Parallel Computing with Matlab® @ CBI Lab

Parallel Computing Toolbox™
An Introduction
Overview

- Parallel programming environment
  - configuration
  - modes of operation (preferred modes)
- Programming constructs
  - Distributed Loop using parfor
  - Single Program Multiple Data using pmode (Similar to MPI paradigm)
- Parfor performance analysis
Environment

- Starting Matlab® from within CBI environment
- Code Development can also take place on any Matlab® instance having access to the Parallel Computing Toolbox™
Environment

• 2 main user workflows

• 1) Development & testing environment
  - Use Local Configuration for development & testing of code utilizing Parallel Computing Toolbox (tm) functionality (parfor, [spmd <-> pmode]).

• 2) Running the PCT enabled developed code on Matlab® Distributed Server Cluster
  - Use batch job submission to Distributed Server Cluster of the same Parallel Computing Toolbox (tm) enabled code.

• Same Code using parfor & spmd  --> 2 running environments
### Development Environment

#### Validate the local configuration

<table>
<thead>
<tr>
<th>Default</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cluster</td>
<td>Jobmanager</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cluster_copy1</td>
<td>Jobmanager</td>
<td></td>
<td></td>
</tr>
<tr>
<td>✔</td>
<td>local</td>
<td>local</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

#### Configuration Validation

- **Test Stage**
  - Find Resource: Passed
  - Distributed Job: Passed
  - Parallel Job: Passed
  - Matlabpool: Passed

- **Max Time Per Stage**: 240 Seconds

- **Status**: Passed

- **Details**:...
Development Environment

• Check details, Find how many “workers” are available

• Since this is the local configuration, with a 4 core system, 4 workers can be used efficiently. (Note: local config has a maximum of 8 workers). We will use 5 workers on a 4 core system in later examples by changing the ClusterSize configuration parameter.

• In local mode, each worker (“lab”) maps to a different operating system Process.
Within the local development environment, constructs from the Parallel Computing Toolbox\textsuperscript{(tm)} can be used in a few ways:

1) Command line ( e.g. parfor directly from command line )
2) Script( e.g. parfor from a .m file called from command line )
3) Function( e.g parfor from a .m function )
4) Pmode ( interactive command line ). This follows the single program multiple data paradigm. Pmode is equivalent to spmd construct. Key difference is that Pmode allows you to see the output of each lab interactively, whereas spmd construct does not. Communication between labs is allowed, similar to MPI.
Development Environment

Command line (e.g. parfor directly from command line)

tic
matlab pool open local 4
n = 300
M = magic(n);
R = rand(n);
parfor i = 1:n
    for j = 1:10000
        A(i) = sqrt(sum(M(i,:).*R(n+1-i,:)));  
    end
end
toc
matlab pool close
There must be enough work in the loop to overcome the creation of the pool of workers.

~ 38 seconds (1 worker)
~ 19 seconds (4 workers)

Note: If there is no Matlab® pool open, parfor still works, it just uses only 1 worker.

Workers are mapped to a separate Matlab® process when running local configuration.
Development Environment
(Interactive SPMD Mode: Pmode)

- Parallel Command Window vs Serial Command Window
  - User can observe all labs at once
  - Each lab maps to a separate process when running in local mode
Development Environment (Interactive Mode: Pmode)

- Each worker can process different parts of the data
- Data can be combined from all workers and then sent back to the client session for plotting
Development Environment (Interactive Mode: Pmode)

- Each worker only works on a piece of the matrix
- Results are gathered on lab 1
- Client session requests the complete data set to be sent to it using lab2client
Preferred Work Environment

• Preferred method to develop code is running local.
• Preferred method to run code is batch mode.
• Same program using constructs from the Parallel Computing Toolbox™ will work in either local mode or batch mode in conjunction with the Distributed Compute Server.
In local mode, the client Matlab® session maps to an operating system process, containing multiple threads.

Each lab requires the creation of a new operating system process, each with multiple threads.

Since a thread is the scheduled OS entity, all threads from all Matlab® processes will be competing for cores.

Using the same number of labs as there are cores is recommended, but not more labs than available hardware cores.
Performance analysis & additional examples

All the Parallel Toolbox constructs can be tested in local mode, the “lab” abstraction allows the actual process used for a lab to reside either locally or on a distributed server node.

```
% [nelson.ramirez@compute-2-4 ~]$ ps
% PID TTY TIME CMD
%14294 pts/2 00:00:00 bash
%14393 pts/2 00:05:32 MATLAB --> Main Matlab client process
%23995 pts/2 00:00:00 smpd
%26064 pts/2 00:00:00 mpiexec
%26069 pts/2 00:00:00 smpd
%26070 pts/2 00:00:02 MATLAB --> Process mapping to a lab
%26071 pts/2 00:00:02 MATLAB --> Process mapping to a lab
%26072 pts/2 00:00:02 MATLAB --> Process mapping to a lab
%26073 pts/2 00:00:00 MATLAB --> Process mapping to a lab
%26391 pts/2 00:00:00 ps
%[nelson.ramirez@compute-2-4 ~]$ ps -T 26070
% PID SPID TTY STAT TIME COMMAND
% 26070 26070 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26342 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26343 pts/2 S1 01:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26352 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26370 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26371 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26372 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26373 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26374 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26375 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26376 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26377 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26378 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26379 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26380 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26381 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26382 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
% 26070 26392 pts/2 S1 00:00 /share/apps/matlab2010b/bin/glnxa64/MATLAB -dml
%[nelson.ramirez@compute-2-4 ~]$ echo
% The SPID is the thread id.
Performance analysis & additional examples

Process instantiation on local node carries overhead

Why? 14 vs 24 vs 45 seconds

While 4 local labs is better than 1 local lab, doing the work on the Matlab® client process was faster in this example, because there was not enough work to be done.

Next example: Add more compute work per lab
If there is enough computation, process instantiation overhead is overcome (48 seconds down to 26 seconds)

% Example 2: Large amount of computation

display('Starting example 2.1');
tic;
n = 300;
M = magic(n);
R = rand(n);
for i = 1:n;
    j = 1:10000;
    A(i) = sum(fft(M(i,:)).*fft(R(i,:))); % Multiply FFTs of each row;
end;
toc;

% Time approx: 48 seconds

display('Starting example 2.2');
tic;

parfor i = 1:n;
    j = 1:10000;
    A(i) = sum(fft(M(i,:)).*fft(R(i,:))); % Multiply FFTs of each row;
end;
toc;

% Time approx: 26 seconds

% Process instantiation overhead quantification

% If the Matlab workers are already up and running, we then work
% more in parallel, since we don't have to restart the
% parallel environment.

display('Starting example 2.3');
tic;

parfor i = 1:n;
    j = 1:10000;
    A(i) = sum(fft(M(i,:)).*fft(R(i,:))); % Multiply FFTs of each row;
end;
toc;

% Time approx: 19 seconds

parfor i = 1:n;
    j = 1:10000;
    A(i) = sum(fft(M(i,:)).*fft(R(i,:))); % Multiply FFTs of each row;
end;
toc;

% Time approx: 19 seconds

parfor i = 1:n;
    j = 1:10000;
    A(i) = sum(fft(M(i,:)).*fft(R(i,:))); % Multiply FFTs of each row;
end;
toc;

% Time approx: 19 seconds
Performance analysis & additional examples

% Example 3: Three parameter scan using script
%
% display('Start example 3');
%
% Use different amounts of data
dataSizeArray = [100 500 1000];
% Use different amount of computation
computeSizeArray = [10000 11000 12000 13000 14000 15000 16000 17000 18000 19000];
% Use different number of workers
workerSizeArray = [1 2 3 4 5];
% Table to keep track of time for each variation
computeTimeResults = zeros(length(dataSizeArray),length(computeSizeArray),length(workerSizeArray));

for dataIndex = 1:3 % Loop through the different amounts of data
    for computeIndex = 1:10 % Loop through the different amounts of compute time
        % Case 1
        % parfor directly from the command line
        tic; % start timer
        % Set the number of workers
        matlabpool open local 1 % open 1 worker

        % Set the data size
        n = dataSizeArray(dataIndex); % control data size
        M = magic(n); % Create a 2D matrix
        R = rand(n); % Create another 2D matrix
        A = zeros(1,n);

        % Set the compute size
        m = computeSizeArray(computeIndex);
        parfor j = 1:m % Loop through all rows of the 2D matrices
            tic; % start timer
            % Compute iteration loop
            %
            % Do any large amount of computational work
            %
            % (h) = sqrt(sum(M(:,j).*R(n+1-j,:)));
            % AC1 = sum(fft(M(:,j).*fft(R(n+1-j,:))));
            end
        toc(t1)
        computeTimeResults(dataIndex,computeIndex,j) = t1;
        matlabpool close
        %... additional #'s of workers from 2,3,4,5 workers

Performance Analysis:
Different data sizes
Different amounts of computation
Different # of labs (workers)

Hardware: 4 cores
ClusterSize set to 5 to allow creating 5 labs on a 4 core system.
( The default is having ClusterSize = # of physical cores, with a limit of 8 in a local configuration )
Performance analysis & additional examples

Performance Analysis:

- Different data sizes
- Different amounts of computation
- Different # of labs (workers)
- Hardware: 4 cores

![Graph showing runtime (seconds) for different numbers of labs (1-5) with local configuration and 4-core system, for 0.15 MB data size. The x-axis represents FFT iterations ranging from 8000 to 20000, and the y-axis represents runtime in seconds, with values ranging from 0 to 50 seconds. The graph includes lines for 1 to 5 labs, each represented by different markers.](image-url)
Performance analysis & additional examples

Performance Analysis:
Different data sizes
Different amounts of computation
Different # of labs (workers)
Hardware: 4 cores

![Graph showing runtime (seconds) vs. FFT iterations for different numbers of labs (1-5). The graph indicates that runtime increases with more FFT iterations and more labs, with a local config of 4 core system and 3.81 MB of data.]
Performance analysis & additional examples

Performance Analysis:

Different data sizes
Different amounts of computation
Different # of labs (workers)
Hardware: 4 cores
Future Presentations

- Additional examples of parfor, and spmd construct
- After program is developed in local mode: Move to Batch mode
- Examples of batch mode
  - Using the Distributed Compute Server
  - Preferred development process examples
- Profiling parallel code (mpiprobe)
- Data distribution strategies
- Inter-lab communication
- Compiling .m code for serial & parallel environment using Compiler Toolbox.
- GPU Matlab® Programming using GPU Toolbox.
CBI Laboratory

http://cbi.utsa.edu

Development team:

Zhiwei Wang, Director

David Noriega, Yung Lai, Jean-Michel Lehker, Nelson Ramirez
References

(1) http://www.mathworks.com/products/parallel-computing/
(2) http://www.mathworks.com/products/distriben/index.html
(6) http://www.nccs.nasa.gov/Matlab® _instructions.html
(7) http://www.hpc.maths.unsw.edu.au/tensor/matlab
(9) http://www.mathworks.com/products/distriben/description2.html