Matlab: Parallel Computing Toolbox

Shuxia Zhang

Supercomputing Institute

University of Mineesota

e-mail: szhang@msi.umn.edu or help@msi.umn.edu

Tel: 612-624-8858 (direct) , 612-626-0802(help)

Outline

Introduction - Matlab PCT

How to use PCT

- terminologies
- pmode
- parfor
- spmd

Hands-on exercises

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

Implicit : Multithreading in MATLAB

- MATLAB runs computations on multiple threads
- No changes to MATLAB code required
- Users can change behavior via preferences
- Maximum gain in element-wise operations and BLAS routines
- To see the performance improvements possible on your multi-core system, run the following demo:

```
>> multithreadedcomputations
>> maxNumCompThreads(1)
% set the number of threads to 1
```

Matlab
Parallel computing

Explicit multiprocessing

– The Parallel Computing Toolbox (PCT)
in the mode of distributed memory, but only on one node.

– MATLAB Distributed Computing Server (DCS),
in the mode of distributed memory, across a series of computing nodes.

– Today we will focus on the use of PCT. DCS is not available at MSI yet.

Parallel Computing ToolBox

- Implement task- and data-parallel algorithms at a high level without programming for hardware or network.
- Solve computationally and data-intensive problems on multicore and multiprocessor computers.
- Converting serial applications to parallel applications requires some, but not many code modifications
- You can run your applications interactively or offline, in batch environments.
- You can use the toolbox to execute applications on a single multicore or multiprocessor desktop. Without changing the code, you can run the same application on a computer cluster (DCS - not available at MSI yet).

Terminologies

Client - the MATLAB session where MATLAB commands are issued after the >> sign.

Workers - A worker object represents the MATLAB worker session that evaluates tasks in a job scheduled by a job manager

Local workers — processors on the local computer.

Remote workers — processors on other computing nodes instead of the local.

Task-parallel job - multiple tasks running independently and without communications among workers.

Data-parallel job - single task running simultaneously on multiple workers that may communicate with each other.

Multiple threads

Matlab -r “MaxNumCompThreads(1)” % use one thread

Matlab % it would launch n matlab threads, n= num of cores on the node

Start Matlab PCT

Load the matlab module

```
module load matlab
matlab
```

```
>> config = defaultParallelConfig()
>> conf = paralleldemoconfig()
conf =
        NumTasks: 4
        NetworkDir: [1x1 struct]
>> conf = paralleldemoconfig('NumTasks',5)
>> paralleldemoconfig('Difficulty', 0.5);
>> paralleldemo_blackjack_seq;
```

PCT - General functions

%	help	- Display help
%	dfeval	- Evaluate function
%	dfevalasync	- Evaluate function asynchronously
%	findResource	- Find available distributed computing
%	get	- Return object properties
%	defaultParallelConfig	- Control the default parallel
%	inspect	- Open Property Inspector
%	length	- Return length of object array
%	matlabpool	- Control an interactive session
%	parfor	- Parallel FOR-loop
%	pctconfig	- Configure settings for PCT
%	pmode	- Control an interactive pmode session
%	taskFinish	- Task finish M-file
%	taskStartup	- Task startup M-file

Use of pmode

```
>> pmode start local 4
```

% a window will pop up ---->
% Lab1, Lab2, … are the workers for the interactive
% parallel computing

On pmode window, one can enter commands: Fir example
p>> y=labindex
p>> labindex; z=3*y;

Functions in pmode

```
%   labBarrier          - Block execution until all labs have
                          reached this call
%   labBroadcast        - Send data to all labs or receive data
                          sent to all lab
%   labindex            - Index of this lab
%   labProbe            - Test to see if messages are ready to be
                          received
%   labReceive          - Receive data from another lab
%   labSend             - Send data to another specified lab
%   labSendReceive      - Simultaneously send data to and
                          receive data
%   numlabs             - Total number of labs or processors
```

Collective Operations

The PCT provides the following collective operations

- **`gplus`**–Global addition
  - Example : **`p = gplus(s)`**

- **`gcat`**–Global concatenation
  - Example : **`c = gcat(s)`**

- **`gop`**–Global operation
  - Example : **`m = gop(@mean, s)`**

Use of pmode

Some useful commands

- **`lab2client labvar lab clientvar`**

  Send data from the lab to the client MATLAB

- **`client2lab clientvar labs labvar`**

  Send data from the client MATLAB to the specified lab(s)

Limitations

- A maximum of 4 Workers permitted on the local node
- Workers cannot use graphics functions

  To plot data, the data must be sent to the MATLAB client that started the **`pmode`** session

Example
Use of pmode

```
>> pmode start local 4
    p>> labindex % predefined
    p>> numlabs  % predefined
    p>> a = [ 2 4; 6 8] + 50*labindex
    p>> A = codistributed(a, codistributor())
        % distributed the arrays
    p>> CA = gather(A);
>> pmode lab2client CA 1
>> plot(CA(:,1),CA(:,2))
```

Use of matlabpool and spmd

```
% matlabpool enables the parallel language features
% spmd - single program multiple data - allows interleaving of
% serial and parallel programming. The spmd environment is
% essentially equivalent to the pmode environment, but without
% the individual window for each worker.

    >> matlabpool open
    >>    spmd
    >>    ……<statements>
    >>    end
    >> matlabpool close
```

Use of matlabpool and spmd - example

```
>> matlabpool(3)
>>    spmd
      % build magic squares in parallel
>>       q = magic(labindex + 2 );
>>    end
>>    for ii=1:length( q )
      % plot each magic square
>>       figure, imagesc( q{ii} );
>>    end
>>    spmd
      % modify variable on workers
>>     q = q*2;
>>    end
    % Access data on a specific worker
>>    figure, imagesc(q{2});
```

Use of matlabpool and spmd - example

```
>> matlabpool(4)
    maxNumCompThreads(1);  % avoid multiple threading
        n=1000000;
        step = 1/n
      spmd
        nlo = ( n * ( labindex - 1 ) ) / numlabs + 1;
        nhi = ( n *   labindex       ) / numlabs;
        slocal=0;
        for i=nlo:nhi
              x=(i-0.5)*step;
        slocal=slocal+4./(1+x^2);
        end
        s = gplus (slocal, 1);
      end
      s{1} % result is stored in labindex 1
    matlabpool close
```

Use of matlabpool and parfor

parfor - Parallel FOR-loop

```
parfor loopvar = initval:endval
    <statements>
end
```

```
% The iterations of STATEMENTS can execute in parallel on separate
% MATLAB workers. In order to execute the iterations in parallel you
% MATLABPOOL must be open.
% loop index must be monotonically
% increasing integer. The following are
% not valid

>> parfor i=1:2:10
>> parfor i=-5.1:10.3
>> parfor i=[5;6;7;8]
```

```
% dependence
>> f = zeros(1,50);
>> f(1) = 1;
>> f(2) = 2;
>> parfor n = 3:50
    f(n) = f(n-1) + f(n-2);
>> end
```

Use of matlabpool and parfor example

```
>> tic;
>> for ii=1:100
     x(ii) = max( eig( abs( rand(1000 ) )) );
>> end
>> toc

>> tic;
>> parfor ii=1:100
    x(ii) = max( eig( abs( rand(1000 ) )) );
>> end
>> toc
```

```
function [time1, time2,s] = newForVersion
%% Serial version
tic; nstep=10^8; step = 1/nstep; s=0;
for i=1:nstep-1
    x=(i-0.5)*step;
s=s+4./(1+x^2);
end
time1 = toc;
```

```
%% Parallel version
matlabpool open 4
tic;
nstep=10^8; step = 1/nstep;s=0;
parfor i=1:nstep-1
    x=(i-0.5)*step;
    s=s+4./(1+x^2);
end
time2 = toc;matlabpool close
```

Convert serial into parallel code

1. Open the matlabpool for multiple morkers
2. Use parfor for task-parallel applications

   Independence - each iteration of the loop is independent of each other iteration.

   ```
   >> output = mlint('my_parfor.m');
   >> displayMlint(output)
   ```

   Reformulate the body of the loop to eliminate the dependency

3. Use spmd for data-parallel applications

   distributed Arrays
   codistributed(X)

- Use both of parfor and spmd if the code invoves both task-parallel and data-parallel applications.
- Hands-on excercises

Working in batch environement

1. Generate a m file contining the application code, e.g., use-pct.m. It contains:

```
matlabpool(4); maxNumCompThreads(1);
tic; parfor ii=1:100
x(ii) = max( eig( abs( rand( 1000 ) )) );
end; toc;  matlabpool close; quit
```

2. Create a job script file, named as job.cmd

```
#!/bin/bash -l
#PBS -l walltime=48:05:00,pmem=1500mb,nodes=1:ppn=4
cd working directory
module load matlab/matlab2009a
matlab -nodisplay < use-pct.m > my.out
```

3. Submit the job

```
qsub job.cmd
```

Reference:

http://www.mathworks.com/products/parallel-computing/index.html

Need help? Send e-mail to:

szhang@msi.umn.edu
help@msi.umn.edu

Or call: 612-624-8858
612-626-0802

Hands-on exercise: Use of pmode

1. Parallelize the folloing using parfor and spmd separately:

```
maxNumCompThreads(1);
tic;
nstep=1000000;
step = 1/nstep;
s=0;
for i=1:nstep-1
x=(i-0.5)*step;
s=s+4./(1+x^2);
end
toc
s
```

a. parfor - hint:
   replace for with parfor

b. spmd - hint:
   split nstep among the workers

Temporary accounts
Username: temp01 - temp12
Passwd: 18mSi5PW

Hands-on exercise: Solution of SPMD

```
function [time1,time2] = newSPMDVersion
n=10^8; step = 1/n;
% Serial Version
tic; s=0;
for i=1:n-1
   x=(i-0.5)*step; s=s+4./(1+x^2);
end
time1 = toc;

% Parallel Version
matlabpool open 4
tic;
spmd
   slocal = myTestSum(n,step);
end
time2 = toc;
matlabpool close
end

%%
function slocal = myTestSum(n,step)
nlo = ( n * ( labindex - 1 ) ) / numlabs + 1;
nhi = ( n *   labindex       ) / numlabs;
slocal = 0;
for i=nlo:nhi
 x=(i-0.5)*step; slocal=slocal+4./(1+x^2);
end
end
```