Using the MATLAB Parallel Computing Toolbox on the UB CCR cluster

N. Barlow, C. Cornelius, S. Matott

Center for Computational Research
University at Buffalo
State University of New York

October, 1, 2013

Outline

- CCR resources (in general, as of October 2013)
- parallel MATLAB resources at CCR
- MATLAB's Parallel Computing Toolbox:
  Running on <u>one</u> compute node
  - running MATLAB interactively
  - matlabpool and parfor
  - spmd, labindex, and numlabs
  - gplus and gop
- Examples:
  - Monte-Carlo π calculation
  - spmd implementation of the Julia set fractal
- submitting a non-interactive parallel MATLAB job to the general (ccr) queue
- MATLAB Distributed Computing Server

CCR resources

- using SLURM:
  http://ccr.buffalo.edu/support/UserGuide/slurm.html
- cluster compute node information:
  http://ccr.buffalo.edu/support/research_facilities/general_compute.html
- CCR MATLAB resources:
  http://ccr.buffalo.edu/support/software-resources/compilers-programming-languages/matlab.html
- remote visualization:
  http://ccr.buffalo.edu/support/research_facilities/remote-visualization.html
- MATLAB PCT example (available from the **rush** front-end):
  /util/matlab/example/MyMatlabScript.m
  /util/matlab/example/slurmMATLAB

parallel MATLAB resources at CCR

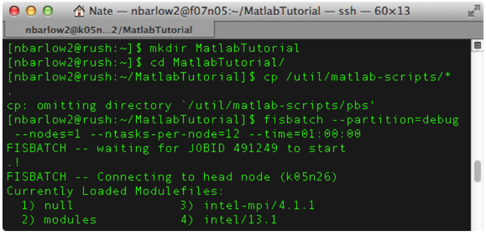
- Parallel Computing Toolbox (PCT)
  - 1 parallel job can use a maximum of 12 cores
  - parallel jobs are restricted to run *within* a node (like open-mp).
  - unlimited licenses: unlimited # of users can submit an unlimited # of 12-core jobs each.
- Matlab Distributed Computing Server (MDCS)
  - 1 parallel job can use a maximum of 256 cores
  - parallel jobs can run between nodes using MPI (Message Passing Interface).
  - 256-core license: users share licenses, such that the sum of all cores being used by all MDCS users (at any given instant) cannot exceed 256.

MATLAB PCT: running interactively on a node

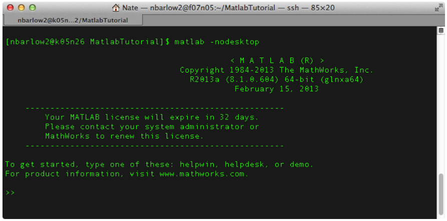
- Open a terminal into the frontend (rush) with graphics enabled (use the `-X` option)
    - Instructions given here: http://ccr.buffalo.edu/support/UserGuide/login.html
- Within your home directory, create a new directory for this tutorial. Copy our example scripts (in `/util/matlab-scripts`) into your new directory.
- When using MATLAB interactively, it is important to request your <u>own</u> compute node to do the computations (i.e. do not run on the front-end).
- Request an (example) interactive job by typing the following:

```
fisbatch --partition=debug --nodes=1 --ntasks-per-node=12
--time=01:00:00
```

MATLAB PCT: running interactively on a node


Nate — nbarlow2@f07n05:~/MatlabTutorial — ssh — 60×13
nbarlow2@k05n...2/MatlabTutorial

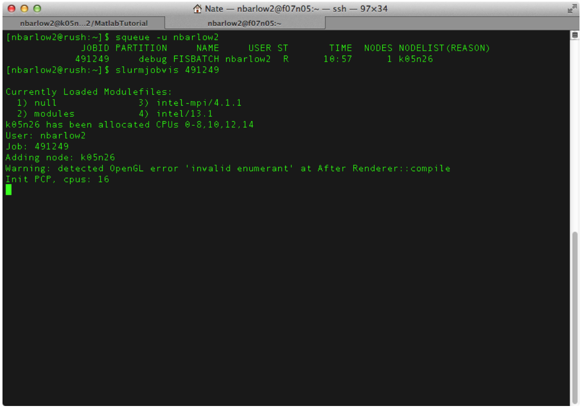
```
[nbarlow2@rush:~]$ mkdir MatlabTutorial
[nbarlow2@rush:~]$ cd MatlabTutorial/
[nbarlow2@rush:~/MatlabTutorial]$ cp /util/matlab-scripts/*
.
cp: omitting directory `/util/matlab-scripts/pbs'
[nbarlow2@rush:~/MatlabTutorial]$ fisbatch --partition=debug
 --nodes=1 --ntasks-per-node=12 --time=01:00:00
FISBATCH -- waiting for JOBID 491249 to start
.!
FISBATCH -- Connecting to head node (k05n26)
Currently Loaded Modulefiles:
  1) null                3) intel-mpi/4.1.1
  2) modules             4) intel/13.1
```


Nate — nbarlow2@f07n05:~/MatlabTutorial — ssh — 85×20
nbarlow2@k05n...2/MatlabTutorial

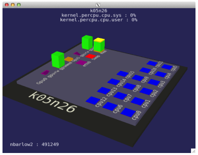
```
[nbarlow2@k05n26 MatlabTutorial]$ matlab -nodesktop

                                          < M A T L A B (R) >
                                Copyright 1984-2013 The MathWorks, Inc.
                                  R2013a (8.1.0.604) 64-bit (glnxa64)
                                           February 15, 2013

    ----------------------------------------------------
        Your MATLAB license will expire in 32 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.

>>
```

- load the matlab/R2013a module and start MATLAB in “nodesktop” mode (see above). ”nodesktop” mode does not load the full MATLAB graphical interface, but still permits access to graphical utilities such as the editor and plots.
- If MATLAB takes a long time to load and/or you are running over a wireless network (or off-campus), it is sometimes better to use start MATLAB by typing `matlab -nodisplay` which disables all graphics.

MATLAB PCT: running interactively on a node


Nate — nbarlow2@f07n05:~ — ssh — 97×34
nbarlow2@k05n...2/MatlabTutorial
nbarlow2@f07n05:~

```
[nbarlow2@rush:~]$ squeue -u nbarlow2
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            491249     debug FISBATCH nbarlow2  R      10:57      1 k05n26
[nbarlow2@rush:~]$ slurmjobvis 491249

Currently Loaded Modulefiles:
  1) null              3) intel-mpi/4.1.1
  2) modules           4) intel/13.1
k05n26 has been allocated CPUs 0-8,10,12,14
User: nbarlow2
Job: 491249
Adding node: k05n26
Warning: detected OpenGL error 'invalid enumerant' at After Renderer::compile
Init PCP, cpus: 16
```



- In a separate terminal (login into rush w/graphics again), type `squeue -u username` to obtain the jobID for your interactive job. Then type `slurmjobvis jobID` (see above) to view the activity on each core of your requested node.

MATLAB PCT: parallelism w/out using PCT commands

- A little known fact: common functions such as exp, sqrt, and others may automatically run over multiple cores **with no change to your code**!
- As you might imagine, this can cause some confusion when comparing the performance of your parallelized code with a (seemingly) non-parallelized code.
- A list of functions that “multi-thread” are given here: http://www.mathworks.com/support/solutions/en/data/1-4PG4AN/?solution=1-4PG4AN
- Try: Apply various MATLAB functions to a large matrix. Look in your slurmjobvis window to determine if the work is being split across multiple cores.

MATLAB PCT: matlabpool

- Although we requested 12 cores, MATLAB is only expected to execute PCT commands over multiple cores if we tell it to open a matlabpool of 12 cores (or less, if we desire). In order to open a new pool, any previous pool must first be closed. The basic structure is:

```
>> matlabpool 12
% (code that may or may not use 12 cores)
>> matlabpool close

>> matlabpool 4
% (code that may or may not use 4 of the 12 requested cores)
>> matlabpool close
```

- Try: Open a matlabpool on x number of cores (with $x < 12$). How many cores (cpus) become active in slurmjobvis? Now close the pool.

MATLAB PCT: parfor

- Each year, the list of MATLAB functions that can make use of multiple cores increases. An example of such a function is parfor (parallel for-loop).
- A parfor loop arbitrarily splits up work between the # of cores specified by the pool. The work on each core is done **independently** for the entire duration of the loop, and so the elements of the loop must be independent (i.e. the loop must be **non-iterative**). The structure:

```
>> matlabpool 4
% (code that may or may not use 4 of the 12 requested cores, ex: create a vector x)

>> parfor j=1:length(x)
% (non—iterative code to be split over 4 cores, ex: create a vector y=f(x))
>> end

% (more code that may or may not use 4 of the 12 requested cores, ex: use y for something)
>> matlabpool close
```

- Try: Do the above. Make the vector size large enough so that the computation lasts for more than a few seconds. Using slurmjobvis, can you tell which portion of the code you are in at a given instant? Repeat this for different size pools.

MATLAB PCT: spmd

- Code within a spmd (single program, multiple data) statement will run on all cores in a pool.
- Each core will run an exact replica of the code simultaneously (i.e. single program). spmd uses the MPI (Message Passing Interface) protocol.
- The data produced from each core may be different (i. e. multiple data), depending how one uses the core-identifying variable labindex (rank) as well as other spmd commands, such as numlabs (total number of cores specified by pool).

```
>> matlabpool 3
Starting matlabpool using the 'local' profile ... connected to 3 workers.
>> spmd
['hello from lab ', int2str(labindex),' of ', int2str(numlabs)]
end
>> matlabpool close
Sending a stop signal to all the workers ... stopped.
```

- Try: Do the above for however many cores you would like. Modify it – put some math in the spmd statement and see what happens.

MATLAB PCT: spmd

- Some other commands that can be used within an spmd statement:
  - gplus - global addition across cores.
    type: `help gplus`
  - gop - general global operation to be done across cores.
    type: `help gop`
  - send and receive pass information between cores – details are given in Workshop II.

spmd example: Monte-Carlo π calculation

- The code `piMC.m` randomly chooses *N* points in a unit square and then finds the ratio of points lying within an inscribed circle to those outside of the circle.
- As *N* increases, the average ratio should limit to $\frac{\text{area of circle}}{\text{area of square}} = \pi/4$.

```
function [time]=piMC(N)
% Monte-Carlo pi calculation
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 (global op here is redundant)
if labindex==1
    save('stuff.mat')
end
time=['took ' num2str(toc) ' seconds'];
```

- The above code provides $N \times$ numlabs independent trials.
- When calculating the error, why is the global maximum not actually needed?
- Why is an if-statement needed when saving the workspace to a .mat file?

spmd example: Monte-Carlo π calculation

- Try: Run piMC inside a spmd statement within pools of 3, 6, and 12 cores. For each case use 12×10^7 trials. Note that $N = \frac{\text{\# of trials}}{\text{\# of cores}}$; see example below.

```
>> matlabpool 3
Starting matlabpool using the 'local' profile ... connected to 3 workers.
>> spmd
piMC(4e7)
end
Lab 1:
  ans =
  took 5.6285 seconds

Lab 2:
  ans =
  took 5.6253 seconds

Lab 3:
  ans =
  took 5.6252 seconds

>> load stuff
>> error

error =

   4.3420e-05
>>matlabpool close
```

- For a fixed # of trials, how much speedup (if any) do you gain by running over twice the # of cores?

spmd example: Julia set fractal

- Julia.m generates a fractal, which is written to a bitmap image.
- The main loop of Julia.m is non-iterative, and so the work can be split into independent tasks to run over multiple cores.
- The sequential loop (shown below)

```
 for i = 0:1:(dim-1)
 .
 .
%main code
 .
 .
end
```

is changed to

```
 for i = (labindex -1):numlabs:(dim-1)
 .
 .
%main code
 .
 .
end
```

- When using 1 core, the two above loops are equivalent (labindex=1, numlabs=1). When the code is run over multiple cores within an spmd statement, how are the loops different on each core?

spmd example: Julia set fractal

- The aggregate image is written to `Julia.bmp`. The image on each core is written to `1.bmp`, `2.bmp`, `....`

```
S = gplus(bmp, 1); % collect results
if(labindex == 1)
   imwrite(S, 'Julia.bmp'); % write image to file
end
imwrite(bmp, [num2str(labindex),'.bmp']);
```

- Try: Run the code within an spmd statement inside a matlabpool. Use these inputs to the code:

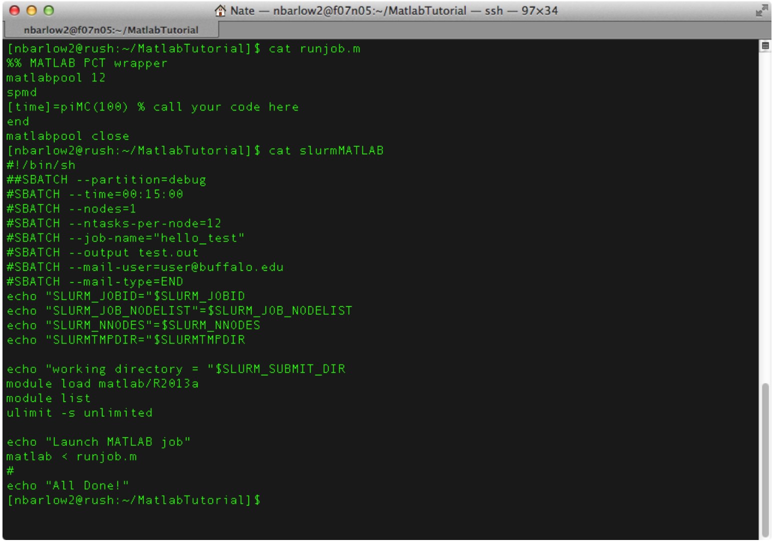
```
>>Julia(288, −0.8, 0.156)
```

- After running the code, display the aggregate image by typing:

```
>>!display Julia.bmp
```

- Look also at the images from each core. Do they look as expected?

MATLAB PCT: submitting a job to the queue



Nate — nbarlow2@f07n05:~/MatlabTutorial — ssh — 97×34
nbarlow2@f07n05:~/MatlabTutorial

```
[nbarlow2@rush:~/MatlabTutorial]$ cat runjob.m
%% MATLAB PCT wrapper
matlabpool 12
spmd
[time]=piMC(100) % call your code here
end
matlabpool close
[nbarlow2@rush:~/MatlabTutorial]$ cat slurmMATLAB
#!/bin/sh
##SBATCH --partition=debug
#SBATCH --time=00:15:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=12
#SBATCH --job-name="hello_test"
#SBATCH --output test.out
#SBATCH --mail-user=user@buffalo.edu
#SBATCH --mail-type=END
echo "SLURM_JOBID="$SLURM_JOBID
echo "SLURM_JOB_NODELIST"=$SLURM_JOB_NODELIST
echo "SLURM_NNODES"=$SLURM_NNODES
echo "SLURMTMPDIR="$SLURMTMPDIR

echo "working directory = "$SLURM_SUBMIT_DIR
module load matlab/R2013a
module list
ulimit -s unlimited

echo "Launch MATLAB job"
matlab < runjob.m
#
echo "All Done!"
[nbarlow2@rush:~/MatlabTutorial]$
```