High Performance Computing

Autumn, 2018

Lecture 19

Imperial College
London

Prasun Ray

6 December 2018

Course info + notes

- Minor corrections on project were posted on Tuesday
- Over the last week+ of the class, keep an eye on one or more of:
    - Announcements section on course webpage
    - Class repo readme file (contains same announcements as on webpage)
    - Online project description

- Frequently push to your *private* bitbucket repo so that you have a backup of your work. If your repo is public, you will receive a 0 on the project.

- We will not use f2py with mpi+Fortran code
    - This can be done using the *mpi4py* module

Plan for last two lectures

- Today:
  - Using clusters at Imperial
  - Amazon cloud computing demo
  - Notes on makefiles

- Tuesday:
  - Overview of GPU computing + demos with Apache Spark and Tensorflow
  - Main takeaways from class

High-end HPC

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660

Historically: cluster computing limited to national labs, research universities

HPC at Imperial

- **Imperial has its own HPC service which supports (and drives) a substantial amount of scientific research**

- **Two distributed-memory machines: cx1, cx2**
- **Two shared-memory: ax3, ax4**

https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/

HPC at Imperial

- Imperial has its own HPC service which supports (and drives) a substantial amount of scientific research

- Two distributed-memory machines: cx1, cx2
- Two shared-memory: ax3, ax4

https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/

ax4:1024 cores, used for genomics and chemistry research

HPC at Imperial

- **Imperial has its own HPC service which supports (and drives) a substantial amount of scientific research**

- **Two distributed-memory machines: cx1, cx2**
- **Two shared-memory: ax3, ax4**

https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/

ax4:1024 cores, used for genomics and chemistry research

cx2: 7000 cores, used throughout science and engineering departments

HPC at Imperial

- How do you use cx1 or cx2?
  - Basic steps:
    - HPC provides you with an account
    - Transfer code(s) to your account on cx2 with *scp*
    - Login to the machine with *ssh*
    - Compile your codes (carefully with the right optimizations!)
    - Create a job submission script (specify number of cores, amount of memory, executable)
    - Submit job with *qsub*

HPC at Imperial

- **Transfer code(s) to your account on cx2 with *scp***

```
$ scp midpoint_mpi.f90 username@login.cx2.hpc.ic.ac.uk:~/lecture19/
```

- **Login to the machine with *ssh***

```
$ ssh username@login.cx2.hpc.ic.ac.uk
```

- **Compile your codes (carefully with the right optimizations!)**

```
cx2: $ module load intel-suite mpi
cx2: $ mpif90 -O3 -o mid.exe midpoint_mpi.f90
```

HPC at Imperial

- **Create a job submission script (specify number of cores, amount of memory, executable)**
- **File *job.script:***

```
#!/bin/sh

# Limit job to 1 hour, 0 minutes, 0 seconds
#PBS -l walltime=01:00:00

# Use 1 node with 8 cpus/node and 4gb memory per node
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=4gb

module load intel-suite mpi

cd $SCRATCH/lecture19/

mpiexec mid.exe
```

- **PBS (portable batch system) is used to manage jobs/queues**
- **Lines in script that start with** #PBS **set job parameters: time limit, number of processors, memory required**

HPC at Imperial

- **Submit job with *qsub:***

```
cx2: $ qsub job_script
```

- **Can then monitor jobs with** qstat –u username
- **Should examine output on cx2, then** scp **the data to a local machine for detailed analysis**

HPC at Imperial

- **Submit job with *qsub:***

```
cx2: $ qsub job_script
```

- **Can then monitor jobs with** qstat –u username
- **Should examine output on cx2, then** scp **the data to a local machine for detailed analysis**
- **This is a very brief overview, have skipped over many details**
- **See *Introduction to HPC at Imperial* short course offered by HPC:**

  https://wiki.imperial.ac.uk/display/HPC/Courses

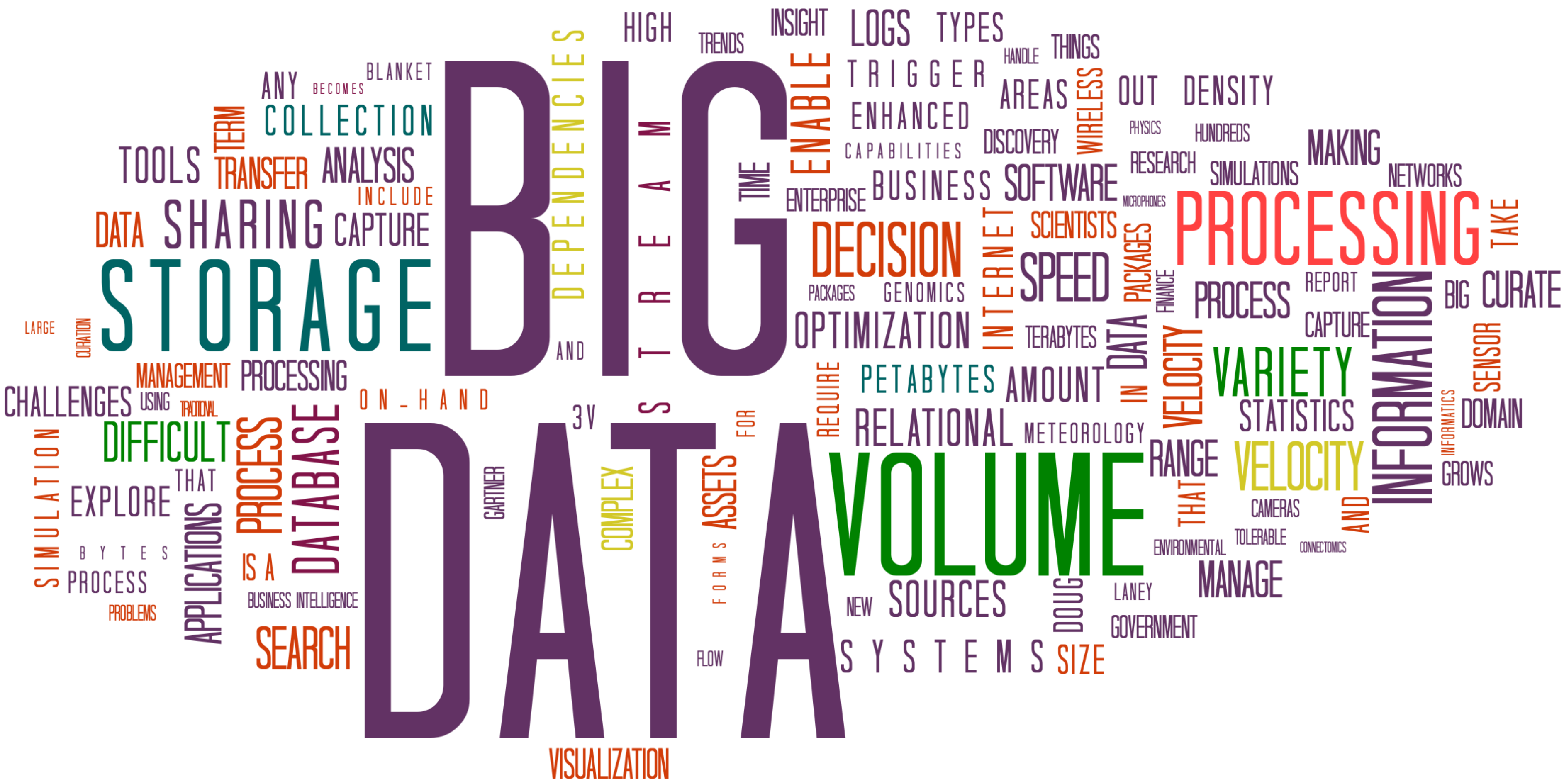
High-end HPC

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,660

Historically: cluster computing limited to national labs, research universities

Cluster computing is mainstream

Big data means big computers!



Cluster computing is mainstream

amazon web services™

Google Cloud Platform

Microsoft | Cloud Services

We're all in.

Cloud computing

- Computing is *outsourced* from our desktops and laptops to remote computing centers
- Can “rent” computational resources by the hour
- If you buy your own cluster, it may sit idle
- Instead you can just use one in the cloud when you need it

Amazon web services

- AWS provide a huge variety of services
- EC2: Elastic cloud computing
- S3: scalable storage
- We will focus on EC2:
  - Create an *instance,* a virtual machine in the cloud
    - Can choose a machine image (AMI), e.g. Ubuntu Linux
  - Many different instance types

Elastic cloud computing (EC2)

- **Many different instance types**

Model	vCPU	CPU Credits / hour	Mem (GiB)	Storage
t2.micro	1	6	1	EBS-Only
t2.small	1	12	2	EBS-Only
t2.medium	2	24	4	EBS-Only
t2.large	2	36	8	EBS-Only

Micro instances are free for 1st year

Elastic cloud computing (EC2)

- **Many different instance types**

Model	vCPU	Mem (GiB)	SSD Storage (GB)	Dedicated EBS Throughput (Mbps)
m4.large	2	8	EBS-only	450
m4.xlarge	4	16	EBS-only	750
m4.2xlarge	8	32	EBS-only	1,000
m4.4xlarge	16	64	EBS-only	2,000
m4.10xlarge	40	160	EBS-only	4,000

You will be charged for any other instances!

See: https://aws.amazon.com/ec2/pricing/

Getting started with EC2

1. Create an account on AWS
   - This requires a credit card!

2. Set up a key-value pair (for appropriate region, e.g. eu-west)
   - This generates a .pem file
   - You use this when you log in (with *ssh*) to your VM instead of a password

3. Set up security group with rule for inbound traffic via ssh

Getting started with EC2

1. Create an account on AWS
   - This requires a credit card!

2. Set up a key-value pair (for appropriate region, e.g. eu-west)
   - This generates a .pem file
   - You use this when you log in (with *ssh*) to your VM instead of a password

3. Set up security group with rule for inbound traffic via ssh

4. Launch an instance!

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html

Launching an instance on EC2

- From AWS site, *sign in to the console*, and click on *EC2*

- Click on *Launch Instance*

  1. Choose your VM (e.g. Ubuntu Server)
  2. Choose instance type (e.g. t2.micro)
  3. Configure instance details (default is ok)
  4. Add storage (default, 8GB hard drive, is ok)
  5. Tag instance (not hugely important)
  6. Configure security group (Type: SSH, Source: My IP)
  7. Review and Launch
  8. Create a new key pair (a .pem file used to log in)
  9. Launch Instance

- Eventually, you will see on the console that your instance is running

- Click on “Connect” for instructions on how to login to your vm

Launching an instance on EC2

- Click on "Connect" for instructions on how to login to your vm
  - After you have used ssh to log in, you have your linux VM in the cloud! (use ssh -Y to be able to open figures/windows)
  - But there's not much software
  - Create a text file, install.sh (using, say, the *nano* text editor) following course webpage:
  http://imperialhpsc.bitbucket.org/software_installation.html
  - Can *scp* files from local machine to cloud VM (as with cx2)
  - But you have git now, so you can just clone repos from bitbucket (or github)!

Makefiles

- **Scientific codes commonly have 50+ subroutines/modules/functions**
- **If you change one subroutine, shouldn't have to re-compile everything**
- ***Make* keeps track of which files have been changed and compiles the code appropriately.**
- **A *makefile* gives *make* information about *dependencies*, e.g. if you change one routine, how many others need to be re-compiled?**

Makefile example: hw2

In homework 2 one fortran module + one python module

Makefile:

```
OBJECTS = hw2.o hw2_main.o
MODULES = nmodel.mod
SOURCE = hw2.f90 hw2_main.f90
```

Makefile example: hw2

In homework 4, two fortran modules + one python test module*:*

Makefile:

```
OBJECTS = hw2.o hw2_main.o
MODULES = nmodel.mod
SOURCE = hw2.f90 hw2_main.f90

# Fortran compilation variables
FC = gfortran
FFLAGS = -O3
LFLAGS = -llapack

OMPF2PYFLAGS = --f90flags='-fopenmp' -lgomp
```

Makefile example: hw2

In homework 4, two fortran modules + one python test module*:*
Makefile:

```
# Fortran compilation variables
FC = gfortran
FFLAGS = -O3
LFLAGS = -llapack

OMPF2PYFLAGS = --f90flags='-fopenmp' -lgomp


.PHONY: clean runhw2py

runhw2py: hw2mod.so hw2.py
	python hw2.py > hw2.txt


m1.so: $(MODULES) $(OBJECTS)
	f2py -c $(SOURCE) -m m1
	mv m1.*.so m1.so
	rm *.mod
```

- **Fortran modules require more care**

Makefile example: hw2

In homework 4, two fortran modules + one python test module*:*
Makefile:

```
# Fortran compilation variables
FC = gfortran
FFLAGS = -O3
LFLAGS = -llapack

OMPF2PYFLAGS = --f90flags='-fopenmp' -lgomp


.PHONY: clean runhw2py

runhw2py: hw2mod.so hw2.py
	python hw2.py > hw2.txt


m1.so: $(MODULES) $(OBJECTS)
	f2py -c $(SOURCE) -m m1
	mv m1.*.so m1.so
	rm *.mod
```

- **Fortran modules require more care**

```
$ make clean
rm -f *.o *.exe *.mod

$ make runhw2py
```

will compile any modified files and then run python code, *runhw2py*

See software carpentry tutorial for further info: http://swcarpentry.github.io/make-novice/