Parallel Computing with Matlab and R

scsc@duke.edu

https://wiki.duke.edu/display/SCSC

Tom Milledge
tm103@duke.edu

Overview

- Running Matlab and R interactively and in batch mode
- Introduction to Parallel Computing
- Running Matlab and R as Array jobs
- Using the Matlab Parallel Computing Toolbox
- Using Rmpi and SNOW
- Using GPUs as accelerators for Matlab and R
**Running Matlab Interactively**

Run Matlab with the qrsh command:

```bash
head4 % qrsh
stat-n03 % /opt/matlabR14/bin/matlab -nojvm -nodisplay
```

or:

```bash
head4 % qrsh
stat-n03 % /opt/matlab2007a/bin/matlab -nojvm -nodisplay
```

or:

```bash
head2 % qrsh -l centos5
core-n75 % /opt/matlabR2009a/bin/matlab -nojvm -nodisplay
```

**Using Matlab with SGE**

```bash
#!/bin/tcsh
#
#$ -S /bin/tcsh -cwd
#$ -o matlab.out -j y
#$ -l centos5

/opt/matlabR2009a/bin/matlab -nojvm -nodisplay -r my_program
```

- `-r` option: immediately run an m-function instead of presenting an interactive prompt
- "my_program.m" (with a ".m" extension), but the Matlab executable is called with "my_program" (without the ".m" extension).
A simple example

% simple Matlab script

% do work here
A = eye(5,5);
x = (1:5)';
y = A*x;

% leaving of the semicolon outputs y to the screen
% where it is captured by SGE and sent to the -o file
y'
quit

Your Matlab script must call the 'quit' command at the end of it's processing or else it will run forever!

About R

About R (http://www.r-project.org/):

R is an Open Source (GPL), most widely used programming environment for statistical analysis and graphics; similar to S.
Provides good support for both users and developers.
Highly extensible via dynamically loadable add-on packages.
Originally developed by Robert Gentleman and Ross Ihaka.
Running R Interactively

Run R with the qrsh command:

(Version 2.2.1)

head4 % qrsh
stat-n03 % R --vanilla

or: (version 2.7.1)

head4 % qrsh
stat-n03 % /opt/R271/bin/R --vanilla

or: (version 2.9.2)

head2 % qrsh –l highprio
core-n75 % R --vanilla

Using R with SGE

#!/bin/tcsh
#
#$ -S /bin/tcsh -cwd
#$ -o R.out -j y

R CMD BATCH My_R_program results.Rout

- CMD BATCH options which tells it to immediately run an R program
  instead of presenting an interactive prompt
- R.out is the screen output
- results.Rout is the R program output
Job Parallelism

- You have a large number of unconnected jobs
- “Pool of Work” or “Bag of Tasks” model
- Parameter space studies
- Many data sets, all of which need to be processed by the same algorithm
- No communication or interaction is needed between Job-#i and Job-#j

- This is often the most efficient form of parallelism possible!
- *IF* you can fit the jobs onto individual machines
- Memory could be a limiting factor
- *AND* it will still take X hours for one job, it’s just that you will get back 10 or 100 results every X hours

Sequential “Parallel” Programming

- For Job Parallelism (pool of work model), you may not have to write “real” parallel code at all
- E.g. you may have (or can generate) 1000 input files, each of which has to be processed on a single CPU

- What input files do you need?
- What output files are produced?
- make sure to name files appropriately to avoid over-writing them
- no keyboard input, no screen output (or use redirection)
Iterative Parallelism

- Iterative parallelism means breaking up a problem where there are large iterative processes or loops
  - Eg. “for” loops in C, “do” loops in Fortran
  - Large matrix-vector problems

Example: Poisson’s Equation:

```
For I=1 to N
  For J=1 to N
    v_new(I,J) = 0.25*( v(I+1,J)+v(I-1,J)+v(I,J+1)+v(I,J-1) )
  Next J
Next I
```

- If N is large, then there is a lot of parallel work that can be done
- Note that v_new(I,J) requires only information at v(I±1,J±1)
- So work on row I=1 is independent of rows I={3,4,5, ...}

Submitting MATLAB or R programs as Array Jobs

- A script that is to be run multiple times
- Only difference between each run is a single environment variable, $SGE_TASK_ID$
- Pre-compute N different input files, or input directories

```
#!/bin/csh
#
#$ -cwd
#$ -t 1-1000
cd dir.$SGE_TASK_ID
matlab -nojvm -nodisplay -r my_program
```

- will run the script 1000 times, first with $SGE_TASK_ID=1, then with $SGE_TASK_ID=2, etc.
- SGE will start as many of the tasks as it can, as soon as it can
Run *Four Local* Workers with a Parallel Computing Toolbox License

- Easily experiment with explicit parallelism on multicore machines
- Rapidly develop parallel applications on local computer
- Take full advantage of desktop power
- Separate computer cluster not required
Parallel for-Loops

\begin{verbatim}
parfor i = 1 : n
    % do something with i
end
\end{verbatim}

- Mix task-parallel and serial code in the same function
- Run loops on a pool of Matlab resources
- Iterations must be order-independent

Parallel for-Loop example

\begin{verbatim}
clear A
d = 0; i = 0;
parfor i = 1:400000000
    d = i*2;
    A(i) = d;
end
A
d
d
A
i
\end{verbatim}
Parallel R Options

- Rmpi offers access to numerous functions from the MPI API, as well as a number of R-specific extensions.
- The snow (Simple Network of Workstations) package provides an abstraction layer by hiding the communications details.

Sample Parallel R program using snow

```r
rm(list = ls())
library("snow")
library("rsprng")
### create a cluster
clusterEvalQ(cl, ...)
clusterExport(cl, "epsilonA")
clusterExport(cl, "epsilonW")
clusterExport(cl, "SIMULATION")
clusterExport(cl, "SIM.PATH")
clusterEvalQ(cl, print(ls()))
#Do job
...
### must always do at the end
stopCluster(cl)
```
Parallel R SGE script

```bash
#!/bin/bash
#
#$ -S /bin/bash -cwd
#$ -l arch=lx26-amd64
#$ -l highprio
#$ -S /bin/bash –cwd
```

```
/usr/bin/lamboot -H -s $TMPDIR/machines
```

```
/usr/lib64/R/library/snow/RMPISNOW CMD BATCH cl_simulation.R results.Rout
```

```
/usr/bin/lamhalt -H
```

Blue Devil Grid GPU cluster

- The BDGPU cluster is a shared set of machines provided by the University, each with one or more Nvidia GT-200 series GPUs.
  - Not a “Beowulf” cluster – just a collection x86-64 Linux boxes
  - Machines have no keyboards and no monitors – must use ssh
  - There is a front-end node, `bdgpu-login-01.oit.duke.edu`, for compilation, job submission, and debugging
  - 17 GPU compute nodes

Machine list

<table>
<thead>
<tr>
<th>Nodes</th>
<th>CPU</th>
<th>#cores</th>
<th>CPU Speed</th>
<th>Mem</th>
<th>GPU (cores, speed, memory)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bddgpu-n01</td>
<td>Phenom II 940</td>
<td>4</td>
<td>3.0 Ghz</td>
<td>4 GB</td>
<td>GTX 260 (2, 3.04 GHz, 896 MB)</td>
<td></td>
</tr>
<tr>
<td>bddgpu-n10</td>
<td>Phenom 9350e</td>
<td>4</td>
<td>2.9 Ghz</td>
<td>4 GB</td>
<td>GTX 275 (2, 1.48 GHz, 896 MB)</td>
<td></td>
</tr>
<tr>
<td>bddgpu-n11</td>
<td>Athlon II 240</td>
<td>2</td>
<td>2.9 Ghz</td>
<td>4 GB</td>
<td>GTX 275 (2, 1.48 GHz, 896 MB)</td>
<td></td>
</tr>
<tr>
<td>bddgpu-n12</td>
<td>Sempron 140</td>
<td>1</td>
<td>2.7 Ghz</td>
<td>4 GB</td>
<td>Tesla C1060 (2, 1.30 GHz, 4 GB)</td>
<td></td>
</tr>
<tr>
<td>bddgpu-n13</td>
<td>Athlon II 620</td>
<td>4</td>
<td>2.6 Ghz</td>
<td>4 GB</td>
<td>Tesla C1060 (2, 1.30 GHz, 4 GB)</td>
<td></td>
</tr>
</tbody>
</table>
BDGPU Filesystems

- Home directory (/afs) - the campus Andrew file system (AFS).
  - can also be mounted directly to your workstation or accessed via a browser: https://webdav.webfiles.duke.edu/~yourNetID
- Scratch directory (/bdscratch)
  - NFS-mounted RAID 0 partition
  - temporary file storage during job execution – not archival
  - create your own subdirectory, copy over files, delete when done

```
mkdir /bdscratch/tm103/job1_scratch
cp -/job1/* /bdscratch/tm103/job1_scratch
cd /bdscratch/tm103/job1_scratch
qsub submit_script (job completes)
rm -rf /bdscratch/tm103/myjob
```

- Applications directory (/opt)
  - All cluster installed applications:
    https://wiki.duke.edu/display/SCSC/BDGrid+Installed+Applications

BDGRID Installed Applications

Bioinformatics
- GPU-HMMER 0.92 /opt/bin/ http://mpihmmer.org/

Math Library
- BLAS 3.0-37 /opt/lib64/libblas.so.3 http://www.netlib.org/blas/
- GPUmat 0.24 /opt/GPUmat http://www.netlib.org/lapack/
- Lapack 3.0-37 /opt/lib64/liblapack.so.3 http://www.netlib.org/lapack/

Math/Statistics
- R 2.10 /opt/bin/R http://www.r-project.org/
- Matlab R2009b /opt/bin/matlab http://www.mathworks.com/

Molecular Dynamics
- VMD 1.8.7 /opt/vmd/bin/vmd http://www.ks.uiuc.edu/Research/vmd/
- AMBER 10 /opt/amber10 http://ambermd.org/

Miscellaneous
- SQLite 3.3.6 /usr/bin/sqlite3 http://www.sqlite.org/
- Boost 1.34.1 /usr/include/boost http://www.boost.org/
Interactive access - Graphical

Linux – connect with ssh 

Windows – connect using X-Win32 (download from www.oit.duke.edu)

Mac – connect with X11 (free from Apple)

---

GPumat CUDA plug-in for Matlab

- GPU computational power can be easily accessed from MATLAB without any GPU knowledge.
- MATLAB code is directly executed on the GPU.
- GPumat speeds up MATLAB functions by using the GPU multi-processor architecture.
- Existing MATLAB code can be ported and executed on GPUs with few modifications.
- GPU resources are accessed using MATLAB scripting language. The rapid code prototyping capability of the scripting language is combined with the fast code execution on the GPU.
- The most important MATLAB functions are currently implemented.
- GPumat can be used as a Source development Kit to create missing functions and to extend the library functionality.
- Supports real/complex, single/double precision data types.
GPUmat example

- Allows standard MATLAB code to run on GPUs.
- Execution is transparent to the user:

```matlab
A = GPUsingle(rand(100)); % A is on GPU memory
B = GPUdouble(rand(100)); % B is on GPU memory
C = A+B; % executed on GPU.
D = fft(C); % executed on GPU

A = single(rand(100)); % A is on CPU memory
B = double(rand(100)); % B is on CPU memory
C = A+B; % executed on CPU.
D = fft(C); % executed on CPU
```

Porting existing Matlab code to GPUmat

- Convert Matlab variables to GPU variables (except scalars)
- The easiest way is to use GPUsingle or GPUdouble initialized with the existing Matlab variable:

```matlab
Ah = [0:10:1000]; % Ah is on CPU
A = GPUsingle(Ah); % A is on GPU, single precision
B = GPUdouble(Ah); % B is on GPU, double precision
```

- The above code can be written more efficiently using the colon function, as follows:

```matlab
A = colon(0,10,1000,GPUsingle); % A is on GPU
B = colon(0,10,1000,GPUDouble); % B is on GPU
```

- Matlab scalars are automatically converted into GPU variables