# Introduction to Alabama Supercomputer

Peng Zeng

Department of Mathematics and Statistics
Auburn University

Spring 2020

# Outline

# Computing Facilities

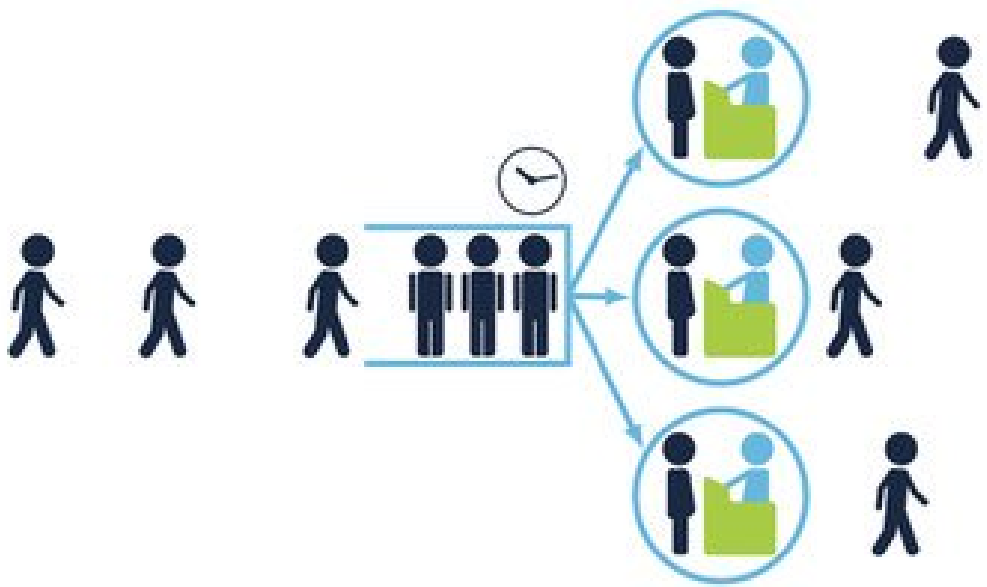
- Personal desktop/laptop
- COSAM
  - virtual machine (windows 10) via VMware
  - {alpha, beta}.auburn.edu
  - {venus, mars, earth}.auburn.edu
- Auburn University
  - CASIC (retired)
  - Hopper
  - Hopper2
- Alabama supercomputer

# Comparison between DMC and Hopper

- Dense Memory Cluster (DMC) at the Alabama Supercomputer Center has
  - 2360 cores
  - 14 TB memory
  - 18 Nvidia GPU
  - 274 TFlops
- Hopper at Auburn University
  - 5888 cores
  - 38 TB memory
  - 2 GPU nodes
  - 1.4 PB disk
  - 250 TFlops

# Queuing System

Request number of cores, size of memory, and computing time when submitting a job.



The more resource your request, the longer waiting time in the queue.

# Check List

In order to use a supercomputer or cluster,

- basic knowledge of linux commands
- secure telnet client (connect to supercomputer)
- secure ftp client (upload/download files)

When submitting a job

- prepare your codes and test it in your desktop/laptop
- specify your resources (core, memory, time)
- check the output or error messages (in files)

# Tools

Tools with graphical interface

- Secure telnet client to connect to supercomputer
  - SecureCRT (download from AU Install)
  - PuTTY
- Secure ftp client to transfer files
  - WinSCP (download from AU Install)
  - Cyberduck (download from AU Install)

Use command lines

- Enable WSL (windows subsystem for Linux) and then install Ubuntu (or other linux distribution) from Microsoft store
- Open a terminal in Mac or linux
- Use command ssh as a telnet client and sftp or scp for transferring files.

# Information

Request an academic account for free at

`https://www.asc.edu/hpc/services/hpc-account-request`

Refer to the following link for documents on supercomputer.

`https://hpcdocs.asc.edu/`

Download the example codes at

`http://www.auburn.edu/~zengpen/tutorials/`

# Login

Using your favorable secure telnet client, login to

`dmc.asc.edu`

When using a terminal, simply type

`ssh user-name@dmc.asc.edu`

# Basic Linux Commands

- `ls`: list the files and subdirectories in the current directory
- `cd`: change directory
- `pwd`: check the current directory

---

- `mkdir`: create a directory
- `rmdir`: remove a directory
- `cp`: copy a file
- `mv`: move a file or subdirectory to another directory
- `rm`: remove a file
- `cat`: display the content of a file
- `nano` or `vim`: text editor

# ASC File Directories

- Home directory: `/home/user-name/`
  - save all your files in this directory
  - type `cd` or `cd ~` to jump to your home directory
- Shared temporary storage: `/scratch/`
  - save temporary files in this directory
  - up to 30 TB by any one user
  - automatically deleted if not accessed in 7 days

# Transferring Files

Using your favorable secure ftp client, login to

`dmc.asc.edu`

When using a terminal, simply type

```
sftp user-name@dmc.asc.edu
```

If you are not familiar with linux commands, edit files in your desktop/laptop and then upload them to `dmc.asc.edu`.

# Example: Singular Value Decomposition

Goal:

- simulate an $n \times n$ matrix $X$ whose entries are iid $N(0, 1)$
- compute the SVD of $X$ and record the computing time

```
nseq = c(100, 500, 1000, 2000);
iter = c(100, 50, 2, 1);
ans = matrix(0.0, nrow = length(nseq), ncol = 5);
for(i in 1:length(nseq)){
  n = nseq[i];
  x = matrix(rnorm(n * n), ncol = n);
  itime = system.time(for(j in 1:iter[i]) tmp = svd(x)) /
    iter[i];
  cat(date(), ': n = ', n, '\n');
  print(itime);
  ans[i, ] = itime;
}
```

# Script Files

Save your R codes as `svd-speed.R`

Create a script file `run-svd-speed` with the following lines

```
#!/bin/sh

# load the paths and MPI
source /opt/asn/etc/asn-bash-profiles-special/modules.sh
module load R

# command to run R
R CMD BATCH svd-speed.R
```

Make the script executable. Type in your terminal

```
chmod +x run-svd-speed
```

# Submit Your Job

Submit the script to the queue system:

```
run_script run-svd-speed
```

Select the queue, time, number of cores, number of memory, and location to run your program following the prompt

| Queue | Wall Time | Mem | # Cores |
|---|---|---|---|
| express | 4:00:00 | 16gb | 1-4 |
| small | 60:00:00 | 4gb | 1-8 |
| medium | 150:00:00 | 16gb | 1-16 |
| large | 360:00:00 | 120gb | 1-128 |
| bigmem | 360:00:00 | 130-500gb | 1-32 |
| knl | 150:00:00 | 94gb | 1-256 |
| benchmark | 24:00:00 | 120gb | 1-64 |

# Check Your Job

- The `squeue` command shows what jobs are running and pending in the queue system.
- Find out why a job isn't running using the command;

```
checkjob_asn <job_number>
```

- Check the information about a job (e.g. max memory, CPU efficiency).

```
jobinfo -j <job_number>
```

- Delete a job using

```
scancel <job_number>
```

# Check Results

Two files will be generated automatically

- `job-name.ijob-number`
- `job-name.ojob-number`

A third file is generated by `R CMD BATCH`

- `svd-speed.Rout`

# Summary

Minimal files to submit your job

- R file that includes all calculations you want to do
- script file that loads module and runs the R file

Some tips

- Write all calculated results to files
- Run your code first in a small scale to assess the computing time and memory

# R Packages

Use install.packages() to install extra packages in R.

Possible warning message when

```
Warning in install.packages("xxxx") :
'lib = "/usr/local/R/3.3.2/lib64/R/library"' is not writable
Would you like to create a personal library
~/R/x86_64-unknown-linux-gnu-library/3.3
to install packages into?  (y/n)
```

# Parallel

Use R package parallel for some simple parallel computing.

```
library(parallel)
detectCores()                       # check num of CPUs or cores
detectCores(logical = FALSE)        # check num of physical CPUs
```

Example: repeat the following procedures 100 times.

- simulate $4000 \times 800$ matrix $X$, whose entries are iid $N(0, 1/n)$
- simulate $y_i \sim Bernoulli(\pi_i)$, where $\text{logit}(\pi_i) = \beta^T x_i$
- fit logistic regression without intercept

# Codes

```
fit_model = function(n, b) {
    p = length(b);
    x = matrix(rnorm(n * p, sd = 1.0/sqrt(n)), nrow = n);
    prob = 1.0 / (1.0 + exp(-x %*% b));
    y = rbinom(n, size = 1, prob = prob);
    fit = glm(y ~ x - 1, family = binomial);
    coef(fit);
}


b = c(rep(10.0, 100), rep(-10.0, 100), rep(0.0, 600));
bmat = matrix(0.0, nrow = 100, ncol = length(b));
for(i in 1:100) {
    bmat[i, ] = fit_model(4000, b);
}

save(bmat, file = 'bmat-single.Rdata');
```

# Updated Codes

```
library(parallel)

fit_model = function(i, n, b) {
    p = length(b);
    x = matrix(rnorm(n * p, sd = 1.0/sqrt(n)), nrow = n);
    prob = 1.0 / (1.0 + exp(-x %*% b));
    y = rbinom(n, size = 1, prob = prob);
    fit = glm(y ~ x - 1, family = binomial);
    coef(fit);
}

b = c(rep(10.0, 100), rep(-10.0, 100), rep(0.0, 600));
ans = mclapply(1:100, fit_model, mc.cores = 2, n = 4000, b = b);
bmat = do.call(rbind, ans);

save(bmat, file = 'bmat-2cores.Rdata');
```

# Results

| cores | 1 | 2 | 4 |
|---|---|---|---|
| CPU efficiency | 98.50% | 95.95% | 90.38% |
| user time (s) | 2997.640 | 2984.208 | 3030.962 |
| elapsed time (s) | 3008.443 | 1505.904 | 785.866 |
| Time efficiency | 100% | 50% | 26% |

# Deep Learning and GPU

- tensorflow and pytorch are installed

```
module load tensorflow/1.14.0
module load tensorflow/gpu_1.14.0
module load pytorch/1.1
```

- submit jobs to nodes with GPU

```
run_gpu myscript
```

Computing time for training MNIST with CNN

- CPU with 4 cores: roughly 70s per epoch
- GPU (Kepler): roughly 9s per epoch

# Conclusion

- Read ASC documents
- Contact `hpc@asc.edu` for help