Imperial College London

Introduction to HPC at Imperial

slides - https://tinyurl.com/ic-gs-jan22

Katerina Michalickova – kmichali@imperial.ac.uk
Research Computing Service and the Graduate School
Jan 22, 2020

Imperial College London

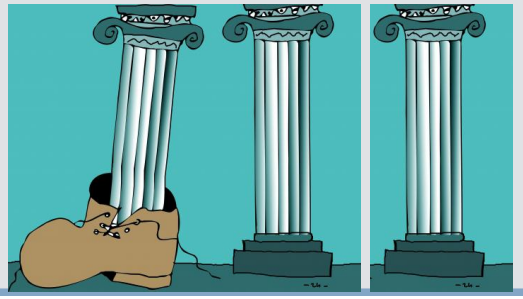
Welcome

I’m here for you for the next two hours.

- Any questions and suggestions are welcome
- Let me know if:
  - I’m going too fast or too slow
  - I’m not audible or
  - The material on the screen is not visible
  - There is anything else that might be hindering the usefulness of the class

Imperial College London

Why are we here today?



Pillars (legs) of science today – theory, experimentation *and computing*.

Much of scientific experimentation such as modeling, simulations or data management and mining is made possible by computing.

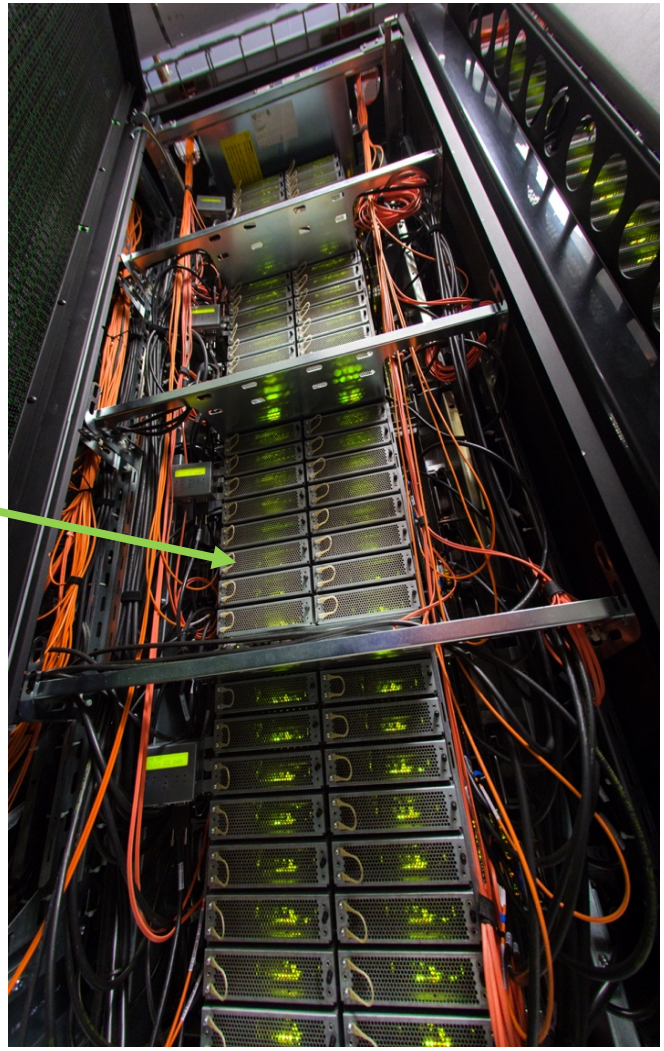
If you need to use a central resource, it is very likely it will come in a form of **high performance computing cluster**.

Computer cluster

A **computer cluster** consists of a set of connected computers that work together so that, in many respects, they can be viewed as a single system.

Inside a rack

a single computer
aka a compute node



Inside a disk rack

All nodes see one central file system – GPFS.

What can clusters do

- the cluster can serve to offload code execution from your laptop/lab server
  - code that runs too long or needs too much memory or disk space
- clusters are particularly useful for executing parallel code
  - on one compute node
  - on multiple compute nodes at once

Note on speed of execution:

- the compute nodes have similar architecture to your desktop - they are not much faster
- the main advantage of cluster computing lies in parallel code execution

Outline

- RCS team, key links and contacts
- hardware
- file management
  - copying files to the cluster
  - filesystems on the cluster
- login
- software
- software execution with PBS queue manager
- job parameters
- job scripts
- serial job
- data parallelism with multiple serial job

Imperial College London

The Research Computing Service team at Imperial

Spencer Sherwin

Director of Research Computing

Matthew Harvey

Research Computing Service Manager

Katerina Michalickova

Training Manager

Bob Cregan

HPC Systems Analyst

Mark Woodbridge

Research Software Engineering Team Lead

Simon Clifford

HPC User Support Analyst

Santiago Lacalle Puig

Research Computing Service User Support Specialist

Mayeul d'Avezac de Castera

Senior Research Software Engineer

Diego Alonso-Álvarez

Research Software Engineer

Chris Cave-Ayland

Senior Research Software Engineer

Imperial College London

Key contacts

Documentation and help - imperial.ac.uk/ict/rcs

User support - rcs-support@imperial.ac.uk, https://imperial.service-now.com/ask (search for RCS)

Service status - https://selfservice.rcs.imperial.ac.uk/service-status

Self-service - selfservice.rcs.imperial.ac.uk

Drop-in clinics - http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/attend-a-clinic/

Getting started - http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/getting-started/

HPC lingo

I will try to avoid or explain again during the class. In case I forget myself, here are few terms that need explanation:

Node = compute node = one computer in the cluster

Job = your program on the cluster

Submit job = instruct the cluster to run your program

Core = the basic computation unit of the CPU - it can run a single process; modern CPUs contain as many as 12 cores each

Serial code = runs on one core

Parallel code = program that runs on two or more cores

Imperial College London

One point of user access to the HPC resources

Components suited to different type of computations:

High throughput (formerly known as cx1)
- ~1700 compute nodes -> ~30K cores
- 12,16,24 core nodes

High end parallel compute (cx2)
- SGI Altix ICEX hardware
- ~ 720 compute nodes -> ~17K cores
- 24 and 28 core nodes

GPUs
- ~200 GPUs (P1000, K80, RTX6000)

Big data (ax4)
- shared memory SGI UV
- 14K cores, 16TB RAM

Imperial College
London

Using the cluster

- file management
  - filesystems on the cluster
  - copying files to the cluster
- login
- software
- software execution

Imperial College London

check your usage with “quota -s”

File management – key environment variables

All computers in the HPC resource are connected to one parallel filesystem – Research Data Store (RDS).

You get access to:

your home directory ($HOME)	• personal working space • 1 TB allocation (up to 10 million files)
temporary storage ($EPHEMERAL)	• additional individual working space • unlimited allocation • files deleted after 30 days from creation data stamp
allocated project space ($RDS_PROJECT)	• your project allocations (if your supervisor has any)

Documentation:
http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/getting-started/data-management/

Imperial College
London

Hands-on – file transfer

Make sure that you can use at least one approach to transfer files from and to the RCS file system.

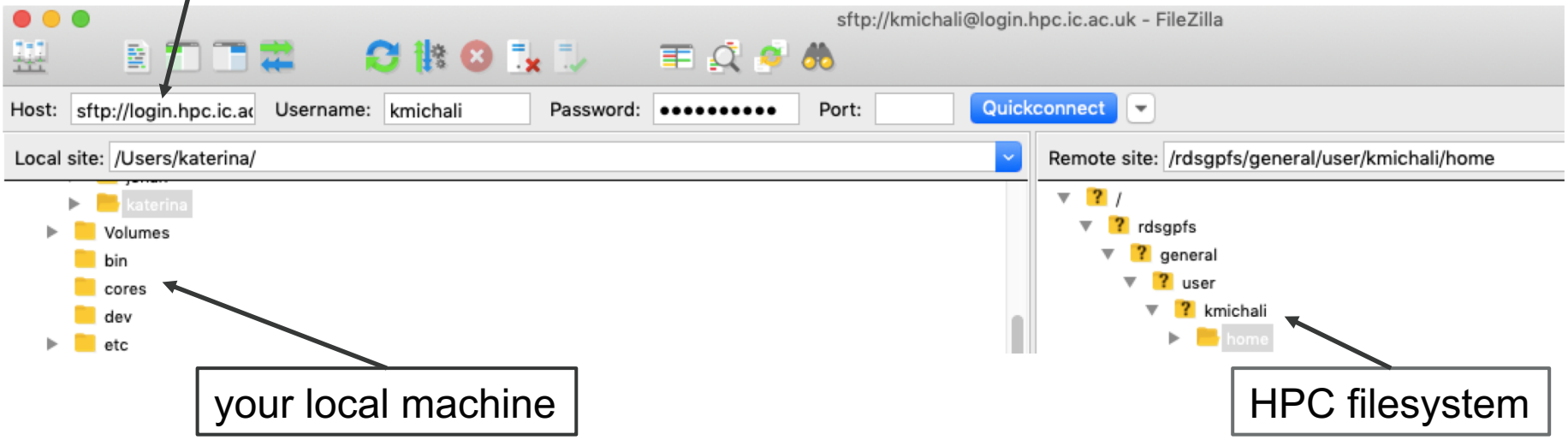
Imperial College London

Copying files from your laptop to HPC and back

- if on laptop, use **Imperial-WPA wifi**
- **FileZilla** file transfer client: https://filezilla-project.org

  type **sftp://login.hpc.ic.ac.uk** into the host window,

  (add username,password, leave port empty, click Quickconnect)



- **WinSCP – good alternative for Windows** https://winscp.net/

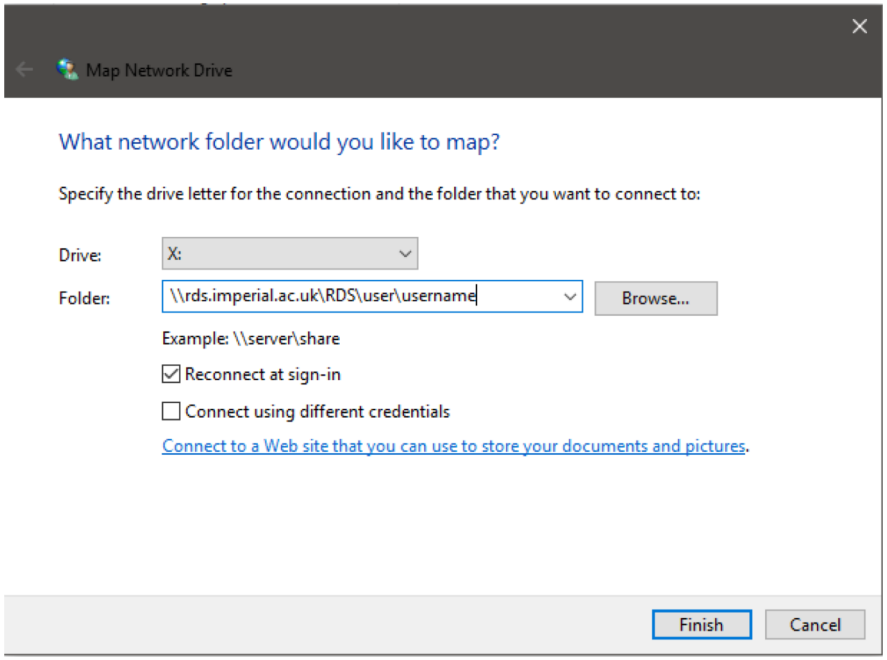
Imperial College London

Connect RDS to your PC using File Explorer on Windows

on Windows 10

File Explorer -> My PC -> Map network drive -> Folder

\\rds.imperial.ac.uk\RDS\user\username



if you don't have College-managed PC, you may need to use "**IC\username**" when loging in.

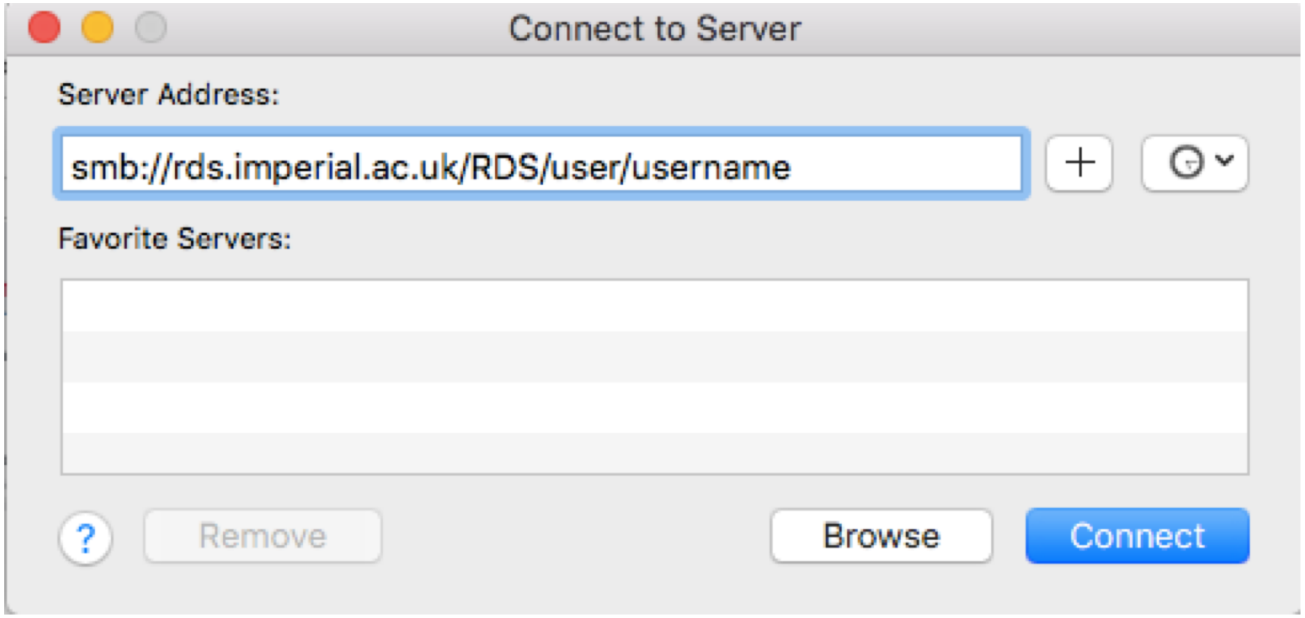
Imperial College London

Connect RDS to your Mac

on a Mac (the latest macOS version)

Finder -> Go -> Connect to Server -> Server Address

`smb://rds.imperial.ac.uk/RDS/user/username`



Imperial College London

Copying files from your Linux or Mac laptop to HPC and back

Secure copy (scp) on the command line (Linux or Mac only)

scp file.txt username@login.hpc.ic.ac.uk:~

(this copies a file – file.txt - from the current directory on your machine to your home directory on RDS)

scp username@login.hpc.ic.ac.uk:~/file2.txt .

(this copies a file from your home directory on RDS – file2.txt – to a current directory on your machine)

Note: The scp command can be used with any directory path:

scp /Users/katerina/Desktop/file3.txt username@login.hpc.ic.ac.uk:/rds/general/user/kmichali/home/projectx

Imperial College London

Before you login

- if on laptop, use **Imperial-WPA wifi**

- you'll have access for the purpose of the class, for regular access, you'll have to register

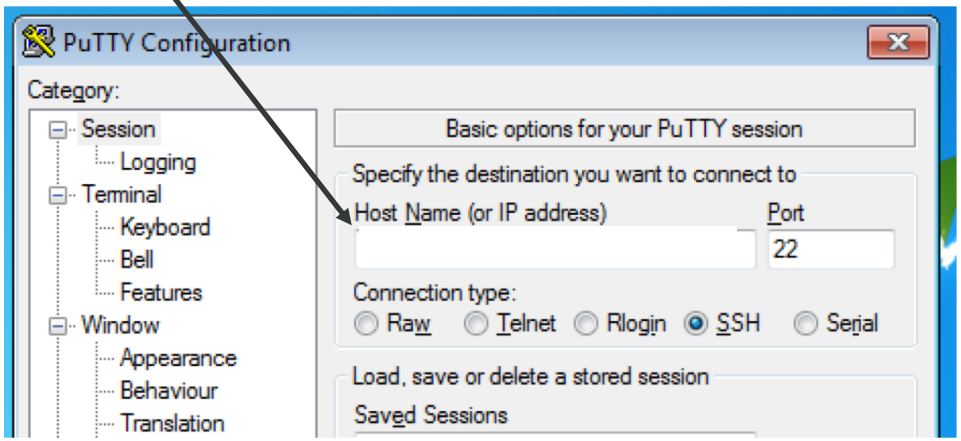
http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/getting-started/

Imperial College London

Hands-on - Log into HPC using secure shell

- **Mac and Linux** – start a terminal and type
  `ssh username@login.hpc.ic.ac.uk`

- **Windows** - start Putty
  `username@login.hpc.ic.ac.uk`



Imperial College London

HPC environment

- Linux operating system
- Bash shell

```
dyn3184-184:~ katerina$ ssh kmichali@login.hpc.ic.ac.uk
Last login: Fri Mar 29 22:02:43 2019 from dyn1251-214.vpn.ic.ac.uk

                Imperial College London Research Computing Service
                --------------------------------------------------

  ***** Continue ONLY if you are an authorised user, otherwise EXIT now *****

Communications, including personal communications, made on or through  Imperial
College's computing and telecommunications systems may be monitored or recorded
to  secure effective  system operation  and for other lawful purposes.  See the
Information Systems Security Codes of Practice for more information.

For any queries or assistance, please visit:

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

Follow our Twitter channel for updates: https://www.twitter.com/imperialRCS

Attend our weekly drop-in clinics held  Tuesday afternoons 14:00-16:00
in the ICT Training Room 204, Central Library, South Kensington Campus


 RDS usage at 15:50 on 3/4/2019

 Individual allocation /rds/general/user/kmichali

   Home       Data:  369GB of 1.00TB (37%)
              Files: 382K of 10.00M (4%)
   Ephemeral  Data:  0GB of 109.95TB (0%)
              Files: 0K of 20.97M (0%)


-bash-4.2$
```

Imperial College London

Module system manages centrally installed packages

- Loading modules sets required environment to use a software package
- List all modules available
  - module avail
- Find a module for your software
  - module avail matlab
- Load a module
  - module load matlab
- Load a specific version
  - module load matlab/2018a
- List all loaded modules
  - module list
- Swap modules
  - module switch matlab matlab/2018a
- Unload a module
  - module unload matlab/2018a
- Get rid of all loaded modules
  - module purge

```
anaconda3/2.4.1
anaconda3/4.1.1
anaconda3/4.3.1(default)
anaconda3/personal
angsd/609
angsd/915
annovar/2013-03-18
annovar/2013-08-05
annovar/2015-06-17(default)
ansys/16.2
ansys/17.1
ansys/18.1
ansys/18.2
ansys/19.1-fluids
ansys/cfx/14.0(default)
ansys/cfx/15.0
ansys/cfx/16.1
ansys/fluent/14.0(default)
ansys/fluent/15.0
ansys/workbench/16.1
ant/1.6.1
ant/1.6.5
ant/1.8.1(default)
ants/2.2.0
ANTs/2015-02-23
aracne/2
armadillo/1.1.90
armadillo/1.2.0
armadillo/2.0.1
armadillo/3.2.4
armadillo/3.4.2
armadillo/6.200.2
armadillo/7.200.2(default)
arpack/96-patch(default)
aspect/1.4.0
aspect/1.5.0(default)
aster/11.3
atat/2.71(default)
ATK/2015.1
magma/2.18-5
magma/2011-07-05
magma/devel-modules(default)
make/3.82(default)
maker/2.10
maker/2.31.9
mamba/1.0.0
mantra/2012-11-19
maple/2016
mapsembler/2.2.4
mapsplice/2.2.0
maq/0.7.1(default)
marv/1.0.4
masurca/2.3.2
masurca/3.2.1
masurca/3.2.1_01202017
matlab/R2017a(default)
matlab/R2017b
matlab/R2018a
matplotlib/0.90.1(default)
mauve/2015-02-13
maxima/5.40.0(default)
mayavi/1.5(default)
mcarts/1.2.0
mcl/14.137
mcr/717(default)
mcscan/0.8
mcstas/2.1(default)
meep/0.10
meep/0.10.1
meep/0.20.3
meep/0.20.4
meep/1.0.3
meep/1.1.1(default)
meep/1.3
meep/1.4.3
meme/4.3.0
meme/4.3.0-nompi
meme/4.9.0
```

Imperial College
London

Cannot find what you're looking for?

- check if the package is available for Anaconda and install yourself

- we also install on demand, submit a request via ICT ASK

https://imperial.service-now.com/ask (search for "RCS Software Install request")

- you can install packages in your home directory

Imperial College London

Setting up python

We distribute personal version of Anaconda environment. Install on the command line using:

```
module load anaconda3/personal
anaconda-setup
```

This is done only once. Afterwards, using "module load ananconda3/personal" will setup your personal python environment.

Anaconda enables you to create separate package environments for your projects. This helps to avoid version and dependency conflicts.

For example, installing scipy in a separate environment:

```
conda create -n projectx python    #create new environment
source activate projectx           #activate env.
conda install scipy                #install scipy
source deactivate                  #deactivate current env.
```

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

Imperial College London

Setting up R

R and libraries can be installed using personal Anaconda.

Setup an a new environment and install R inside it. Packages can be installed using conda install or from within R (plus other libs, e.g. bioconductor-teqc)

```
conda create -n Renv
source activate Renv
conda install r -c conda-forge    #install R in new env.
                                  #using conda-forge channel
conda install bioconductor-teqc -c conda-forge
source deactivate
```

http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/applications/r-including-bioconductor/

Imperial College London

Your command

Assembling the command line is your responsibility, it requires some knowledge of the particular package. We can help with making sure the application runs correctly and advise you on managing your files. We can comment on individual applications to some degree (depending on having the expertise in the group).

Examples:

- **python** python_code.py –i input.txt –o output.txt
- **matlab** -nodesktop –nodisplay –nosplash < matlab_script.m
- **R** CMD BATCH r_script.R
- **bash** shell_script.sh
- **g09** water.com > log
- **blastall** -i query -d fasta_db -o output
- mpiexec **nwchem** input.nw > log

Running software

Do not execute your program from the command line.

Why?

When you log into the cluster, you find yourself on one of six login nodes. If all users compute here, the nodes would crash and the rest of the cluster would be idle. Instead, instruct the queue system to execute your job.

Imperial College London

Queue system

We use PSBPro (Portable Batch System Professional) queue system. PBS takes care of queuing and starting jobs whenever resources become available, monitoring the work and then delivering the results back to the user.

X

Imperial College London

Talking to PBS queue system

- Write **a shell script (or job script)** instructing PBS what to do for you. Passing your finished script to PBS is called “submitting a job”.

- PBS needs to know your **command** and also what **resources** your job needs.

  - Every job script must specify **time**, **memory** and **number of cores** parameters.

  - If the requested resources are exceeded during the job run, the job is terminated.

  - The bigger the resource, the longer the wait for a job to start.

A job script

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

module load hellocx1

hellocx1.py
```

Imperial College London

A job script

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

module load hellocx1

hellocx1.py
```

job params

load software

your command

Imperial College
London

Submit and monitor a job

katerina — ssh kmichali@login.hpc.ic.ac.uk — 100×40

```
-bash-4.2$ cat submit.pbs
#PBS -l walltime=00:15:00
#PBS -l select=1:ncpus=1:mem=2gb

module load hellocx1
sleep 20

hellocx1.py
-bash-4.2$
-bash-4.2$
-bash-4.2$
-bash-4.2$ qsub submit.pbs
805949.pbs
-bash-4.2$
-bash-4.2$
-bash-4.2$
-bash-4.2$ qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
805949.pbs        submit.pbs       kmichali          00:00:00 R v1_debug1
-bash-4.2$
```

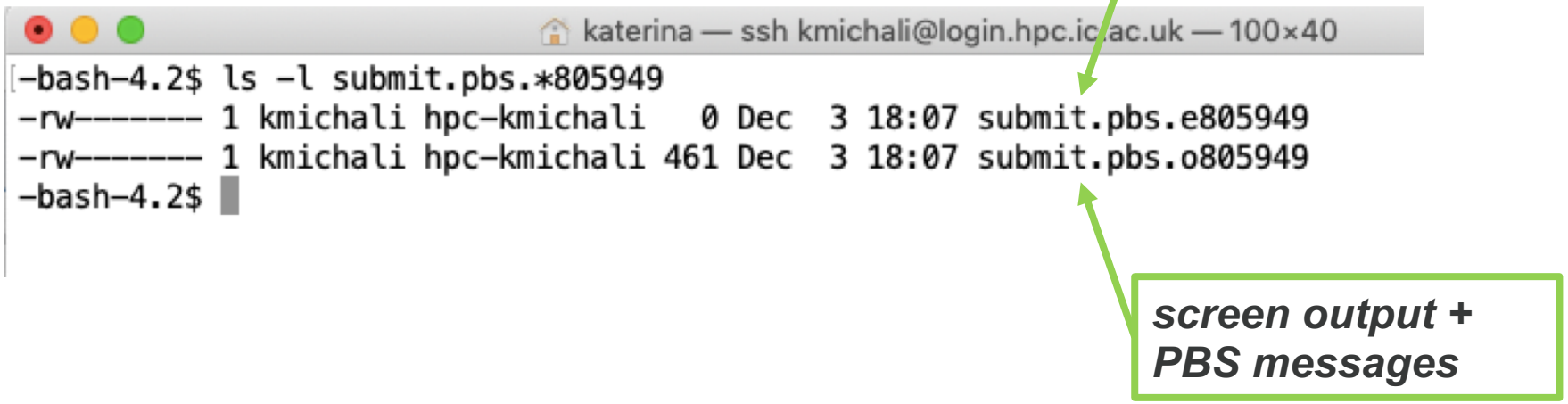
Imperial College
London

Submit and manage your job

submit	**qsub** your_script	system returns a jobid
monitor	**qstat** qstat –f job_id qstat job_id qstat -s	short overview of your jobs long form one job overview job status information
delete	**qdel** jobid	caution: it takes time to see the job disappear from the queue

Imperial College
London

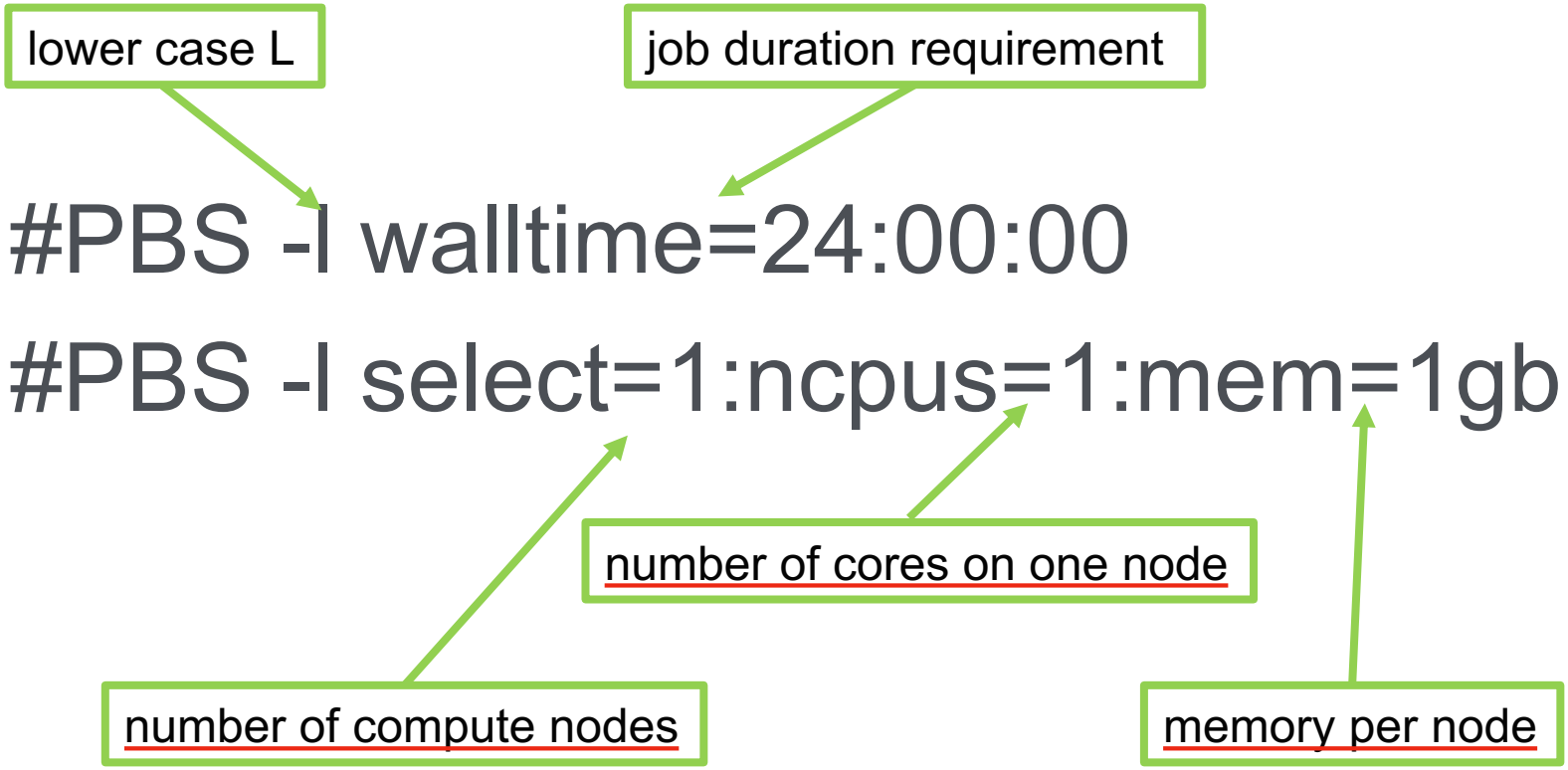
Every job produces two log files



Imperial College London

PBS requirements syntax

The system expects the following syntax at the top of your job script:



Imperial College London

Job duration requirement

- This specifies how long you expect the program to run.
- Sensible values are 30 minutes, 24 hours, 48 hours or 72 hours. There is little benefit in selecting intermediate values.
- In some cases, specified time can be extended using the RCS self-service page. If your job is eligible for an extension, the option will appear on: https://selfservice.rcs.imperial.ac.uk/jobs
- If possible, use checkpointing, i.e. save intermediate results that can be used to restart your program.
- It is possible to run jobs longer than 72h, one at a time.

Memory requirement

- When you run a program, the program and data are read into the computer's memory (RAM). The data is processed in the memory and eventually written out (onto a screen or into a file on the disk).

- PBS needs an approximate information about how much memory (RAM) your program requires.

- Majority of programs that do not deal with lots of data need no more than 2gb of RAM. The data usually determines the bulk of memory requirement. If your program reads large data files, ask for more memory.

- If in doubt, select 2GB to start with. If this is not enough, PBS will terminate your program and place a message (including how much RAM you tried to use) in the **PBS logs** (explanation coming later). In this case, increase the memory requirement and run again.

Imperial College London

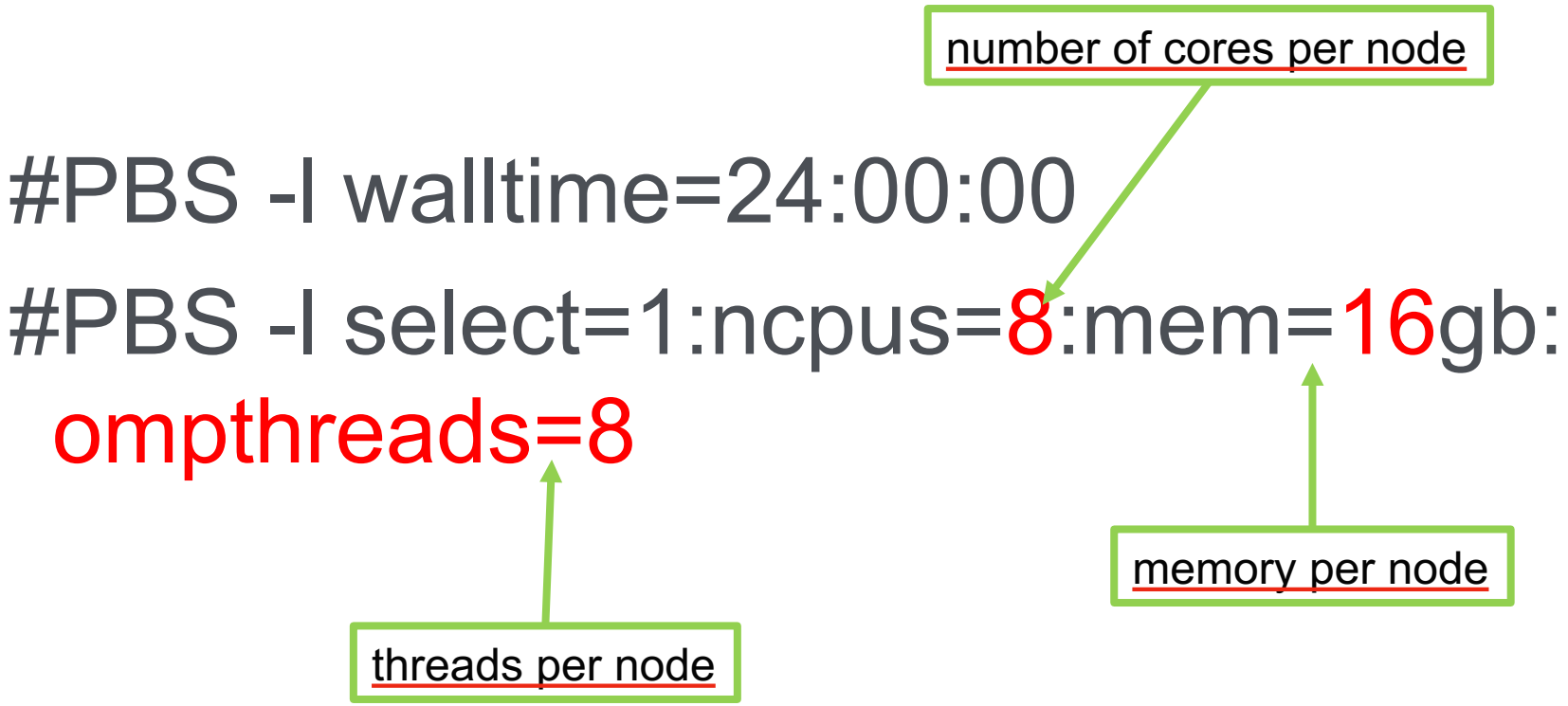
Number of parallel processes

- Each compute nodes has 2 CPUs that in turn have several “cores”.
- Cores a capable of running one process each, this results in each node being able to accommodate 12, 16, 24 or 28 parallel processes depending on the compute node type. This value is often doubled due to hyperthreading.
- You program might be capable of using more than one core at the time. If you are not sure, read the manual, ask your colleagues or check the PBS logs that specify how many cores your program tried to use. Failing that, visit the RCS clinic.
- Bear in mind that, even if your software runs in parallel, it does not automatically mean that it will run much faster. If you’re not sure, benchmark first.

Imperial College London

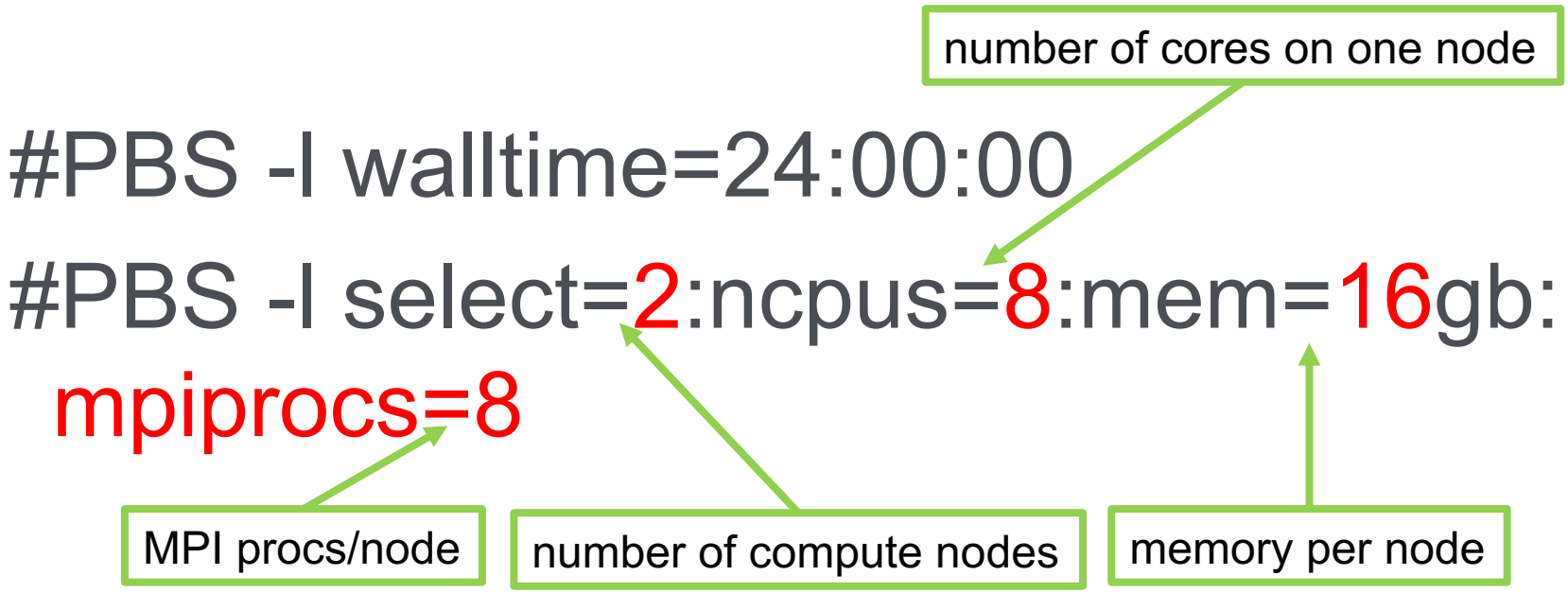
Syntax for threaded parallel jobs

OpenMP (threaded) jobs run on multiple cores on a single node and they require shared memory. Increase ncpus, memory (to accommodate more cores) and add ompthreads.



Syntax for MPI jobs

MPI jobs don't require shared memory and run on multiple nodes. State the number of nodes using "select". The rest of the params (ncpus, mem and mpiprocs) state the requirement per one node.



Imperial College London

Job sizing

make sure your job requirement combination fits one of the following categories

Job class	Number of nodes N	ncpus/node	Max mem/node	Max walltime/hr	Max number of running jobs per user
throughput	1	1-8	96GB	up to 72hr	unlimited for jobs <=24hr in length
general	1 - 16	32	62GB or 124GB	up to 72hr	unlimited for jobs <=24hr in length
singlenode	1	48	124GB	up to 24hr	10
multinode	3 - 16	12	46GB	up to 48hr	unlimited
debug	1	1-8	96GB	up to 30 mins	1
large memory	1	multiples of 10	multiples of 120GB	48 hr	unlimited
GPU	--	--	--	--	Read more
long	1	1-8	96GB	72 - 1000 hr	1

http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/computing/high-throughput-computing/job-sizing/

Imperial College London

Job sizing continued

make sure your job requirement combination fits one of the following categories

Job class	Number of nodes N	ncpus/node	Max mem/node	Max walltime/hr	Number of concurrently running jobs per user
short	1-18	24 or 48	120GB	2hr	2
large	18-72	24 or 48	120GB	48hr	3
capability	72-270	28 or 56	120GB	24hr	1

http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/computing/high-end-computing/job-sizing-/

Job class	Number of nodes	ncpus	Max mem	Max walltime/hr	Max number of concurrently running jobs per user
largemem	1	10**n**	120**n** GB	72hr	50

http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/computing/big-data/job-sizing/

Imperial College London

Jobs using GPUs

GPU type	Number of gpus n	ncpus/gpu X	mem/GB Y	Max walltime/hr
K80, P100, RTX6000	1,2,4,8	4n	24n	24
P1000	1	1-8	96	72

Where **n** is a multiplier matching the number of GPUs requested. The additional options ngpus= and gpu_type= must be added to the PBS selection.

For example, to specify a job using 4 RTX6000 GPUs, the selection must be

#PBS -lselect=1:ncpus=16:mem=96gb:ngpus=4:gpu_type=RTX6000

Within the context of the running job, the shell environment variable CUDA_VISIBLE_DEVICES will be set with indices of the allocated GPUs. Jobs **must** respect this setting, or they will interfere with other jobs co-located on the execution node

http://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/computing/high-throughput-computing/job-sizing/gpu-jobs/

Current working directory for a job

A Job **will start running in a temporary directory** that is created for it. This directory is referred to as $TMPDIR.

This is somewhat counter-intuitive and can lead to file management errors.

For example, if you place your input file in the same directory as your job script and expect your program to read it in a current directory, the file will not be found.

The next few slides show different ways of managing this situation.

Hellocx1 output

Note the job executed in a different directory from where it was submitted.

```
-bash-4.2$ more submit.pbs.o636291
Hello from r1i0n4 on Tue Oct 15 13:17:07 2019.
I'm job 636291.pbs submitted by kmichali from
/rdsgpfs/general/user/kmichali/home/intro_hpc_sep13
and executed in
/rds/general/ephemeral/tmpdir/pbs.636291.pbs.

================================================

        Job resource usage summary

                 Memory (GB)    NCPUs
 Requested  :          1             1
 Used       :          0 (peak)   0.00 (ave)

================================================
-bash-4.2$
```

Job script with input and output files – this will not work

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load myprog

# input and output will not be found

myprog input output
```

Environment variable pointing to your submit directory

$PBS_O_WORKDIR

- will exist when your job is running
- this is where you submitted your job
- usually a directory where your job script resides
- you can use it in your script

Job script for with $PBS_O_WORKDIR 1

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load myprog



myprog $PBS_O_WORKDIR/input $PBS_O_WORKDIR/output
```

Job script for with $PBS_O_WORKDIR - the easy solution

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load myprog


cd $PBS_O_WORKDIR
myprog input output
```

Imperial College London

Hands-on – submit a job script that produces an output file

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

module load hellocx1

cd $PBS_O_WORKDIR

hellocx1.py > mylog
```

Example of a python script (using personal anaconda)

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load anaconda3/personal
source activate projectx

cd $PBS_O_WORKDIR
myscript.py

source deactivate
```

Imperial College
London

Example of a R script (using personal anaconda)

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load anaconda3/personal
source activate Renv

cd $PBS_O_WORKDIR
R CMD BATCH myscript.R
```

Matlab script (no anaconda involved)

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load matlab

cd $PBS_O_WORKDIR
matlab -nodesktop -nodisplay -nosplash < myscript.m
```

Note about $TMPDIR

$TMPDIR holds a path to the temporary job directory

on cx1, it is made on a disk directly attached to the compute node, therefore the speed of reading and writing data is faster

if your job depends on fast reading and writing, $TMPDIR can be used to stage your job and improve the performance

Job script using $TMPDIR

```
#PBS -l walltime=01:00:00
#PBS -l select=1:ncpus=1:mem=1gb

module load myprog

cp $PBS_O_WORKDIR/input $TMPDIR

myprog input output

cp output $PBS_O_WORKDIR
```

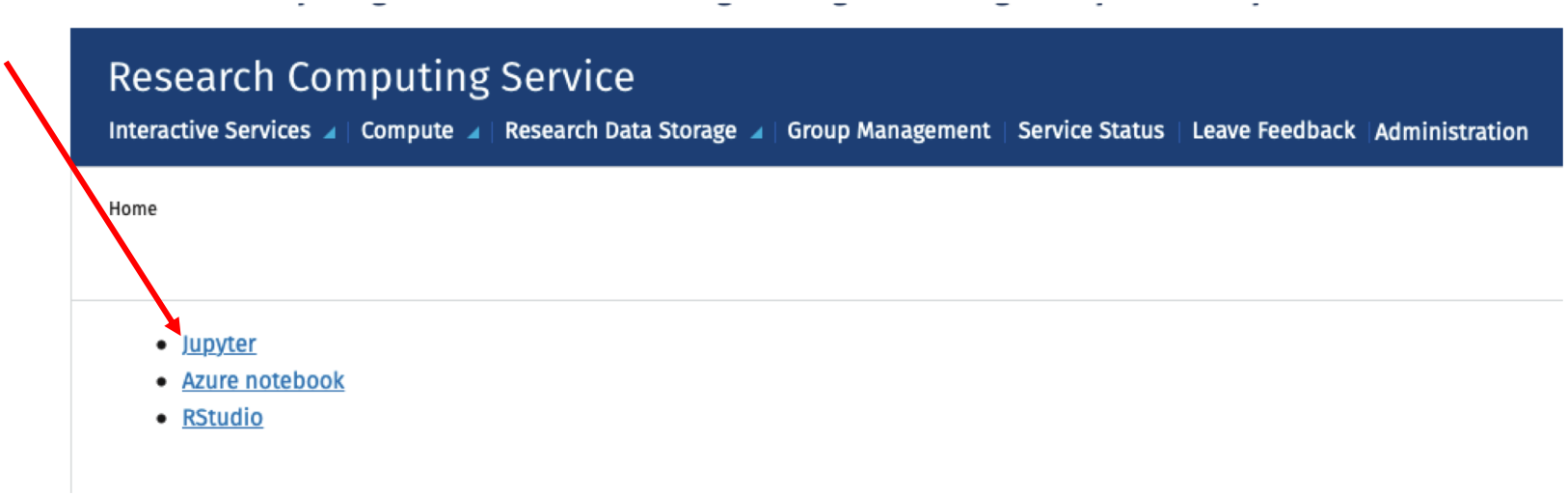
Overview of PBS parameters

-l select=1:ncpus=1:mem=1gb	Specify cpu and memory requirement
-l walltime=HH:MM:00	Specify walltime
-l select=1:ncpus=1:mem=1gb: mpiprocs=1:ompthreads=1	Specification for number of MPI processes and/or OpenMP threads
-N myjob	Give your job a name
-q myqueue	Specify a private job queue
-j oe	Merge STDOUT and STDERR logs
-o myout	Rename STDOUT log
-e myerr	Rename STDERR log

Imperial College London

Using the HPC resource via Jupyter Lab

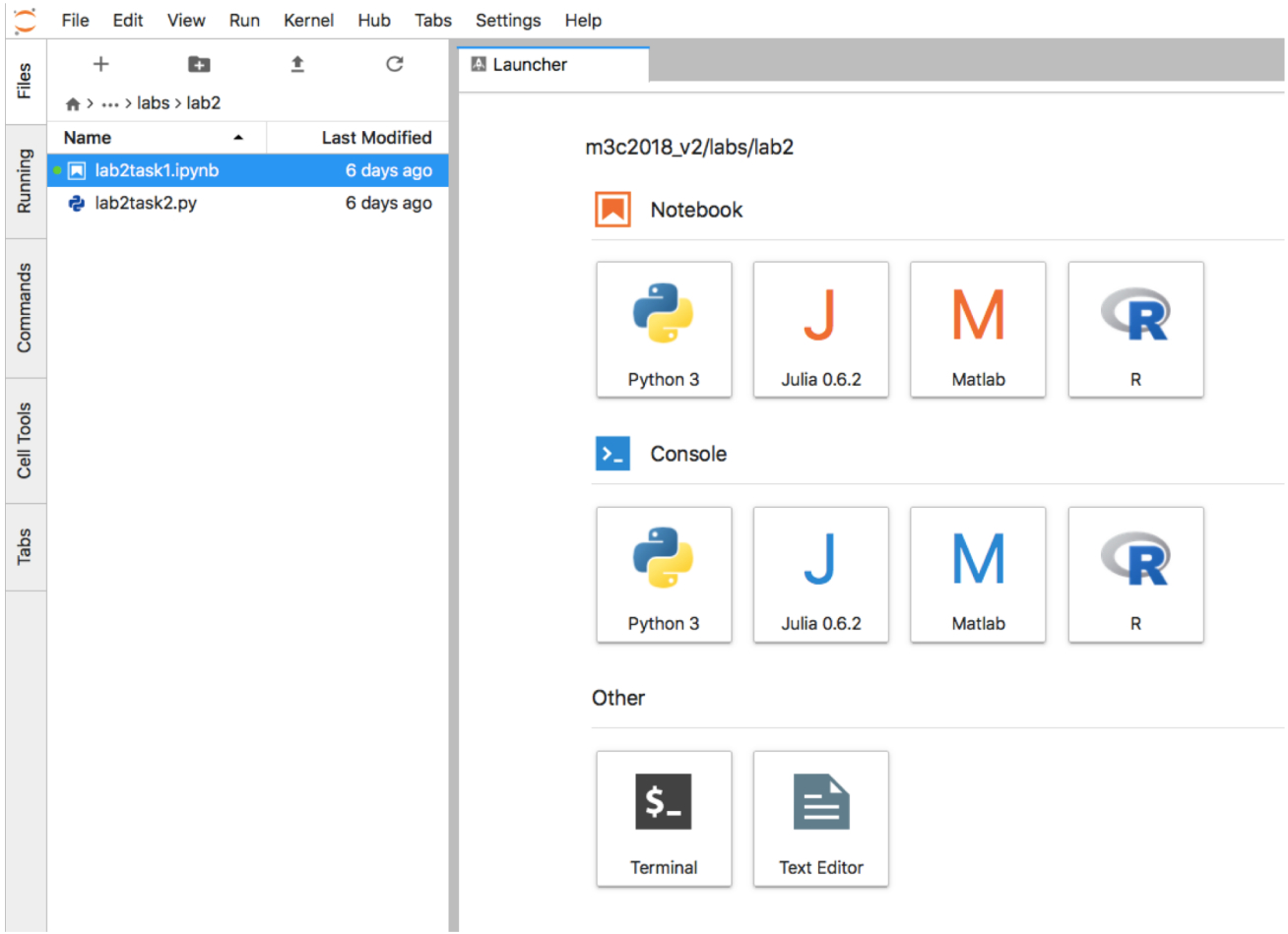
https://selfservice.rcs.imperial.ac.uk/interactive



Imperial College London

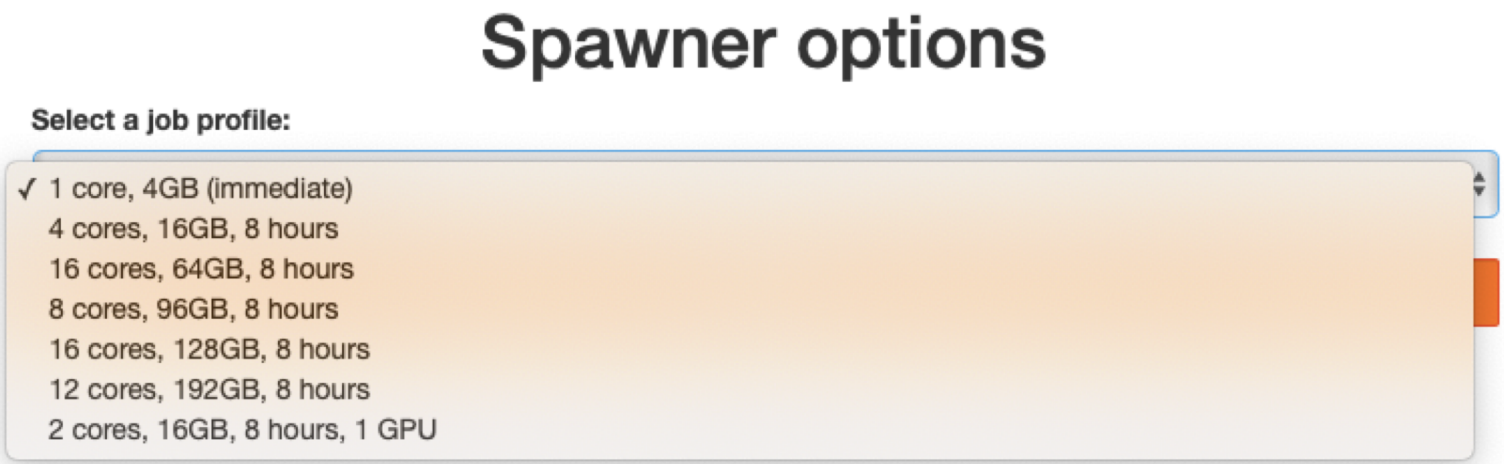
Jupyter Lab

https://selfservice.rcs.imperial.ac.uk/interactive



Imperial College
London

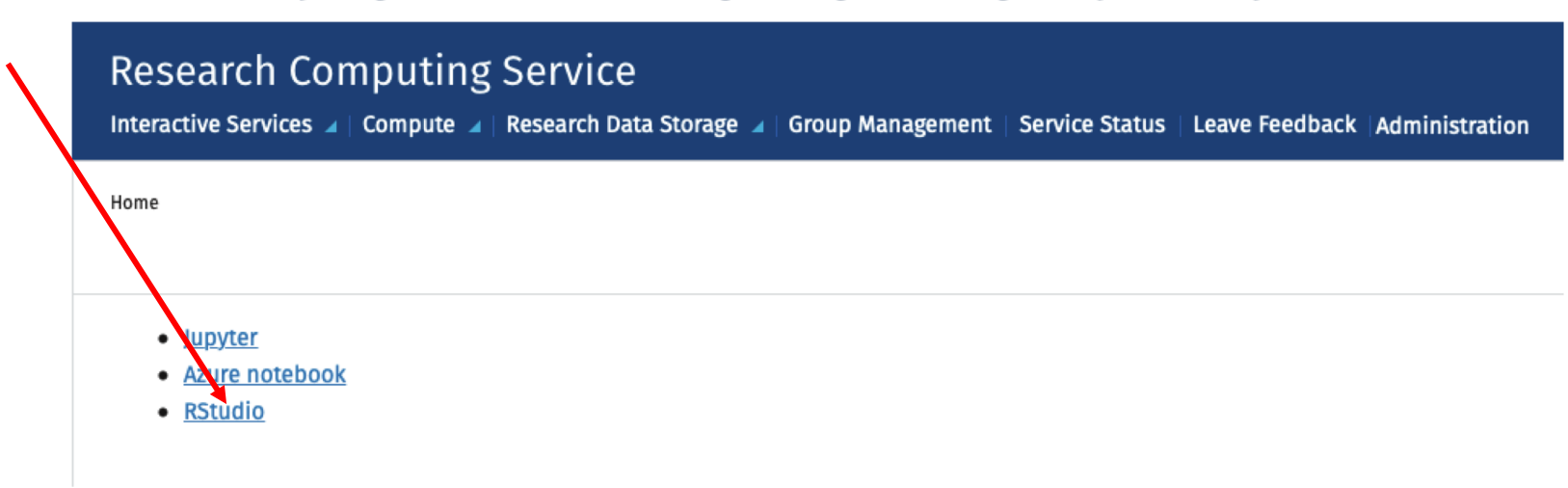
Jupyter lab options



Imperial College London

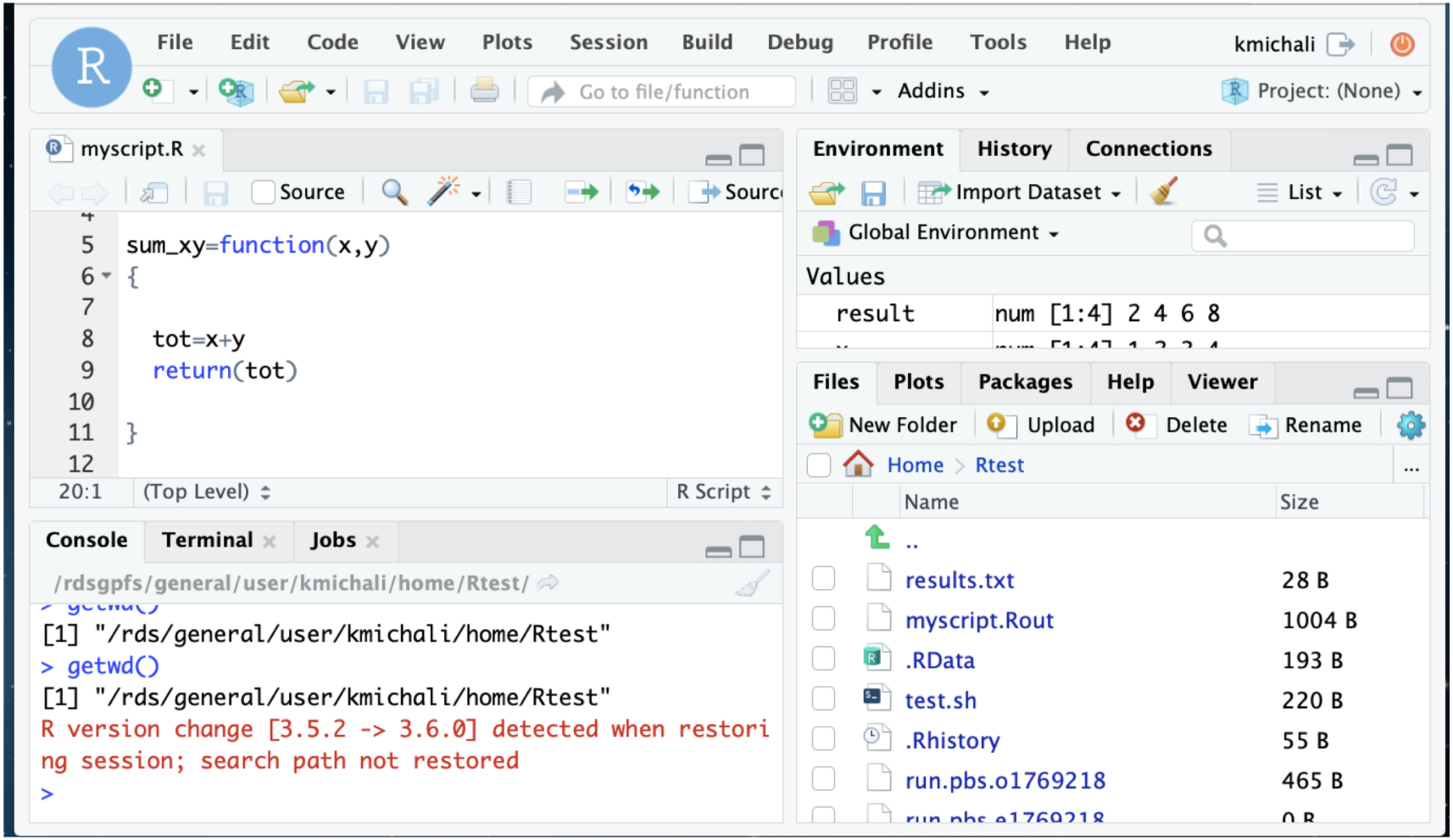
Using the HPC resource via RStudio

https://selfservice.rcs.imperial.ac.uk/interactive



Imperial College London

RStudio interface



Imperial College London

Job status using selfservice

This page will list all your jobs (and also private queues that you are part of)

https://selfservice.rcs.imperial.ac.uk/jobs/qstat

Imperial College London

Troubleshooting your jobs

All standard output (screen and errors - STDOUT and STDERR) from your job plus other useful info is written out by PBS into files in your job directory. **Read the log files** carefully even if all seems fine.

Default names:

STDOUT - e.g. `submit.pbs.o9795758`

STDERR - e.g. `submit.pbs.e9795758`

If asking for help, use the ICT ASK form https://imperial.service-now.com/ask (search for "RCS Job problem")

please provide as much information as your can to help us troubleshoot:

- job id
- steps to reproduce the error
- your script
- the PBS logs (expected and observed)

Imperial College London

Interactive PBS jobs

A job script:

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb

module load hellocx1

hellocx1.py
```

You can ask for the same resource directly on the command line using qsub -I (capital “i”). When you get the prompt back, the PBS will place you on a compute node. You can proceed working directly on the command line (within your resource requirement).

```
qsub -I -l walltime=00:10:00 -l select=1:ncpus=1:mem=1gb
```

Imperial College
London

Parallel computing

- Data parallelism (array jobs)
  - next topic in this class

- Program level parallelism – subject of ”Guide to HPC at Imperial – OpenMP, MPI and hybrid jobs”

  - Threads or multiprocessing - requires shared memory
  - Message Passing Interface – on multiple compute nodes at once

Imperial College London

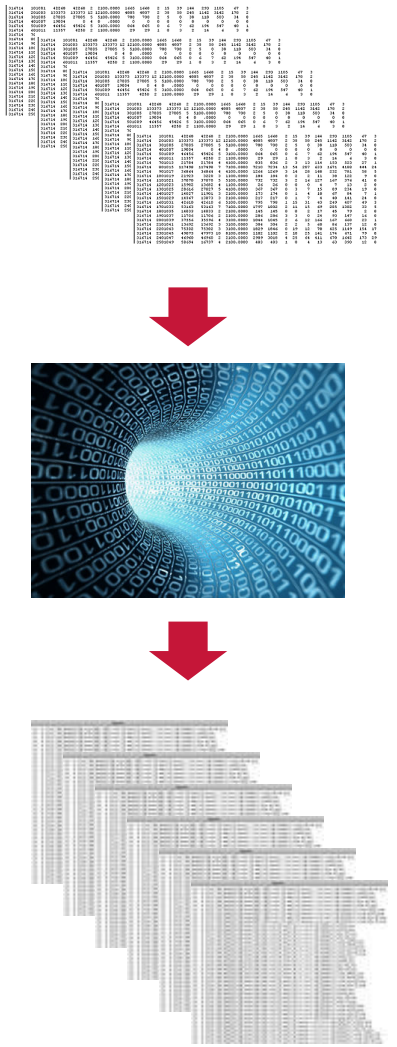
Data parallelism

Data parallelism examples:

- processing multiple files
- processing large files that can be split
- simulation replicates
- parameter sweeps

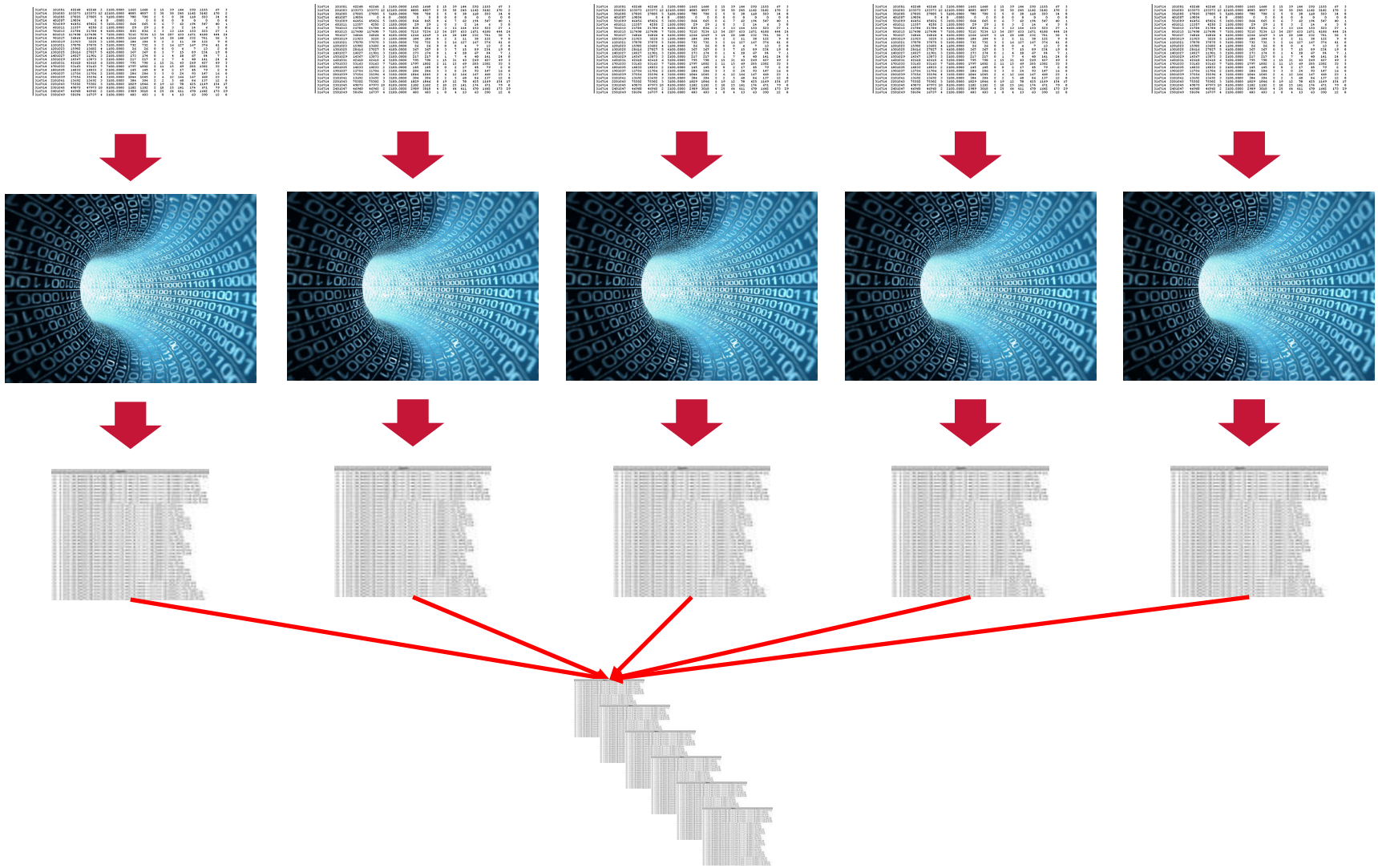
Individual cases have to be independent:

- they do not exchange data
- input for any case doesn't depend on output of the other



Imperial College
London

Data parallelism – array jobs



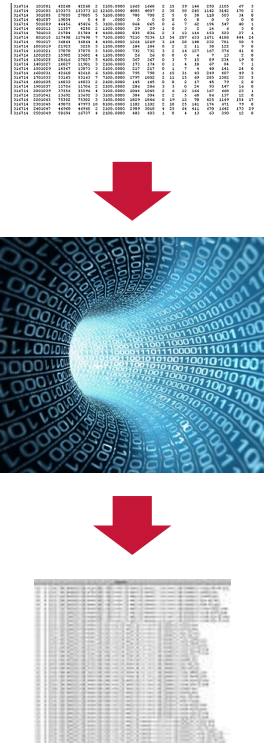
Advantages of data parallelism

- jobs run concurrently (in semi-parallel fashion)
- relatively easy to implement
- PBS jobs run independently whenever a resource becomes available; no need to wait until a whole compute node is available (program-level parallel jobs would have to)

Imperial College London

Single job (before parallelisation)

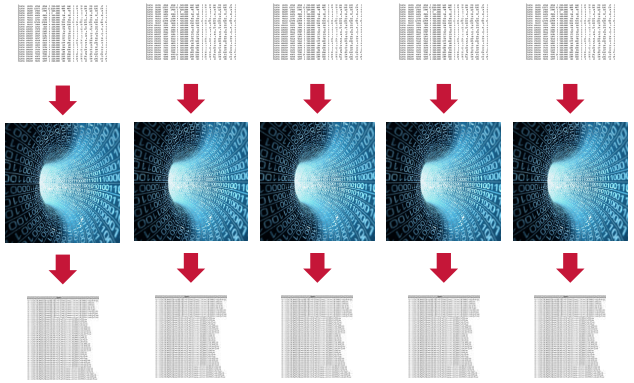
```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -N class


module load myprog

myprog input output
```



Imperial College London

Parallelisation - array job script

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -N class
#PBS -J 1-5

module load myprog

myprog  input? output?
```



Imperial College London

Hands-on – array jobs

All files are available from GitHub

```
git clone https://github.com/kmichali/array_test.git
cd array_test

cd filter
qsub submit_filter_array.pbs

cd ../molecules
qsub submit_list.pbs

cd ../matrix
qsub submit_matrix.pbs
```

Imperial College London

Array job scenario 1 – numbered files

Task: **run filter.py on infile;** BUT the file is too big and takes a long time to process.
Solution: Split infile to smaller chunks and process them concurrently.

```
-bash-4.2$ ll
total 32
-rw-r--r--. 1 kmichali hpc-kmichali 24471 Apr  4 13:56 infile
-bash-4.2$
-bash-4.2$
-bash-4.2$ split -n 3 -a 1 --numeric-suffixes=1 infile infile.
-bash-4.2$
-bash-4.2$
-bash-4.2$ ll
total 32
-rw-r--r--. 1 kmichali hpc-kmichali 24471 Apr  4 13:56 infile
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.1
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.2
-rw-r--r--. 1 kmichali hpc-kmichali  8157 Apr  4 14:13 infile.3
-bash-4.2$
```

Imperial College London

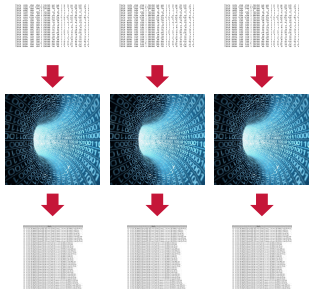
Array job script – numbered files (before parallelisation)

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -N class



cd $PBS_O_WORKDIR
./filter.py infile outfile
```

Imperial College London

Array job script – numbered files

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -N class
#PBS -J 1-3


cd $PBS_O_WORKDIR
./filter.py infile.$PBS_ARRAY_INDEX outfile.$PBS_ARRAY_INDEX
```



$PBS_ARRAY_INDEX takes values from 1 to 3 (or any other value specified by –J)

Imperial College London

Array job scenario 2 – file list

Task: **run stats.py on each molecule;** BUT there are too many molecules and it takes a long time.

Solution: It is possible to process the molecules concurrently.

```
-bash-4.2$ ll
total 3
-rw-r--r--. 1 kmichali hpc-kmichali 1158 Feb 28 13:16 cubane.pdb
-rw-r--r--. 1 kmichali hpc-kmichali  622 Feb 28 13:16 ethane.pdb
-rw-r--r--. 1 kmichali hpc-kmichali  422 Feb 28 13:16 methane.pdb
-rw-r--r--. 1 kmichali hpc-kmichali 1828 Feb 28 13:16 octane.pdb
-rw-r--r--. 1 kmichali hpc-kmichali 1226 Feb 28 13:16 pentane.pdb
-rw-r--r--. 1 kmichali hpc-kmichali  825 Feb 28 13:16 propane.pdb
-bash-4.2$
-bash-4.2$
-bash-4.2$ ls -1 *.pdb > list
-bash-4.2$
-bash-4.2$ cat list
cubane.pdb
ethane.pdb
methane.pdb
octane.pdb
pentane.pdb
propane.pdb
-bash-4.2$
```

Array job script – file list (before parallelisation)

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb


cd $PBS_O_WORKDIR

INFILE=cubane.pdb

./stats.py $INFILE ${INFILE}.out
```

Imperial College London

Array job script outline – file list

```
#PBS -l walltime=0:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-6

cd $PBS_O_WORKDIR

INFILE=$(head -n $PBS_ARRAY_INDEX list | tail -n 1)

./stats.py $INFILE $INFILE.out
```

Array job scenario 3 – for loops and matrices

if you are running simulation replicates or parameter sweep, your program might contain a very time-consuming for loop.

If the loop iterations are independent, there is a good chance that you can deploy it as an array run.

The following Python example uses $PBS_ARRAY_INDEX value directly in the Python code.

The code iterates through a matrix and uses individual elements for a calculation. In the original code, the calculations happen in serial manner.

Python template for matrix iteration

```
input = np.loadtxt(”matrix.txt", dtype='f’, delimiter=',’)
counter = 0

for i in range(0,input.shape[0]):
    for j in range(0, input.shape[1]):
       print('processing element', input[i,j])
       # your code for processing goes in here
```

Array job scenario 3 – for loops

Task: Iterate though a large matrix and use each element for a calculation using an array run.

Solution:

The following Python example uses $PBS_ARRAY_INDEX value directly in the Python code.

The code uses $PBS_ARRAY_INDEX to execute the correct iteration of the loop that processes just one element of the matrix.

All array runs combined process all elements of the matrix in parallel.

Python template for matrix iteration with array run

```
input = np.loadtxt(”matrix.txt", dtype='f’, delimiter=',’)
counter = 0
array_index = int(os.environ['PBS_ARRAY_INDEX'])

for i in range(0,input.shape[0]):
    for j in range(0, input.shape[1]):
        counter = counter+1
        if counter == array_index:
            print('processing element', input[i,j])
            # your code goes in here
```

Submit script for 3x3 matrix

```
#PBS -l walltime=00:10:00
#PBS -l select=1:ncpus=1:mem=1gb
#PBS -J 1-9


module load anaconda3/personal

cd $PBS_O_WORKDIR
python matrix_array.py
```

Summary

You know about:

- Serial and array run job scripts

You can:

- Write and modify a job script
- Submit a serial job
- Submit an array job

Program level parallel jobs

As it has become difficult and expensive to produce faster and faster processors, the focus has shifted in combining the power of multiple CPUs to work on a single task. We supports two main parallel programming paradigms.

OpenMP (C,C++,Fortran) and multiprocessing (Python)

OpenMP jobs require shared memory. The parallelization is achieved by using compiler directives around individual blocks of code, often loops or functions. Converting your serial program into OpenMP is “relatively” easier than attempting MPI programming.

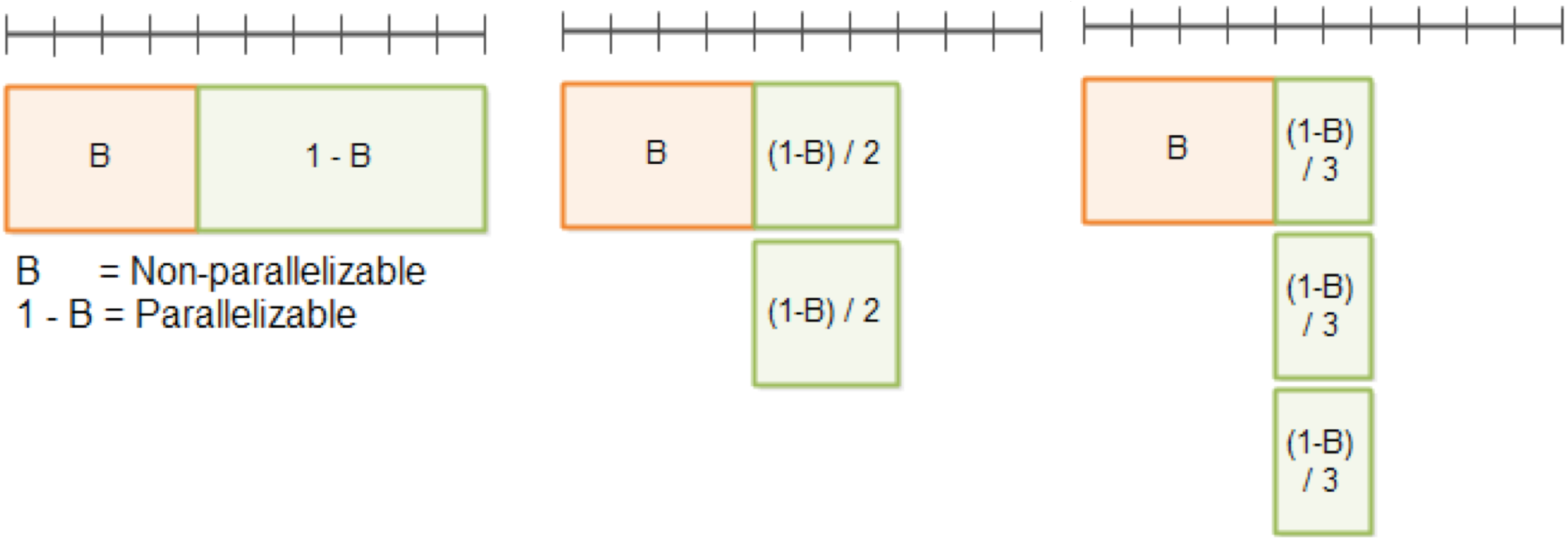
MPI (C, C++,Fortran, Python)

MPI jobs do not require shared memory and can be executed on separate cluster nodes. MPI standard consists of library routines that create parallel processes and pass necessary data (messages) between them. The messages are passed across the network if needed.

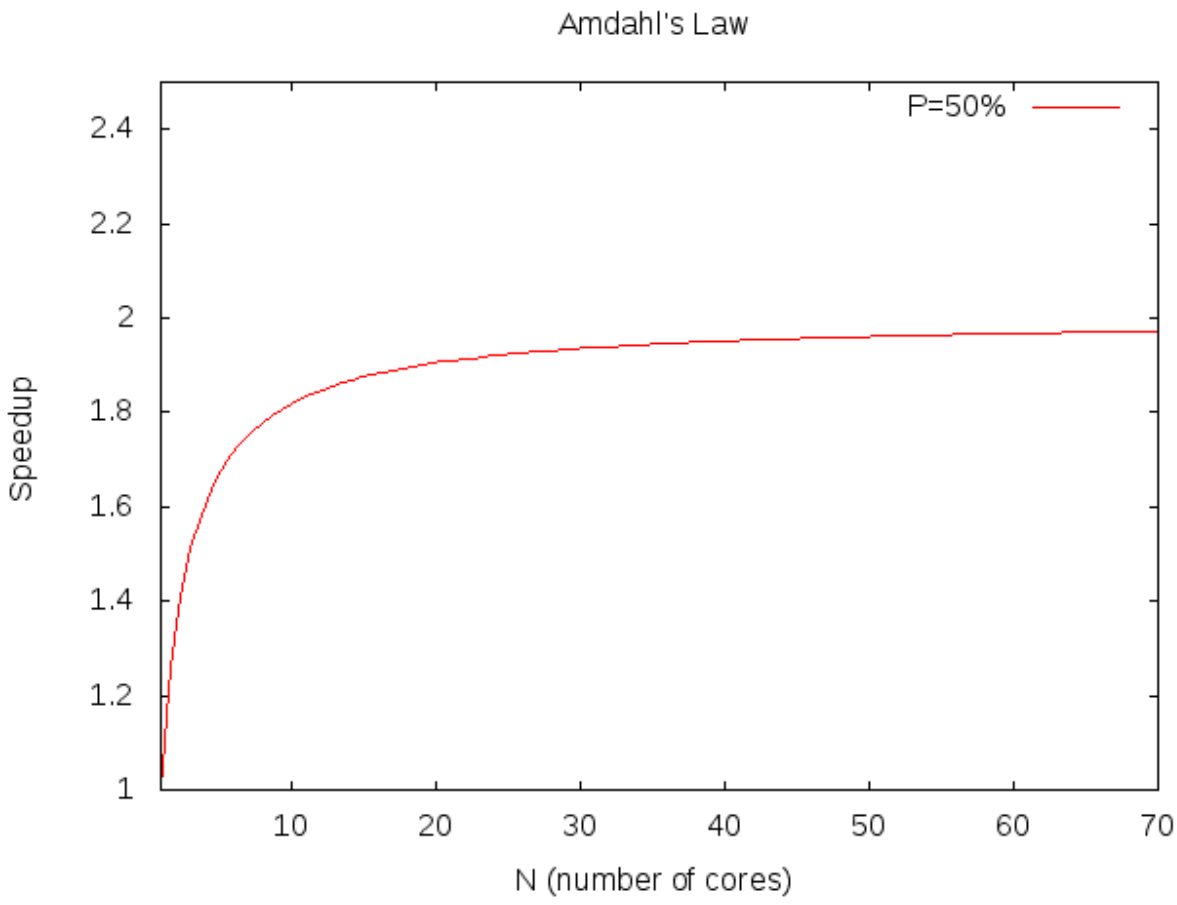
Amdahl's law

Amdahl's law states that in parallelization, if 1-B is the proportion of a system or program that can be made parallel, and B is the proportion that remains serial, then the maximum speedup that can be achieved using N number of processors is 1/(B+((1-B)/N).

If N tends to infinity then the maximum speedup tends to 1/B.

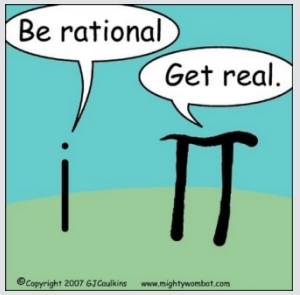


Amdahl's law


Amdahl's Law
P=50%
Speedup
2.4
2.2
2
1.8
1.6
1.4
1.2
1
10
20
30
40
50
60
70
N (number of cores)

Imperial College London

Real problem - approximate Pi



$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

The answer becomes more accurate with increasing N

Iterations over N are independent, the calculation can be parallelised

We'll discuss pi code examples using serial, OpenMP, MPI and various python approaches.

Imperial College London

Serial Pi

$$\frac{\pi}{4} = \int_0^1 \frac{dx}{1+x^2} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{1}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

```
program pi_serial

implicit none
integer(KIND=8)   :: n, i
double precision :: w, x, sum, pi, a, start, finish
double precision :: duration, timef
n=1E10
start=timef()

w=1.0/n
sum=0.0

do i=1,n
   x=w*(i-0.5)
   sum=sum+4.0/(1.0+x*x)
end do

pi=w*sum
finish=timef()
duration=finish-start
print*,n,"   ",pi,"   ",duration
end
```

Imperial College London

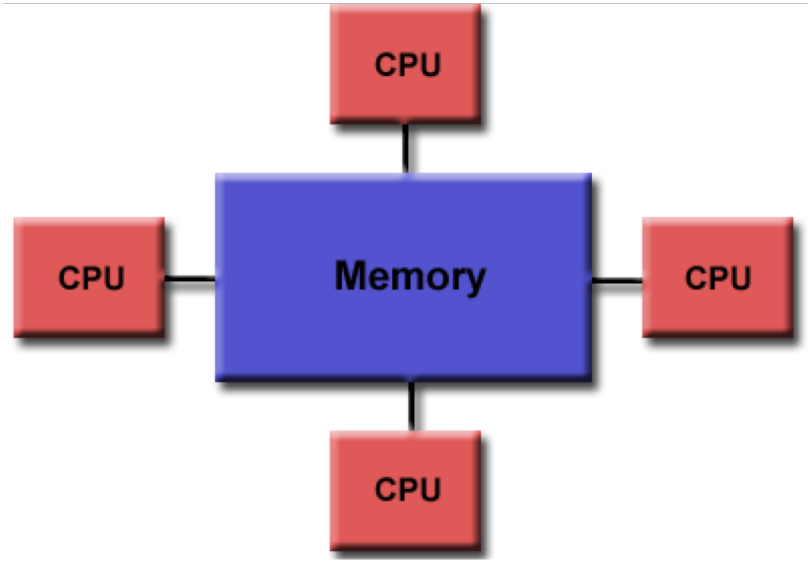
Hands-on – get codes

```
#clone repository
git clone https://github.com/kmichali/parallel_ex.git

# go to repository
cd parallel_ex

# load compiler and parallel library
module load intel-suite
module load mpi

# compile all codes
make

#run serial pi code
qsub pi_serial.pbs
```

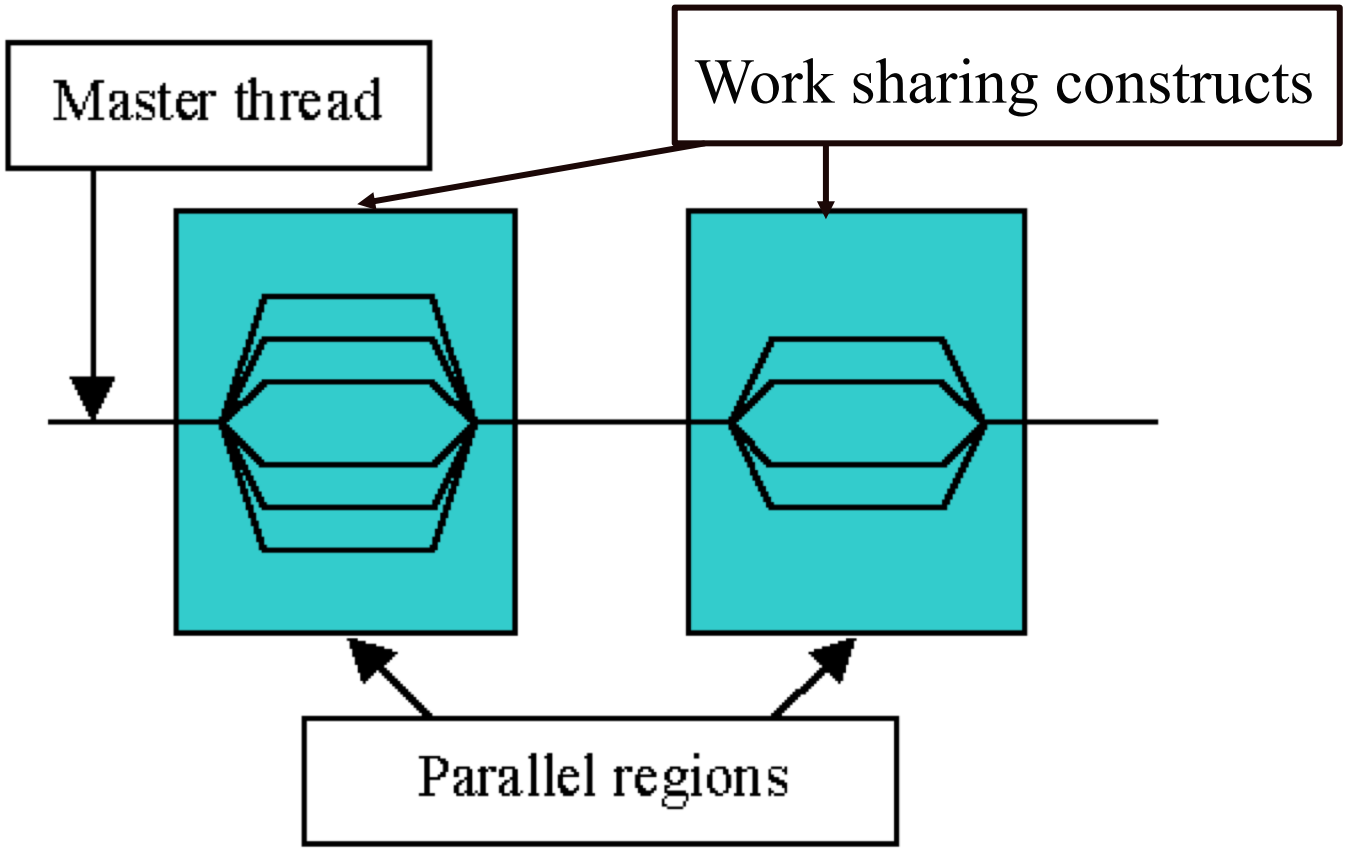
Parallel programming paradigms – OpenMP

OpenMP (open multiprocessing)

- set of extensions (compiler pragmas and API calls) to provide Fortran/C/C++ with the ability to run certain parts of the code in parallel, without explicitly managing (creating, destroying, assigning) threads
- requires shared memory



OpenMP



OpenMP Hello world

```
#include <omp.h>
#include <stdio.h>
main (int argc, char *argv[])
{

  #pragma omp parallel
   {
       printf("Hello world!");
   }
}
```

OpenMP submit script

```
#PBS -l walltime=01:00:00
## Use 1 compute node with 8 cores and 4gb of memory
#PBS -l select=1:ncpus=8:mem=4gb:ompthreads=8


cd $PBS_O_WORKDIR
myprog
```

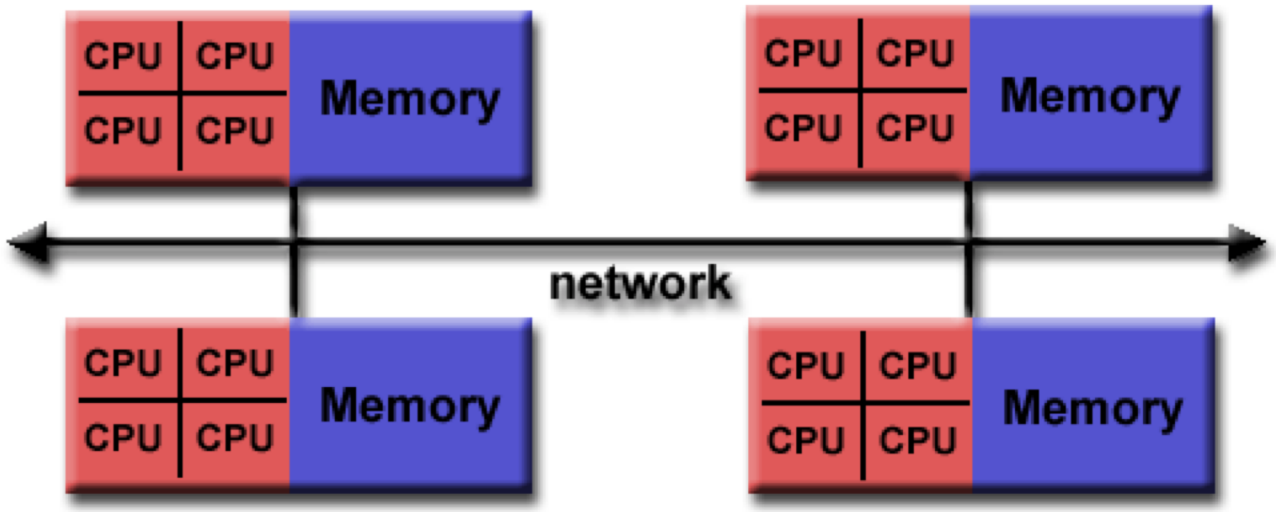
3.14159265358979323846264 3

Hands-on 3– calculate Pi with serial and OpenMP code

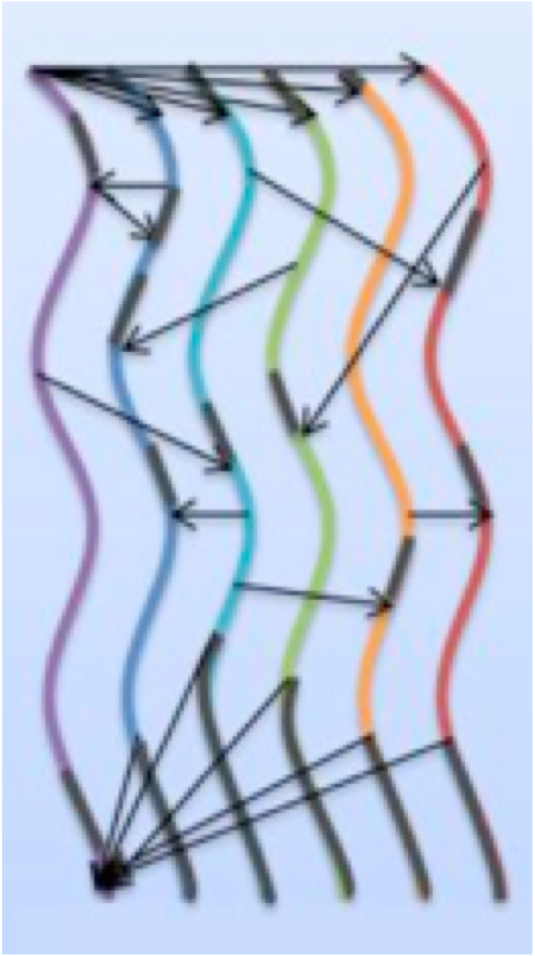
```
# have a look at OpenMP pragmas in
# for_openmp.c

# have a look at pi_openmp.pbs and pi_openmp.f90

qsub pi_openmp.pbs
# examine the results
```

Message Passing Interface - MPI

- message passing parallel programming model
- one program is deployed on multiple nodes with distributed memory
- messages (data) are sent over the network
- MPI is a specification for a library with different implementations (Open MPI, Intel, MVAPICH)



MPI program



Imperial College London

MPI job submit script

```
#PBS -l walltime=01:00:00
## Use 2 nodes with 24 cores each and 4 gb of memory per
   node.
#PBS -l select=2:ncpus=24:mem=4gb:mpiprocs=24

module load intel-suite mpi

cd $PBS_O_WORKDIR
mpiexec myprog
```

Imperial College London

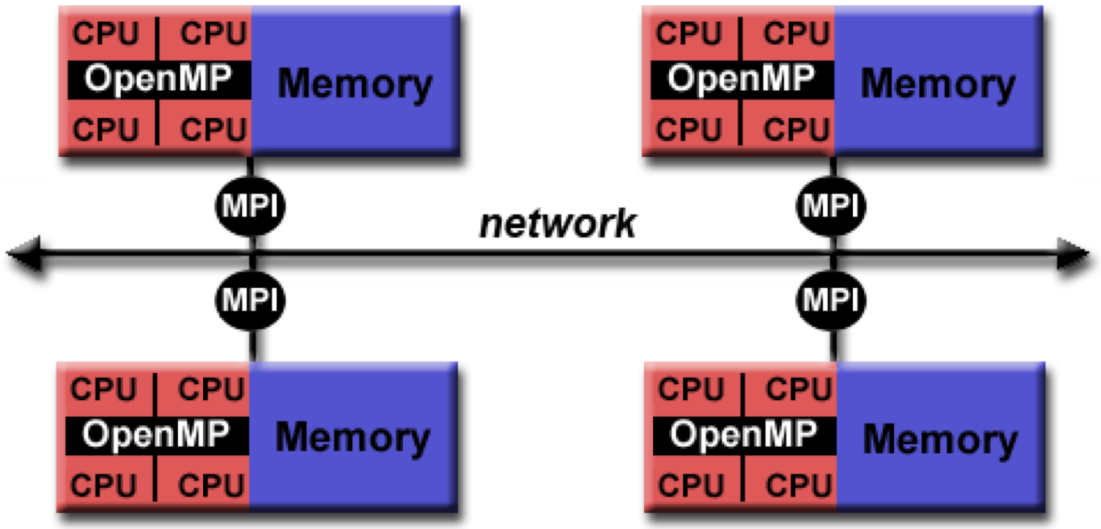
3.14159265358979323846264643

Hands-on 4 – calculate Pi with MPI code

```
# examine hello_mpi.c and hello_mpi.pbs
qsub hello_mpi.pbs
#examine the results

# have a look at pi_mpi.pbs
and pi_mpi.f90

qsub pi_mpi.pbs
# examine the results
```

Hybrid jobs

- combine MPI code with OpenMP code
- MPI provides distributed memory parallelism
- OpenMP provides on-node shared memory parallelism
- the model reduces data movement within a node
- way to combine vectorisation and large scale parallel code



Hybrid submit script

```
#!/bin/sh
#PBS -l walltime=01:00:00
#PBS -l select=2:ncpus=24:mem=4gb:mpiprocs=1:ompthreads=24
## Use 2 nodes with 24 cores each and 4 gb of memory per node.
## Each node will host 1 mpirank and 24 threads.

module load intel-suite mpi

mpiexec $PBS_O_WORKDIR/myprog
```

Imperial College London

3.14159265358979323846264643

Hands-on 4 – calculate Pi with hybrid code

```
# have a look at pi_hybrid.pbs
and pi_mpi.f90

qsub pi_hybrid.pbs
# examine the results
```

Parallel python?

It is possible but not straightforward.

The usual suspects – multiprocessing and MPI – were not the best way to go.

Pi code with python

- serial with python interpreter
- serial with pypy3 interpreter
- serial with just in time compiling
- parallel with numba and jit
- python MPI with mpi4py
- multiprocessing with Pool

Serial pi in Python – python is ~250x slower than Fortran

```
import time

n = int(10e9)

start_time = time.time()

w = 1.0/n
psum = 0.0

for i in range(1,n+1):
    x = w*(i - 0.5)
    psum = psum+4.0/(1.0 + x*x)

pi=w*psum
duration = time.time() - start_time

print(f"{n:,d}", "   ", pi, "   ", duration)
```

Imperial College London

Multiprocessing with Pool does not scale

```
import time
import multiprocessing as mp
import functools
import operator

def calc_sum(i):
    psum = 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
w = 1.0/n
start_time = time.time()

pool = mp.Pool(4)
result = pool.map(calc_sum, range(1,n+1))
total = functools.reduce(operator.add, result)
pi=w*total

print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time)
```

Imperial College London

MPI with mpi4py (196s on 32 cores)

```
import numpy as np
from mpi4py import MPI

n = int(10e9)

comm = MPI.COMM_WORLD
myrank = comm.Get_rank()
nproc = comm.Get_size()

if myrank == 0:
    start_time = time.time()

start = 1+myrank * int(n/nproc)
finish = (myrank+1) * int(n/nproc)

psum = 0.0
pi = 0.0
w = 1.0/n
```

```
for i in range(start, finish + 1):
    x = w*(i - 0.5)
    psum = psum+4.0/(1.0 + x*x)
```

```
psum = np.asarray(psum)
pi = np.asarray(pi)
comm.Reduce(psum, pi, op=MPI.SUM)
pi=w*pi

if myrank == 0:
    duration = time.time() - start_time
    print(n, "   ", pi, "   ", duration)
```

Imperial College London

Numba and jit implemenation (57s on HPC, 16s airbook)

```
import time
from numba import jit

@jit(nopython=True)
def calc_sum(n,w):
    psum = 0.0

    for i in range(1,n+1):
        psum += 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
start_time = time.time()
w = 1.0/n
sum = calc_sum(n,w)
pi=w*sum

print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time
```

Imperial College London

Parallel Numba (7s on 8 cores)

```
import time
from numba import jit, prange

@jit(nopython=True, parallel=True)
def calc_sum(n,w):
    psum = 0.0

    for i in prange(1,n+1):
        psum += 4.0/(1.0 + w*w*(i - 0.5)*(i - 0.5))
    return psum


n = int(10e9)
start_time = time.time()
w = 1.0/n
sum = calc_sum(n,w)
pi=w*sum

print(f"{n:,d}", "   ", pi, "   ", time.time() - start_time)
```

Imperial College
London

Serial and parallel python – pi with 10 billion iterations

- serial with python interpreter – 135 minutes (HPC)
- serial with pypy3 interpreter – 192s (HPC), 122s (airbook)
- serial just-in-time compiling – numba and jit – 57s (HPC), 16s (airbook)

- parallel with numba and jit – 7s (HPC, 8 cores), 8s (airbook, 4 cores)
- python MPI with mpi4py – 196s (HCP, 32 cores)
- multiprocessing with Pool – does not scale
- modified multiprocessing with Pool – 106s (HPC, 8 cores)

Imperial College London

Thank you!

I welcome your feedback at kmichali@imperial.ac.uk.

All classes are listed at http://tinyurl.com/ichpcclass