the MATLAB client?

No Shared File System between clients and cluster

Client Machine

- MATLAB
- Simulink
- Toolboxes
- Blocksets

Login Node

Cluster

- Scheduler
- Workers

Shared File System

Distributed Computing Toolbox

Submit jobs

ssh

ssh
Configure MATLAB and PCT on PC

- **Cluster server side**
  - setup MATLAB distributed computing server engine
  - setup ‘matlab’ queue
  - command/script for job submission
  - * create data directory (scratch/userid/matlab)

- **Client side**
  - client configuration
  - create MATLAB batch job script
  - create local data directory ‘C:\temp’

Install and configure instruction in the online document
https://www.sharcnet.ca/help/index.php/Using_MATLAB
SHARCNET MATLAB Version info

MATLAB Version: 7.14.0.739 (R2012a)
MATLAB License Number: 359672
Operating System: Microsoft Windows XP Version 5.1 (Build 2600: Service Pack 3)
Java Version: Java 1.6.0_17-b04 with Sun Microsystems Inc. Java HotSpot(TM) Client VM mixed mode

Parallel Computing Toolbox
Version 6.0 (R2012a)

64 server side license seeds (max. 64 MATLAB workers shared by users)
Program Development Guidelines

1) **Run code normally on your local machine.** First verify all your functions so that as you progress, you are not trying to debug the functions and the distribution at the same time. Run your functions in a single instance of MATLAB® software on your local computer.

2) **Decide whether you need a distributed or parallel job.** If your application involves large data sets on which you need simultaneous calculations performed, you might benefit from a parallel job with distributed arrays. If your application involves looped or repetitive calculations that can be performed independently of each other, a distributed job might be appropriate.

3) **Modify your code for division.** Decide how you want your code divided. For a distributed job, determine how best to divide it into tasks; for example, each iteration of a for-loop might define one task. For a parallel job, determine how best to take advantage of parallel processing; for example, a large array can be distributed across all your labs.
4) **Use interactive parallel mode (pmode) to develop parallel functionality.** Use pmode with the local scheduler to develop your functions on several workers (labs) in parallel. As you progress and use pmode on the remote cluster, that might be all you need to complete your work.

5) **Run the distributed or parallel job with a local scheduler.** Create a parallel or distributed job, and run the job using the local scheduler with several local workers. This verifies that your code is correctly set up for batch execution, and in the case of a distributed job, that its computations are properly divided into tasks.

6) **Run the distributed job on a cluster node with one task.** Run your distributed job with one task to verify that remote distribution is working between your client and the cluster, and to verify file and path dependencies.

7) **Run the distributed or parallel job on multiple cluster nodes.** Scale up your job to include as many tasks as you need for a distributed job, or as many workers (labs) as you need for a parallel job.
First simple example – dfeval function

The dfeval function allows you to evaluate a function in a cluster of workers without having to individually define jobs and tasks yourself. When you can divide your job into similar tasks, using dfeval might be an appropriate way to run your job. The following code uses a local scheduler on your client computer for dfeval.

```matlab
results = dfeval(@sum, {[1 1] [2 2] [3 3]}, 'Configuration', 'local')
```

```
results =
    [2]
    [4]
    [6]
```

This example runs the job as three tasks in three separate MATLAB worker sessions, reporting the results back to the session from which you ran dfeval.
How to run it as a batch job? – Batch script

Steps taken in the toolbox to execute the `dfeval` function

```matlab
results = dfeval(@sum, {[1 1] [2 2] [3 3]}, 'Configuration', 'local')
```

1) Finds a job manager or scheduler

2) Creates a job

3) Creates tasks in that job

4) Submits the job to the queue in the job manager or scheduler

5) Retrieves the results from the job

6) Destroys the job
Batch script

- Create a job scheduler / manager object:
  `sched = findResource('scheduler', 'type', 'generic');`

- Create a job
  `j = createJob(sched);`

- Create three tasks within the job `j`
  `createTask(j, @sum, 1, {[1 1]});
  createTask(j, @sum, 1, {[2 2]});
  createTask(j, @sum, 1, {[3 3]});`

- Submit the job
  `submit(j);`

- Wait for the job to complete
  `waitForState(j);
  results = getAllOutputArguments(j);`

- Destroy the job
  `destroy(j);`
Batch script – for SHARCNET cluster

```matlab
% Specify a cluster environment and use a local folder as the JobStorageLocation
cluster = parallel.cluster.Generic('JobStorageLocation', 'C:\Temp');

% Specify File system and MATLAB Root
set(cluster, 'HasSharedFilesystem', false);
set(cluster, 'ClusterMatlabRoot', '/opt/sharcnet/matlab/R2012a');
set(cluster, 'OperatingSystem', 'unix');

% Define the additional inputs to the submit functions
clusterHost = 'hound.sharcnet.ca';
remoteJobStorageLocation = '/work/jenmyhu/matlab';

% The IndependentSubmitFcn must be a MATLAB cell array that includes the two additional inputs
set(cluster, 'IndependentSubmitFcn', {@independentSubmitFcn, clusterHost, remoteJobStorageLocation});
set(cluster, 'GetJobStateFcn', @getJobStateFcn);
set(cluster, 'DeleteJobFcn', @deleteJobFcn);

% Create a job object:
j = createJob(cluster);

% Create independent tasks
createTask(j, @sum, 1, {{1 1}});
createTask(j, @sum, 1, {{2 2}});
createTask(j, @sum, 1, {{3 3}});
% where sum is the command name, 1 is the number of return values and
% the rest is the argument list that is to be passed to sum().

% Submit the job - this is non blocking
Submit(j)

% Retrieve the job
wait(j)
results = fetchOutputs(j)
```

Demo

- test serial example in the online document `stest_hound_12a.m`

- task distribution
Second Example – a parallel job

In this example, four labs run a parallel job with a 4-by-4 magic square. The first lab broadcasts the matrix with labBroadcast to all the other labs, each of which calculates the sum of one column of the matrix. All of these column sums are combined with the gplus function to calculate the total sum of the elements of the original magic square.

The function for this example is shown below.

```matlab
function total_sum = colsum
    if labindex == 1
        % Send magic square to other labs
        A = labBroadcast(1,magic(numlabs))
    else
        % Receive broadcast on other labs
        A = labBroadcast(1)
    end
    % Calculate sum of column identified by labindex for this lab
    column_sum = sum(A(:,labindex))
    % Calculate total sum by combining column sum from all labs
    total_sum = gplus(column_sum)
end
```

DEMO: parallelexttestfunction.m, ptest_hound_10.m
function total_sum = paralleltestfunction

    if labindex == 1
        % Send magic square to other labs
        A = labBroadcast(1,magic(numlabs))
    else

        % Receive broadcast on other labs
        A = labBroadcast(1)
    end

    % Calculate sum of column identified by labindex for this lab
    column_sum = sum(A(:,labindex))

    % Calculate total sum by combining column sum from all labs
    total_sum = gplus(column_sum)
% Specify a cluster environment and use a local folder as the JobStorageLocation
cluster = parallel.cluster.Generic('JobStorageLocation', 'C:\Temp');

% Specify file system and MATLAB Root
set(cluster, 'HasSharedFilesystem', true);
set(cluster, 'ClusterMatlabRoot', '/opt/sharcnet/matlab/R2012a');
set(cluster, 'OperatingSystem', 'unix');

% Define the additional inputs to the submit functions
clusterHost = 'hound.sharcnet.ca';
remoteJobStorageLocation = '/work/jennyhu/matlab';

% If you want to run communicating jobs (including matlabpool), you must specify a CommunicatingSubmitFcn
set(cluster, 'CommunicatingSubmitFcn', @(communicatingSubmitFcn, clusterHost, remoteJobStorageLocation)
set(cluster, 'GetJobStateFcn', @getJobStateFcn);
set(cluster, 'DeleteJobFcn', @deleteJobFcn);

% Create a Job Scheduler object
numprocs = 4;
pjob = createCommunicatingJob(cluster, 'Type', 'SPMD');
set(pjob, 'NumWorkersXRange', numprocs);

% We need to include this file on the cluster
set(pjob, 'AttachedFiles', {'paralleltestfunction.m'})

% Create and submit the task, wait for results
t = createTask(pjob, @paralleltestfunction, 1, []);

% Submit and Wait for results
submit(pjob)
wait(pjob)
results = fetchOutputs(pjob)
y = results(1)
numprocs = 

4

pjob = 

ErrorIdentifier: 
ErrorMessage:

results =

[136]
[136]
[136]
[136]

y =

136

>>
function piApprox = quadpi

% QUADPI Approximate pi via parallel numerical quadrature.
% Copyright 2005 The MathWorks, Inc.
% Approximate pi by the numerical integral of F = 4/(1 + x^2) from 0 to 1.
F = @(x)4./(1 + x.^2);

% Each lab calculates the integral of F over a subinterval [a, b] of [0, 1].
a = (labindex - 1)/numlabs;
b = labindex/numlabs;

% Use a built-in MATLAB quadrature method to approximate the integral.
myIntegral = quadl(F,a,b);

% The labs have now all calculated their portions of the integral of F,
% and will all send their results to lab 1, which will add them together
% to form the entire integral over [0, 1].
if (labindex == 1)
% Receive the integral contribution from all the other labs.
piApprox = myIntegral;
else
for otherLab = 2:numlabs
    piApprox = piApprox + labReceive(otherLab)
end
end
% Send the integral contribution to lab 1.
piApprox = []
labSend(myIntegral, 1)
end
% Use a local folder as the JobStorageLocation
cluster = parallel.cluster.Generic({'JobStorageLocation', 'C:\Temp'});
set(cluster, 'HasSharedFileSystem', false);
set(cluster, 'ClusterMatlabRoot', '/opt/sharcnet/matlab/R2012a');
set(cluster, 'OperatingSystem', 'unix');
% Define the additional inputs to the submit functions
clusterHost = 'hound.sharcnet.ca';
remoteJobStorageLocation = '/work/jemmyhu/matlab';

% If you want to run communicating jobs (including matlabpool), you must specify a CommunicatingSubmitFcn
set(cluster, 'CommunicatingSubmitFcn', @communicatingSubmitFcn, clusterHost, remoteJobStorageLocation);
set(cluster, 'GetJobStateFcn', @getJobStateFcn);
set(cluster, 'DeleteJobFcn', @deleteJobFcn);

% Create a Job Scheduler object
numprocs = 4;
pjob = createCommunicatingJob(cluster, 'Type','SPMD')
set(pjob,'NumWorkersRange',numprocs);

% We need to include this file on the cluster
set(pjob,'AttachedFiles', {'quadpi.m'})
% We need to include this file on the cluster
set(pjob,'FileDependencies', {'quadpi.m'})

% Create and submit the task, wait for results
t = createTask(pjob, @quadpi, 1, {})
submit(pjob)
wait(pjob)
results = fetchOutputs(pjob)
y = results(1)
t =

Task ID 1 from Job 17 Information
============================================

    State: pending
    Function: @quadpi
    StartTime:
    Running Duration: 0 days 0h 0m 0s

- Task Result Properties

    ErrorIdentifier:
    ErrorMessage:

results =

    [3.1416]
    []
    []
    []

y =

    3.1416
References

http://www.mathworks.com/products/parallel-computing/?s_cid=HP_FP_ML_parallelcomptbx