Parallel Programming in MATLAB
Fall 2011 update for use on the CCR cluster

Nate Barlow, PhD
HPC1 Talk

October, 27, 2011
When should you parallelize Matlab code?

The moderate overhead in Matlab can deter parallel speedup. For more scalable parallelism, one may wish to instead use MPI or openMP. Nevertheless, here are some situations in which Matlab parallelism can be quite useful:

- You have an existing serial code that solves a “large enough” problem, where the bulk of the work is non-iterative.
- You would like to use features of Matlab (graphics, symbolics, math libraries, etc.).
- You would like to use a specific toolbox that has parallel functionality.
- You plan on writing an MPI or openMP code in C or Fortran, but wish to first test your line of thinking...Matlab is a great prototyping tool, with relatively useful error messages.
- You have access to a large cluster (CCR) that has the MDCS (Matlab Distributed Computing Server).
Overview

- Types of MATLAB parallelism by example
  - `spmd` (Monte Carlo “dart throwing” example)
  - `matlabpool` & `parfor` (Fourier series example)
  - more `spmd`: domain decomposition (snowflake growth example)

- Running your MATLAB code on the CCR cluster using the MATLAB Distributed Computing Server (MDCS)
  - Setup
  - Submitting the Code
  - Monitoring the Run
  - Getting the Results
There are 2 main types of MATLAB parallelism: `matlabpool` and `spmd`. The choice of mode is specified upon submitting a job to the cluster (discussed later).

- **spmd** (single program, multiple data): The same code is run simultaneously on each core (think MPI). It’s up to you to handle unique data production and communication between cores.
  - global operations (reductions)
  - send and receive
- **matlabpool** (think OpenMP)
  - `parfor` (parallel for loops)
  - MATLAB functions with built-in parallelism
spmd (single program multiple data)

Standard “dart-throwing” Monte-Carlo $\pi$ calculation

**sequential code**

```matlab
function [time]=piMCserial(N)
tic
n=0;
for j=1:N
    x=rand; y=rand;
    if (x^2+y^2)<=1, n=n+1; end
end
mypi=4*n/N
error=abs(pi−mypi)
time=toc;
end
```

**parallel code**

```matlab
function [time]=DCSpiMC(N)
tic
n=0;
for j=1:N
    x=rand; y=rand;
    if (x^2+y^2)<=1, n=n+1; end
end
mypi=4*gplus(n)/(numlabs*N); %global sum
error=gop(@max, abs(pi−mypi)); %global max
end
```

labindex identifies the rank of the processor/core/lab, starting with 1.

numlabs is the number of cores being used.

gplus is the + reduction operator. gop is used for other types of reductions.

m-files must be written as functions

The workspace is packed in to a .mat file and saved to a directory created beforehand.
To quickly parallelize a serial Matlab code, look for loops with no dependencies at all between iterations. Then simply replace the `for`’s with `parfor`’s. The items in the loop will be distributed across machines in a `matlabpool`. It’s basically MATLAB’s version of the openMP “parallel for” loop.

- **non-iterative**
  
  ```matlab
  for j = 1:8
    A(j) = j;
  end
  ```

  **parfor will work**
  
  ```matlab
  parfor j = 1:8
    A(j) = j;
  end
  ```

- **iterative**

  ```matlab
  n = 0;
  for j = 1:8
    n = n + 1;
    A(j) = n;
  end
  ```

  **parfor will not work**

  ```matlab
  n = 0;
  parfor j = 1:8
    n = n + 1;
    A(j) = n;
  end
  ```
using \textit{parfor} to compute a Fourier series

Ex: resolve the 15\textsuperscript{th} mode of a free rectangular membrane, vibrated from the center

- The Problem:
  - plate height is $u(x, y, t)$ in $-a < x < a; -b < y < b$
  - forced 2-D wave equation: $\nabla^2 u - \frac{1}{c^2}u_{tt} = \delta(x)\delta(y)\cos(\omega t)$
  - boundary conditions: $u_x(\pm a, y, t) = 0, u_y(x, \pm b, t) = 0$
  - initial conditions: $u(x, y, 0) = u_t(x, y, 0) = 0$

- Fourier series solution:

$$u(x, y, t) = T_0(t) + \sum_{n=1}^{\infty} \Phi_n(y) T_n(t) + \sum_{m=1}^{\infty} \Phi_m(x) T_m(t) + \sum_{m=1}^{\infty} \sum_{n=1}^{\infty} \Phi_{mn}(x, y) T_{mn}(t)$$

where $T_0(t), \Phi_n(t), T_n(t), \Phi_m(x), T_m(t), \Phi_{mn}(x, y)$, and $T_{mn}(t)$ depend on the parameters $a, b, \omega,$ and $c$. The plate resonates when $\omega^2 = c^2 \left[ \left( \frac{m\pi}{a} \right)^2 + \left( \frac{n\pi}{b} \right)^2 \right]$. The 15\textsuperscript{th} mode refers to $n = m = 15$. 

Nate Barlow, PhD (HPC1 Talk)  
Parallel Programming in MATLAB  
Fall 2011  7 / 29
using *parfor* to compute a Fourier series

The Code:

```matlab
% plate.m

function plate = f(mode1, mode2, N)

a = 1; b = 1; % dimensions of rectangle
c = 0.5; % stiffness of membrane
t = 1.06; % particular time to look at
w = c * sqrt((mode1 * pi / a)^2 + (mode2 * pi / b)^2) + 1e-12; % forcing near resonance
n = 1:N; m = n;
dx = a / (N * pi); dy = b / (N * pi); eigm = m * pi / a; eign = n * pi / b;

T0 = (cos(w * t) - 1) / (2 * w^2);
Tn = (cos(w * t) - cos(c * eign * t)) / (w^2 - c^2 * eign^2);
Tm = (cos(w * t) - cos(c * eigm * t)) / (w^2 - c^2 * eigm^2);

tic
parfor j = 1:floor(a / dx + 1)
    x = (j - 1) * dx - a / 2;
    PhiM(j) = cos(eigm * x) * Tm';
end

parfor j = 1:floor(b / dy + 1)
    y = (j - 1) * dy - b / 2;
    PhiN(j) = cos(eign * y) * Tn';
end
```

Nate Barlow, PhD (HPC1 Talk)  Parallel Programming in MATLAB  Fall 2011
using `parfor` to compute a Fourier series

either the inner or outer loops can be parallelized...but not both (`parfor` cannot be nested)
using `parfor` to compute a Fourier series

- The Results: modal lines computed with N terms in the series
  
  $N = 100$  
  $N = 200$  
  $N = 400$  

\[ u_{200} - u_{100} \]  
\[ u_{400} - u_{200} \]  
\[ \Delta u \]
using parfor to compute a Fourier series

Performance:

Parallel Speedup = \frac{\text{Sequential Execution Time}}{\text{Parallel Execution Time}}

The problem should be large enough to justify the use of multiple machines. For this problem, using more than 6 cores is only beneficial if \( N > 200 \).
clear; clc; clf; colormap bone
L=21; T=9; CI=60; % grid size, # of steps, and colormap intensity
alpha=1; beta=0.65; gamma=0.0001; %parameters

%STEP 1: initialize grid
A=beta*ones(L); A2=A; A2n=A2; A1=zeros(L);
c=ceil(L/2); A(c,c)=alpha;

for n=1:T %STEP 2: scan cells
    for i=2:L−1
        for j=2:L−1
            if A(i, j)>=alpha || A(i+1, j)>=alpha || A(i−1, j)>=alpha ...
                || A(i, j−1)>=alpha || A(i+1, j+1)>=alpha || A(i−1, j−1)>=alpha ...
                || A(i+1, j−1)>=alpha
                A1(i, j)=A(i, j)+gamma; %STEP 3: grow ice
                A2(i, j)=0;
            else
                A1(i, j)=0; A2(i, j)=A(i, j);
            end
        end
    end
    for i=2:L−1 %STEP 4: diffuse water
        for j=2:L−1
            A2n(i, j)=(A2(i+1, j)+A2(i−1, j)+A2(i, j+1)+A2(i, j−1)...
                        +A2(i+1, j+1)+A2(i−1, j−1)+A2(i+1, j−1)+A2(i−1, j+1))/8;
        end
    end
    image(CI*A); drawnow; pause(.5); %plot growth
    A=A1+A2n; %STEP 5: add updated water and ice
    A2=A2n; %update water for next iteration
end
Matlab uses “column-major order” to store arrays in memory. It seems appropriate then to strip the domain by columns. Note: In the C programming language, it is preferable to strip by rows.
For 4 cores (numlabs is 4), assign each core a piece of the domain

Functions sent through the MDCS act as SPMD’s (single program, multiple data). A single m-file will run on every core, but can operate on different sections of data.

labindex and numlabs are defined in Matlab and can be used in your code to distinguish between different sections of the data.
labindex:  1  2  3  4

- Pad $\mathbf{A}$ with “ghost” and boundary cells. The ghost columns will allow for communication b/w labs.

- Since a *single* program is run, $\mathbf{A}$ now refers to a $12 \times 5$ matrix with different contents on each lab. The former (sequential) $\mathbf{A}$ is the horizontal concatenation of all inner ($12 \times 3$) matrices.

- Boundary cells are a result of $\mathbf{A}$ having 5 columns (on every core); they may be unused or can frame the computation with boundary conditions.
labindex: 1 2 3 4

For a middle lab (3 or 4), the following commands are used (cols=5 for above ex.):

- `labSend(A(:,2),labindex-1)` sends column 2 from the current lab to the preceding lab.
- `A(:,1)=labReceive(labindex-1)` receives data from the preceding lab and stores it in the left ghost column (column 1).
- `labSend(A(:,cols-1),labindex+1)` sends column cols-1 (column 4) from the current lab to the next lab.
- `A(:,cols)=labReceive(labindex+1)` receives data from the next lab and stores it in the right ghost column (column 5).
For computations that depend on the neighboring cells, the handling of send/receive commands is important!

Matlab is known for its relative ease of use, when prototyping or debugging a problem. This is extended for the MDCS, so let us try a few parallelization choices and then see if they are right.

To communicate b/w ghost columns of $A$, the following is inserted at the beginning of every iteration:

```matlab
if labindex>1
    labSend(A(:,2),labindex-1);
end
if labindex<numlabs
    labSend(A(:,cols-1),labindex+1);
    A(:,cols)=labReceive(labindex+1);
end
if labindex>1
    A(:,1)=labReceive(labindex-1);
end
```

Notice that all statements apply to the middle labs, but only 2 statements are associated with each boundary lab.
Flakes for $n = 4, 8, \text{ and } 16$ cores

$\text{attempt}\#2(\text{wrong})$

$\text{attempt}\#3(\text{wrong})$

$\text{attempt}\#4(\text{correct})$

Moral: If the answer changes as you increase the # of cores, recheck your send/recvies!
How much faster does the code become as the # of labs increases?

![Graph showing execution time vs. number of cores for 512 x 512 and 1024 x 1024 domain sizes.](image)

- **512 x 512 domain**
- **1024 x 1024 domain**
Parallel Speedup = \frac{\text{Sequential Execution Time}}{\text{Parallel Execution Time}}

Speedup tends to increase with problem size \((L)\)

The parallel speedup increases with \(L\).

Nate Barlow, PhD (HPC1 Talk)
Parallel Programming in MATLAB
Fall 2011  20 / 29
There are a few ways to write graphics to `remoteDataLocation` for later retrieval.

If you are dealing with a matrix $A$ where the cells can be interpreted as pixels and the values as colors, the following command will print the matrix to a jpeg:

```matlab
imwrite(A, 'A.jpg');
```

Remember, Matlab is started on each lab (processor), so this file will be printed (and overwritten) from each processor!

If $A$ is different on each processor. Then $A$.jpg will be the the last one to print.

To rename the file for each of $n$ labs and store $n$ jpegs, issue the following commands:

```matlab
filename=strcat('A',int2str(labindex),'.jpg');
imwrite(A,filename);
```

The files A1.jpg, A2.jpg, ... will be created, one for each lab.
Parallel Visualization during the computation

If the preceding lines are put in a loop where A.jpg is overwritten at each iteration, you can view the pieces of the computation mid-computation. To do this, open a terminal into the cluster and navigate to the relevant Job folder in remoteDataLocation, then type:

```
[nbarlow2@edge Job482]$ display A1.jpg -update 2&
[1] 6250
[nbarlow2@edge Job482]$ display A2.jpg -update 2&
[2] 10084
[nbarlow2@edge Job482]$
```

The image will update every 2 seconds, showing the computation on each lab as it progresses.
If you wish to combine \( A \) from all labs and just print it from one processor, you could issue something like this:

```matlab
AA = gop(@horzcat, A);
if labindex == 1, imwrite(A, 'A.jpg'); end
```

In our example we use \( AA = gop(@horzcat, A(:,2:cols-1)) \) instead because it excludes the ghost columns.

To store each image of an iteration as a frame in a movie, follow the following structure in your m-file:

```matlab
if labindex == 1
    vidObj = VideoWriter('A.avi');
    open(vidObj);
end
for n = 1:15000
    (iteration code)
    AA = gop(@horzcat, A);
    if labindex == 1
        imwrite(AA, 'A.jpg');
        f1 = imread(filename);
        writeVideo(vidObj, f1);
    end
end
```
Using the MDCS on CCR’s cluster

- The MDCS is configured to use CCR’s purchase of Intel-MPI and has the capability to communicate over Infiniband.
- Under our current license agreement, 128 total cores can be used at once.
- If you purchase a copy of MATLAB R2010b (or later) from UB, you have the option of submitting jobs from within your local MATLAB installation, as long as you are connected to the UB network and have a CCR account. More details are given here (http://www.ccr.buffalo.edu/display/WEB/Matlab+Distributed+Computing+Server+(MDCS)+Tutorial)
MDCS Setup

Login into the CCR cluster and create 3 directories:

- Create a directory for temporary storage of Matlab output. This will also serve as the "working directory". We suggest creating a "username-matlab" folder in /panasas/scratch or using a /projects space.

```bash
[user@u2 ~]$ cd /panasas/scratch
[user@u2 scratch]$ mkdir username-matlab
```

- Create a directory for input files in your home directory. Copy the following submit script and sample code into this directory.

```bash
[user@u2 ~]$ mkdir sample-run
[user@u2 ~]$ cd sample-run/
[user@u2 ~/sample-run]$ cp /util/pbs-scripts/DCSsubmitC2C.m .
[user@u2 ~/sample-run]$ cp /util/pbs-scripts/DCSpiMC.m .
```

- Create a directory for MATLAB output in your home directory.

```bash
[user@u2 ~]$ mkdir MDCS_output
```
Navigate to the sample-run directory, startup MATLAB, and open DCSsubmitC2C.m:

```
[user@u2:~/sample-run]$ module load matlab/R2011b
[user@u2:~/sample-run]$ matlab -nodesktop

< MATLAB (R) >
Copyright 1984–2011 The MathWorks, Inc.
R2011b (7.13.0.564) 64–bit (glnxa64)
August 13, 2011

Your MATLAB license will expire in 05 days.
Please contact your system administrator or
MathWorks to renew this license.

To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.

>> edit DCSsubmitC2C.m
```
MDCS: Submitting the Code

- Make the appropriate changes

```
%%N. Barlow 10/2011

clear

%% The following lines below should remain unchanged
sched = findResource('scheduler', 'type', 'generic');
set(sched, 'ClusterMatlabRoot', '/util/matlab/R2011b');
set(sched, 'HasSharedFileSystem', true);
set(sched, 'ClusterOSType', 'unix');
set(sched, 'GetJobStateFcn', @getJobStateFcn);
set(sched, 'DestroyJobFcn', @destroyJobFcn);

%% modify below
set(sched, 'DataLocation', '/panasas/scratch/usernameMatlabData'); % folder for output files, create if it doesn't exist
ppn=8;
time='01:00:00';
email='username@buffalo.edu';
nodoFlag='IB2'; % for ppn-12 use 'IB1', for ppn-8 use 'IB2', for ppn-2 use 'CM'

%% Don't modify this line
set(sched, 'ParallelSubmitFcn', @parallelSubmitFcn, ppn, time, nodoFlag, email);

%% for spmd code
pjob = createParallelJob(sched, 'MinimumNumberOfWorkers', 1, 'MaximumNumberOfWorkers', 8)

%% for code with parfor loops or functions w/ built-in parallelism (ex. optimization toolbox stuff)
pjob = createMatlabPoolJob(sched, 'MinimumNumberOfWorkers', 8, 'MaximumNumberOfWorkers', 3)

%% Example for a function w/ 1 output: DCSpiMC(10)
set(pjob, 'FileDependencies', {'DCSpiMC.m'}); % list all user-created stuff
createTask(pjob, @DCSpiMC.1{10});
submit(pjob)
```
Run the script; this submits the MATLAB code through PBS

```
>> DCSsubmitC2C
pjobj =

Parallel Job ID 13 Information
==============================================
    UserName : nbarlow2
    State : pending
    SubmitTime :
    StartTime :
    Running Duration :

− Data Dependencies
    FileDependencies : {}
    PathDependencies : {}

− Associated Task(s)
    Number Pending : 0
    Number Running : 0
    Number Finished : 0
    TaskID of errors :

− Scheduler Dependent (Parallel Job)
    MaximumNumberOfWorkers : 8
    MinimumNumberOfWorkers : 1
```
The job will enter the matlab queue:

```
[user@u2:~]$ qstat -u nbarlow2
d15n41.ccr.buffalo.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Req’d Memory</th>
<th>Req’d Time</th>
<th>Elap Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1203555.c</td>
<td>nbarlow2</td>
<td>matlab</td>
<td>Job13</td>
<td>—</td>
<td>1</td>
<td>8</td>
<td>—</td>
<td>01:00</td>
<td>Q</td>
</tr>
</tbody>
</table>
```

After the job finishes, `stuff.mat` will be saved to the MATLAB output directory created earlier.