National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Module 6: Parallel processing large data

Thanks to all contributors:

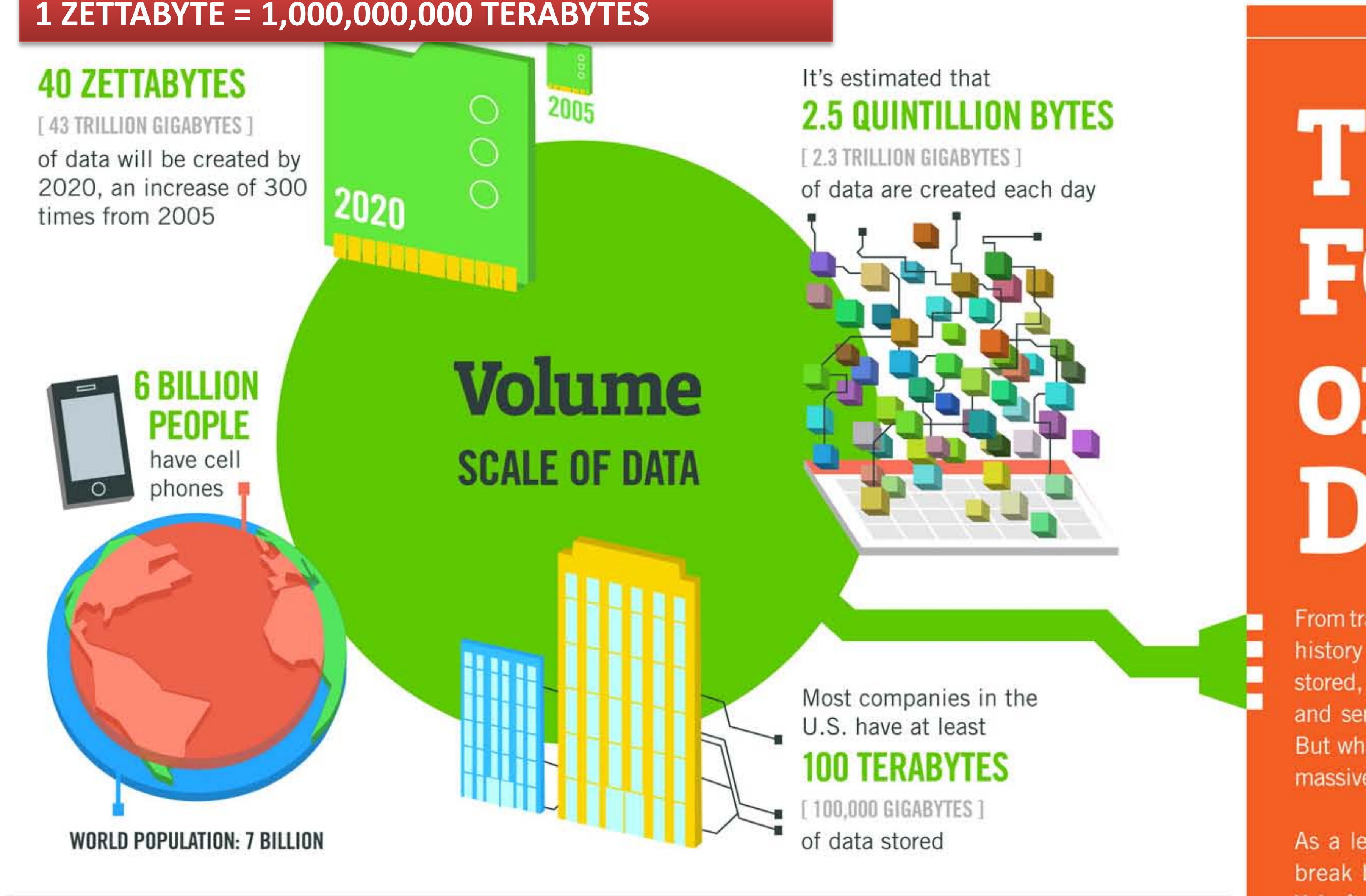
Alison Pamment, Sam Pepler, Ag Stephens, Stephen Pascoe, Kevin Marsh, Anabelle Guillory, Graham Parton, Esther Conway, Eduardo Damasio Da Costa, Wendy Garland, Alan Iwi and Matt Pritchard.

Overview of presentation

- Traditional parallel processing and parallelising data analysis
- Parallel processing on JASMIN / LOTUS
- Examples of running parallel code (on LOTUS)
- Re-factor your code for efficiency

What is Big Data?

1 ZETTABYTE = 1,000,000,000 TERABYTES



Processing big data: the issues

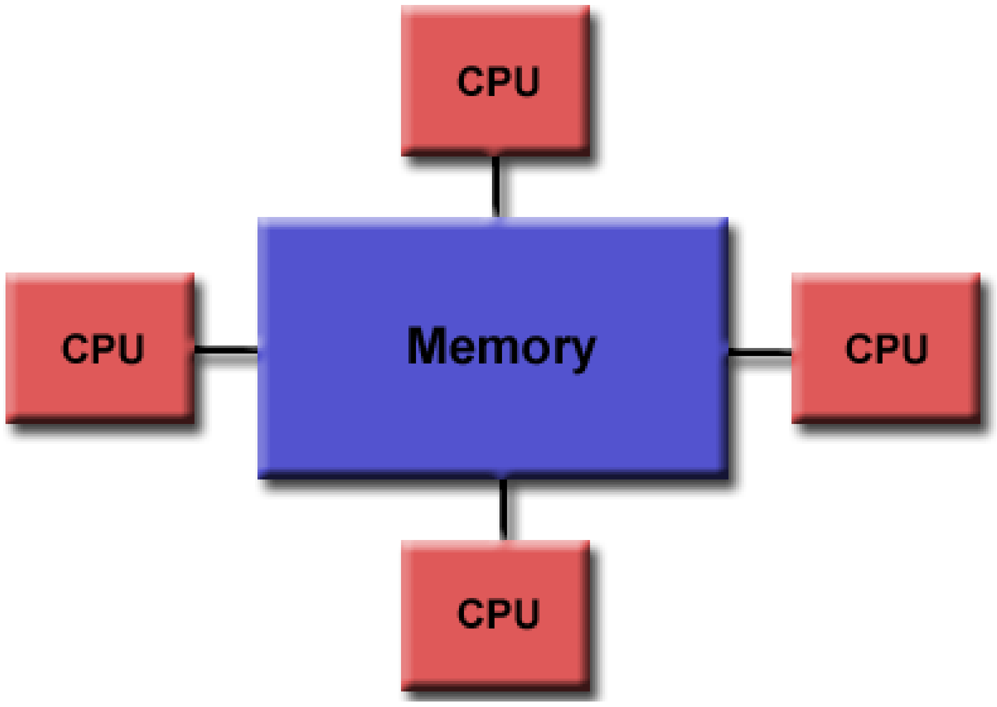
- Parallel processing in the Environmental Sciences has historically focussed on **highly-parallel models.**
- Data analysis was typically run **sequentially** because:
  - It was a smaller problem
  - It didn't have parallel resources available
  - The software/scientists were not equipped to work in parallel
- But now we generate enormous datasets (e.g. UPSCALE – around 300Tb) means that:
  - Processing big data **requires** a parallel approach, but
  - Platforms, tools, and programmers are becoming better equipped

The traditional view of parallel processing

- It’s what models do on High Performance Computing (HPC) platforms
- Modellers may work with…

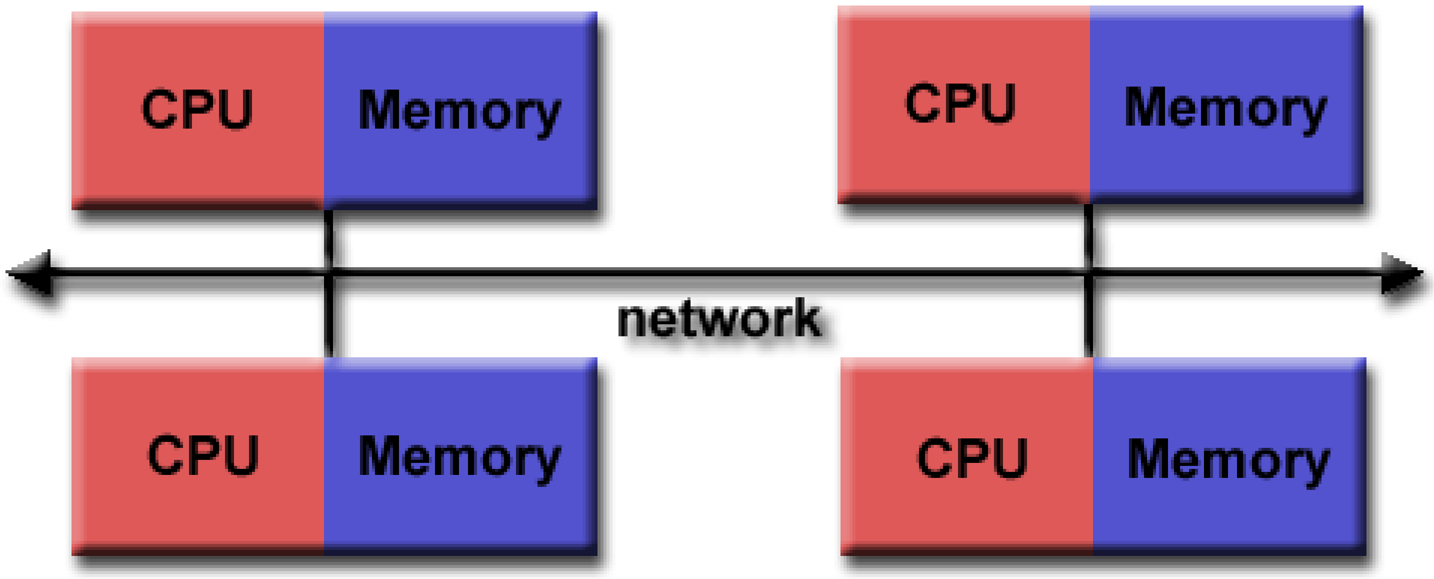
The traditional view of parallel processing

… shared memory (e.g. OpenMP)



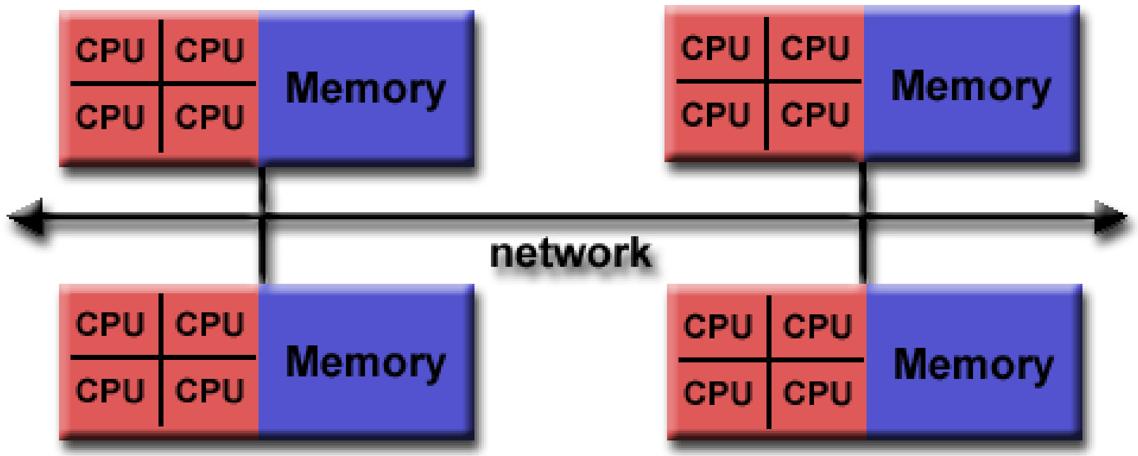
The traditional view of parallel processing

… distributed memory (e.g. MPI)



The traditional view of parallel processing

… or shared distributed memory



*It's a complex subject...**which we mainly intend to avoid!***

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Some Terminology

Concurrency: A property of a system in which multiple tasks that comprise the system remain active and make progress at the same time.

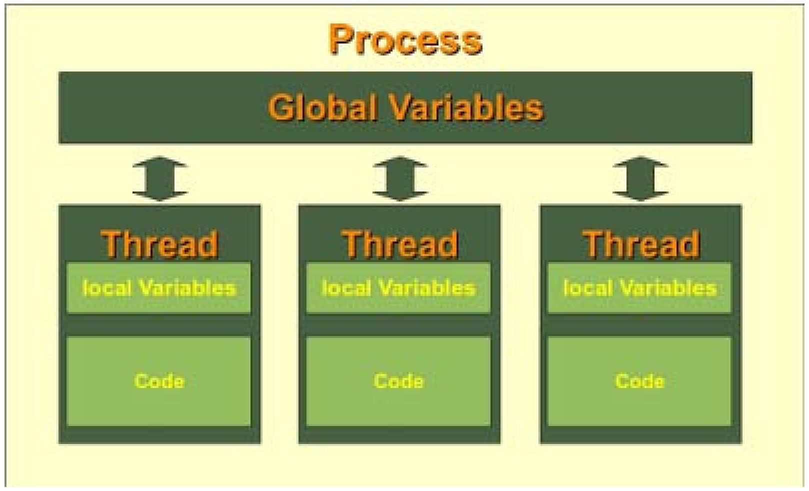
Parallelism: Exploiting concurrency in a programme with the goal of solving a problem in less time.

Race condition: A race condition occurs within concurrent environments: where the output is dependent on the sequence or timing of other uncontrollable events. It can lead to *bugs* when parts of the code complete in an unexpected order.

How does a single processor do three things at once?

Even on a **single processor** modern operating systems can give the **illusion that multiple tasks are running at the same time** by rapidly switching between many active threads.

This is because the modern CPU clock is measuring time in **nanoseconds** whereas we can only keep track of **milliseconds**.



Picture: http://www.python-course.eu/threads.php

(Almost) everything is parallel these days

YOUR DESKTOP MACHINE IS A PARALLEL COMPUTER!

It runs a multi-core processor...

...which means you can speed up processing by asking different parts of your programme to run on different cores.

*"But what about **race conditions**?"...*

...True: you still need to design your approach to avoid things getting out of hand!

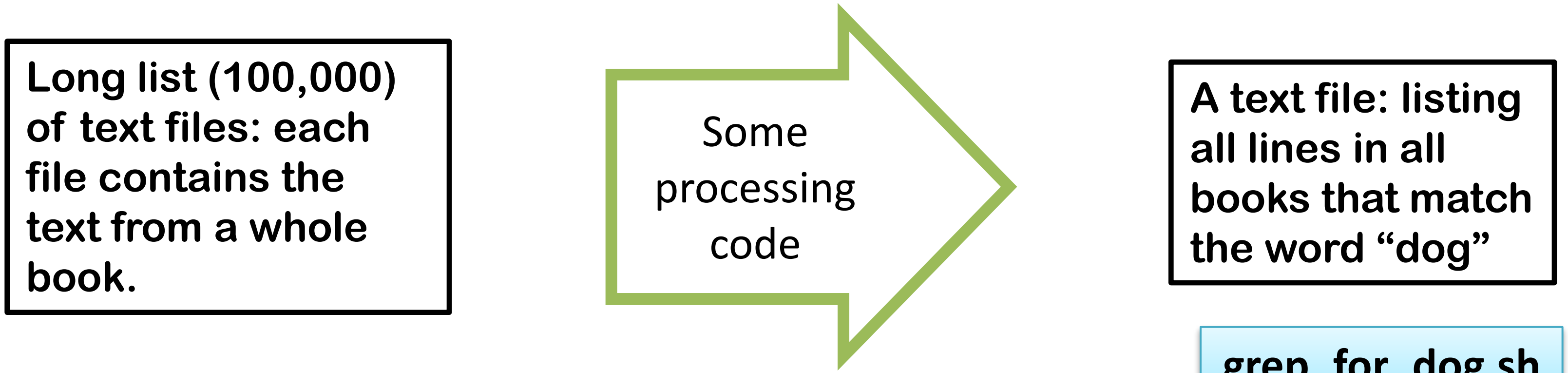
Parallel processing for data analysis

- Data analysis tools do **not** (typically) do parallelisation automatically.
- But parallelisation is normally achievable at a small price.
- A lot can be done with:
  - Batch processing
  - Decomposition of large jobs into smaller jobs
  - Understanding tools and schedulers

We will look at these and show examples.

Simple parallelism by hand (1)

- Running on a multi-core machine you can exploit local processes, e.g.:



Long list (100,000) of text files: each file contains the text from a whole book.

Some processing code

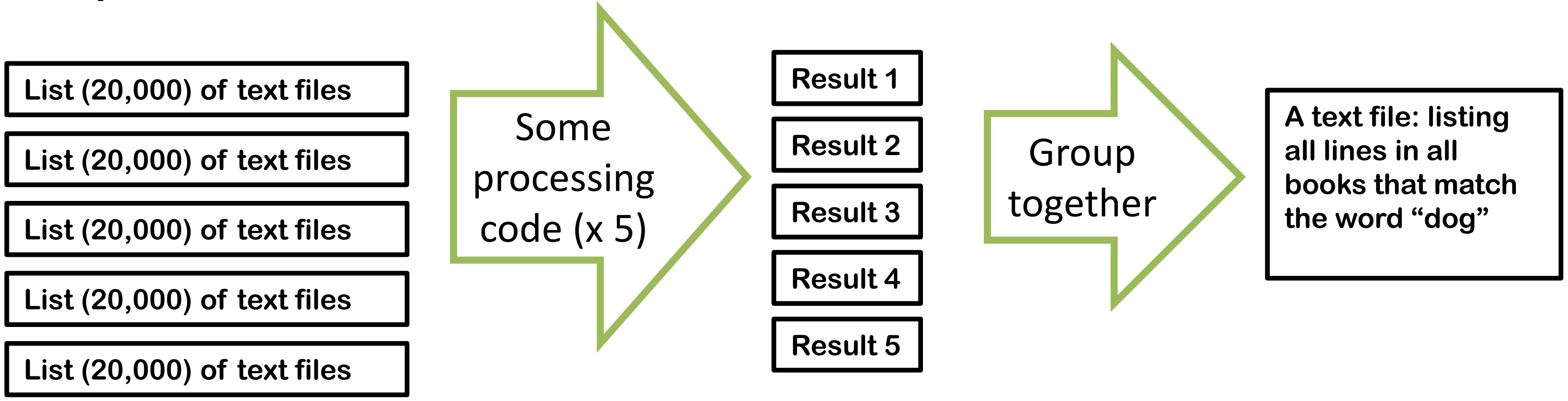
A text file: listing all lines in all books that match the word "dog"

grep_for_dog.sh

```
#!/bin/bash
input_file=$1
while read FILENAME; do
      grep dog $FILENAME >> ${input_file}_result.txt
done <  $input_file
```

Simple parallelism by hand (2)

- A simple re-factoring splits the job into five parts:


List (20,000) of text files
List (20,000) of text files
List (20,000) of text files
List (20,000) of text files
List (20,000) of text files
Some processing code (x 5)
Result 1
Result 2
Result 3
Result 4
Result 5
Group together
A text file: listing all lines in all books that match the word "dog"

```
$ split -l 20000 -d list_of_files.txt   # Writes to "x00", "x01", …, "x04"
$ for i in x??; do grep_for_dog.sh $i & done
$ cat *_result.txt > output.txt
```

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Simple parallelism with Jug (1)

- **Jug** is a Python library that:
  - allows you to write code that is **broken up into tasks** and
  - **run different tasks on different processors**.
- Jug also saves all the intermediate results to its backend in a way that allows them to be retrieved later.
  - The backend is typically the file system

Simple parallelism with Jug (2)

- Jug is useful for:
  - Running parallel jobs on a multi-core machine
  - Running parallel jobs on a cluster (such as LOTUS)

For more information, see:
http://pythonhosted.org/Jug/

Jug Example

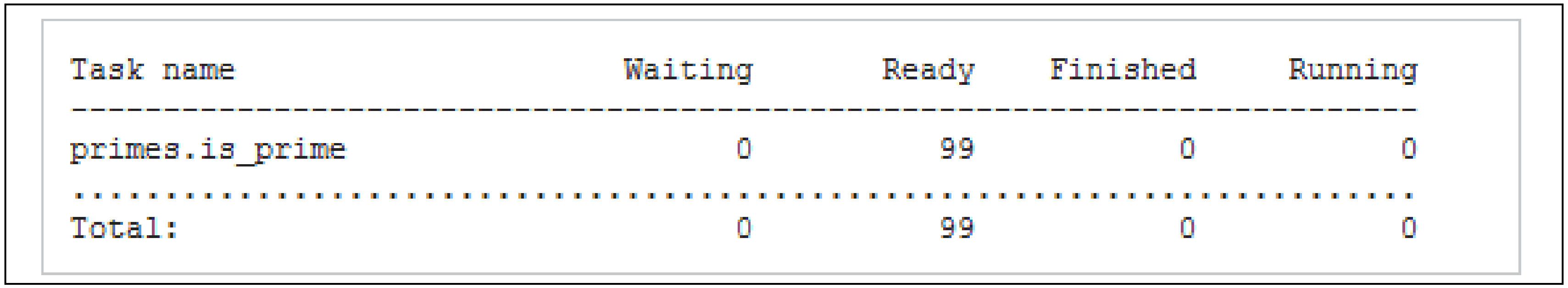
Jug allows you to turn a Python function into a parallel “Task”. It handles the parallelisation for you.

```
from jug import TaskGenerator
from time import sleep

@TaskGenerator
def is_prime(n):
    sleep(1)
    for j in xrange(2, n-1):
        if (n % j) == 0:
            return False
    return True

primes100 = map(is_prime, xrange(2, 101))
```

Running Jug

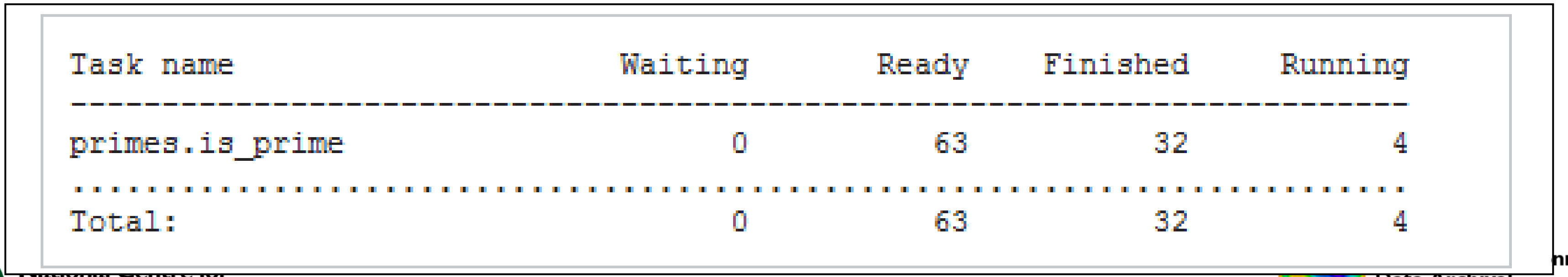
Jug has a command-line tool that gives you interactive information about the job:

```
$ jug status primes.py
```



```
Task name                   Waiting       Ready    Finished     Running
-----------------------------------------------------------------------
primes.is_prime                   0          99           0           0
.......................................................................
Total:                            0          99           0           0
```

```
$ jug execute primes.py &  # Execute this line 4 times
$ jug status primes.py
```



```
Task name                   Waiting       Ready    Finished     Running
-----------------------------------------------------------------------
primes.is_prime                   0          63          32           4
.......................................................................
Total:                            0          63          32           4
```

JASMIN & LOTUS

Main components of JASMIN

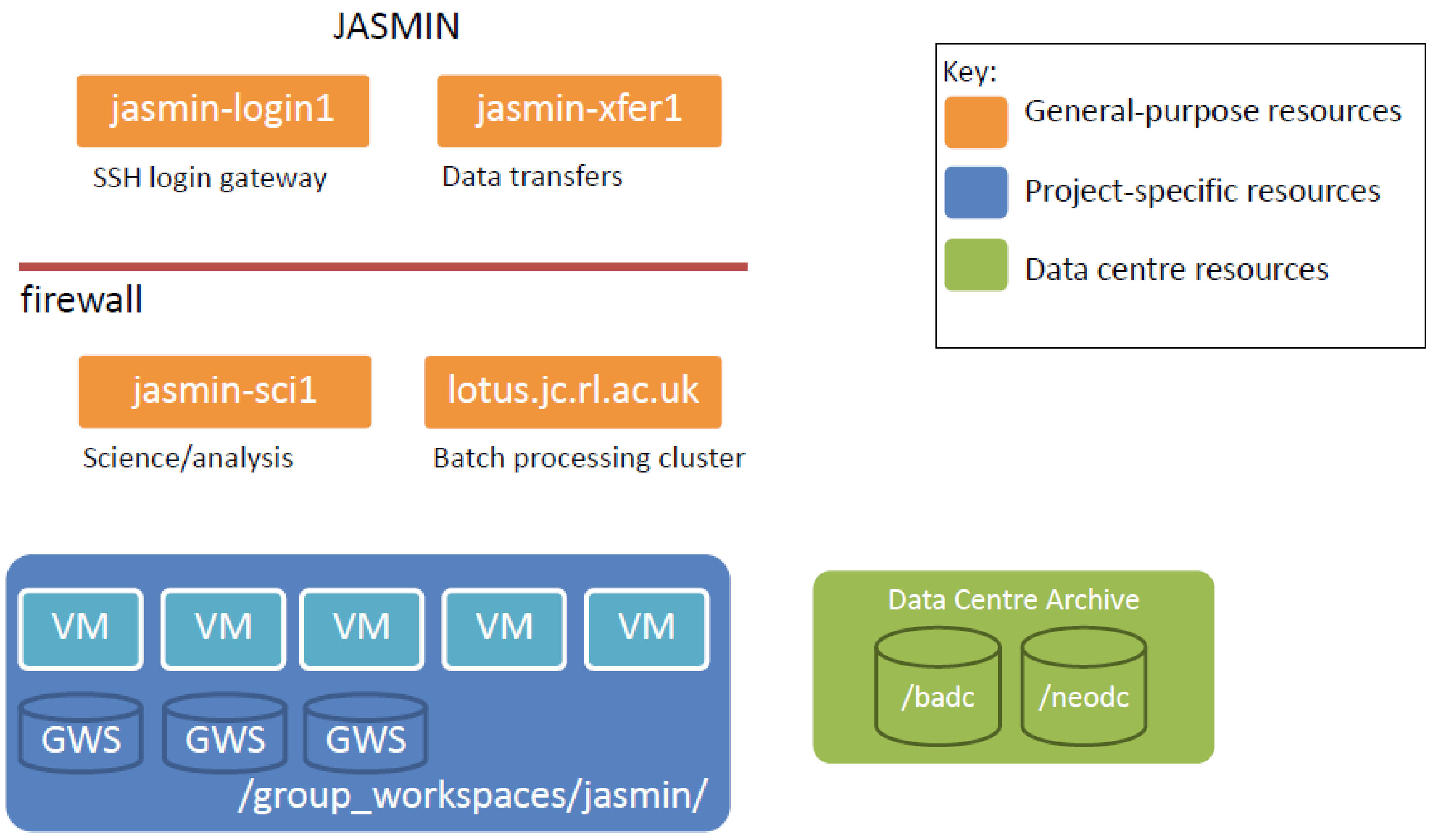
- ~10 Petabytes of high-performance parallel disk: for archives, collaboration and data analysis
- A large compute platform for:
  - Hosting virtual machines:
    1. For specific projects/organisations
    2. For generic scientific usage (transfer/analysis)
  - Compute Cluster (LOTUS):
    - For parallel and batch jobs
    - For running models

...and a lot of other stuff not mentioned here.

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

JASMIN in pictures



The JASMIN Scientific Analysis Servers (1)

- A number of processing servers exist for general data analysis tasks:
  - jasmin-sci[12].ceda.ac.uk
  - cems-sci[12].cems.rl.ac.uk
- These servers allow direct access to BADC and NEODC archives as well as any “group workspaces” - for authorised users.

The JASMIN Scientific Analysis Servers (2)

A common set of tools are installed (known as the “JASMIN Analysis Platform”) including:

NCO, CDO, CdatLite, Python2.7, NetCDF4, python-netcdf, Iris, Matplotlib, Octave, R, ...

See: http://proj.badc.rl.ac.uk/cedaservices/wiki/JASMIN/AnalysisPlatform/Packages

The LOTUS cluster on JASMIN

The JASMIN Scientific Analysis Servers are very useful – but they are limited in resource and may well be swamped by other users.

The **LOTUS cluster is a far bigger resource**.

We'll see some examples later...

LOTUS: Queue configuration

Batch queues:

1. **lotus:** (8 Nodes with 2x6 Cores Intel 3.5GHz, 48G RAM, 10G Networking, 10Gb Std latency TCP MPI) = 96 Cores
2. **lotus-g:** (3..6 Nodes with 2x6 Cores Intel 3.0GHz 96G RAM, 10G Networking, 10Gb Gnodal low latency TCP MPI) = 36..72 cores
3. **lotus-smp:** (1 node with 4x12 cores AMD 2.6GHZ 256GB RAM, 10Gb Networking)
4. **lotus-serial:** (co-exists with lotus-smp and lotus queue hardware)

LOTUS: System software

- RHEL6.2 Operating System
- Platform LSF batch scheduler
- Platform MPI (+/- OpenMPI)
  - Full Support for MPI I/O on Panasas parallel file systems
- Intel and PGI compilers
- Central repository for installed software
- Environment modules

http://proj.badc.rl.ac.uk/cedaservices/wiki/JASMIN/LOTUS

LOTUS: Data Analysis Software

All LOTUS nodes have the **same software** packages available as the JASMIN Scientific Analysis Servers.

This means you can:

1. **Develop code** on the generic **Analysis Servers**
2. Run in **batch mode via LOTUS**

LOTUS: Job Control

Submitting a job (you must SSH to **lotus.jc.rl.ac.uk**):

```
$ bsub [options] <command>
```

View the status of jobs:

```
$ bjobs
JOBID USER STAT QUEUE FROM_HOST EXEC_HOST JOB_NAME SUBMIT_TIME
71880 fred PEND lotus lotus.jc.rl */hostname Mar 18 16:26
```

Cancel a job with:

```
$ bkill <job_id>
```

See details at:

http://www.ceda.ac.uk/help/users-guide/lotus/

LOTUS: Where to read/write?

LOTUS can see:

- Home directories (read/write) – but 10Gb QUOTA!
- Group Workspaces (read/write)
- BADC & NEODC Archives (read-only)
- `/work/scratch/<userid>` (read/write) – but SMALL!
- `/tmp` (read/write) – but LOCAL TO NODE!

Group workspaces allow you to write large volumes of data (and keep it!)

Parallel Processing Examples

Example 1: extracting CF standard names from 200,000 files (1)

Requirement:

- Extract CF-netCDF standard name attributes from 200,000 netCDF files.

Details:

- Starting with a file containing a list of 200,000 netCDF files
- For each netCDF file:
  - Run the "**ncdump –h**" command to extract the header as text
  - Pipe that into the "**grep**" command to match only "**standard_name**"

grep_names.sh

```
#!/bin/bash
cd /group_workspace/jasmin/megalon/test
while read FILENAME; do
    ncdump –h $FILENAME | grep standard_name | cut –d: -f2 | cut -d; -f1 >> $2
done < $1
```

Example 1: extracting CF standard names from 200,000 files (2)

Running in parallel via LOTUS:

- Split the file containing a list of 200,000 netCDF files into 10:

  ```
  $ split -l 20000 -d input_file.txt
  ```

  Which writes files called: "x00", "x01", ..., "x09"

- Write a loop and submit 10 jobs to LOTUS:

  ```
  for input in x* ; do
      bsub ./grep_names.sh $input  ${input}_result.txt
  done
  ```

Example 1: extracting CF standard names from 200,000 files (3)

View progress of the jobs running on LOTUS:

```
$ bjobs
...will print out details of jobs running...
```

When all jobs have run: merge results into one file:

```
$ cat x??_results.txt > output.txt
```

"output.txt" contains all the results in one place.

NOTE: This example requires that all results are collected before merging them into "output.txt"

Example 2: extract spatial subsets from CMIP5 experiments (1)

Requirement:

Extract spatial subsets from CMIP5 a set of experiments.

Details:

- For each model:
  - For each variable (hus, ps, ta, ua & va):
    - Extract a spatial subset (80° to 140° Longitude; -30° to 40° Latitude)
    - Where:
      - Frequency: 6hr
      - Realm: atmosphere
      - Ensemble: r1i1p1

Example 2: extract spatial subsets from CMIP5 experiments (2)

Basic Implementation:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Glob all relevant input NetCDF files
  - Call Python script; extract spatial subset; write output

Script 2 (Python):

- Read input file; extract spatial subset for variable; write output file.
- Main code used: cf-python library

Extract from: **extract_cmip5_subset.py**

```
import cf
f = cf.read(infile)
subset = f[2].subspace(latitude=cf.wi(bb.south, bb.north), longitude=cf.wi(bb.west, bb.east))
cf.write(subset, outfile)
```

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Example 2: extract spatial subsets from CMIP5 experiments (3)

Parallel Implementation using LOTUS:

Script 1 (bash):

- For each variable (hus, ps, ta, ua & va):
  - Make output directory
  - Find all relevant input NetCDF files
  - Submit a job to the LOTUS scheduler that will call the Python script
  - Use the "**bsub**" command:

```
bsub -q lotus -o $outdir/`date +%s`.txt ~/extract_cmip5_subset.py $nc_file $this_dir $var
```

Why use this approach?

- Because you can submit 200 jobs in one go.
- Lotus executes jobs when resource becomes available
- They will all run and complete in parallel

Example 3: chaining tasks with Jug: calculating monthly means (1)

Requirement:

- Calculate a monthly mean from 6-hourly model data

Details:

- For each date in the month:
  - Extract data for 00Z, 06Z, 12Z & 18Z
  - Calculate daily average from them
- Gather all daily averages
- Calculate monthly average
- Write monthly average to netCDF file

Example 3: chaining tasks with Jug: calculating monthly means (2)

Basic Implementation in Python (PART 1):

```
import cdms2

def calcDailyMean(date, var = "U10"):
    f = cdms2.open("/badc/ecmwf-era-interim/metadata/cdml/era-interim_ggas.xml")
    v = f(var, time = ("%s 00:00" % date, "%s 23:59" % date))
    total = sum([v[i] for i in range(4))
    f.close()
    return total

def calcMonthlyMean(data, date):
    total = sum([data[i] for i in range(len(data))])
    return total

def writeMonthlyMean(data, fname):
    f = cdms2.open(fname, "w")
    f.write(data)
    f.close()
```

Example 3: chaining tasks with Jug: calculating monthly means (3)

Basic Implementation in Python (PART 2):

```
(...continued...)

year = 2001
month = 1
all_days = []

for day in range(1, 32):
    d = "%.4d-%.2d-%.2d" % (year, month, day)
    all_days.append(calcDailyMean(d))

monthly_mean = calcMonthlyMean(all_days, "%.4d-%.2d-15" % (year, month))

output_path = "/group_workspace/jasmin/megalon/output.nc”
writeMonthlyMean(monthly_mean, output_path)
```

Run sequentially: `$ python2.7  monthly_mean.py`

Example 3: chaining tasks with Jug: calculating monthly means (4)

Convert to Jug Tasks by adding:

```
from jug import TaskGenerator, barrier
import cdms2

@TaskGenerator
def calcDailyMean(date, var = "U10"):
    f = cdms2.open("/badc/ecmwf-era-interim/metadata/cdml/era-interim_ggas.xml")
    v = f(var, time = ("%s 00:00" % date, "%s 23:59" % date))
    total = sum([v[i] for i in range(4))
    f.close()
    return total

@TaskGenerator
def calcMonthlyMean(data, date):
    total = sum([data[i] for i in range(len(data))])
    return total

@TaskGenerator
def writeMonthlyMean(data, fname):
```

Example 3: chaining tasks with Jug: calculating monthly means (5)

Add Jug "barriers" to wait for dependent tasks to complete:

```
(...continued...)

year = 2001
month = 1
all_days = []

for day in range(1, 32):
    d = "%.4d-%.2d-%.2d" % (year, month, day)
    all_days.append(calcDailyMean(d))

barrier()
monthly_mean = calcMonthlyMean(all_days, "%.4d-%.2d-15" % (year, month))

barrier()
output_path = "/group_workspace/jasmin/megalon/output.nc"
writeMonthlyMean(monthly_mean, output_path)
```

Example 3: chaining tasks with Jug: calculating monthly means (6)

Now we can run with Jug, locally (on 4 processors):

```
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &
$ jug execute monthly_mean.py &


$ jug status monthly_mean.py    # will report on status
```

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Example 3: chaining tasks with Jug: calculating monthly means (7)

Or run on LOTUS:

On LOTUS we can use the “Job Array” submission feature to submit the same Jug script multiple times to the cluster, e.g. Run 40 times (on 40 processors):

```
$ dir=/group_workspace/jasmin/megalon
$ bsub -e $dr/%J-%I.err -o $dr/%J-%I.out -J "jug[1-
40]" jug execute monthly_mean.py
```

Each process will output to:

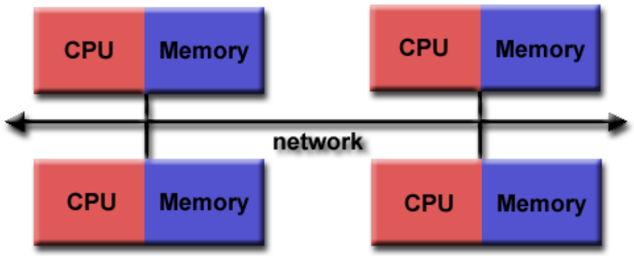
- ${dir}/${job_id}-${array_count}.out

So why did we use Jug?

- Jug makes it trivial to **convert existing functions into parallel “Tasks”**.
- Jug allows simple management of *race conditions* where dependencies exist between tasks.
- Jug gives us a simple way of farming out our workflows onto processing clusters (such as LOTUS).

But clearly you could write it yourself...if you want.

What about using MPI?



Some background: What is MPI?

- **MPI** stands for ***Message Passing Interface***.
- Explaining MPI is beyond our scope.
- MPI provides a standard specification that enables message passing between different nodes of a parallel computer system.
- It follows a distributed memory model.

So it maps nicely to use on the LOTUS cluster...and lots of existing parallel code uses MPI (as implemented in C, Fortran, Python etc).

Using MPI on LOTUS

The LOTUS User Guide provides detailed instructions on how to:

- Load MPI modules
- Compile C/Fortran code to use MPI
- Log on to interactive nodes for compilation
- Submitting jobs using MPI

There is not time to cover this in detail, please see:

http://www.ceda.ac.uk/help/users-guide/lotus/

Re-factoring is important too!

Efficiency gains through re-factoring (1)

Major gains can be made by changing the order and structure of your code.

Problems with your code might include:

1. Code will not run because of memory requirements
2. Code runs sequentially and takes a long time
3. Code does run but falls over because of resource limits.

In some cases you can create loops that can be scripted as separate processes (or JUG tasks) allowing you to submit them in parallel.

Efficiency gains through re-factoring (2)

Here is a real-world example:

The Problem: Trying to run the NCO tool "ncea" to calculate an average from a large dataset. It will not run!

Why? The "ncea" command reports this...and then exits:

– "unable to allocate 7932598800 bytes" (which is about 8 Gbytes)

Possible solutions:

1. Data files hold multiple variables: Operate on one at a time:

```
ncea -v vosaline means/199[45678]/*y01T.nc -o test.nc
```

2. Reduce the number of files (i.e. years) processed each time:

```
ncea means/199[45]/*y01T.nc -o test.nc
```

Other Python-based Parallel tools

The following page brings together details of many different parallel tools available for python users:

- https://wiki.python.org/moin/ParallelProcessing

Running “iPython Parallel” within JASMIN

iPython is a suite very powerful packages and tools to extend the capability of python. This includes a sophisticated architecture for parallel and distributed computing able to support a range of styles including:

- Single program, multiple data (SPMD) parallelism.
- Multiple program, multiple data (MPMD) parallelism.
- Message passing using MPI.
- Task farming.
- Data parallel.

http://ipython.org/ipython-doc/stable/parallel/

The future of parallel data analysis

- Analysing Big Data is a challenge! Software needs to adapt and scientists need to be able to adapt their code to keep up!

Number of files	3,222,967
Number of datasets	54,274
Archive Volume (TB)	1,483
Models with data published	64
Models with documentation published in archive	38
Experiments	108
Modelling centres	32
Data Nodes	22

CMIP5 Status (early 2013)

The future of parallel data analysis

We are likely to see more:

- Parallel I/O in software libraries;
- Web processing services that do the parallel analysis remotely;
- Analysis Platforms (like JASMIN) that allow scientists to run code next to the data;
- Learning to write parallel code now is likely to be of great benefit in future;

Further information

JASMIN Analysis Platform (software packages):

http://proj.badc.rl.ac.uk/cedaservices/wiki/JASMIN/AnalysisPlatform/Package

LOTUS Overview:

http://proj.badc.rl.ac.uk/cedaservices/wiki/JASMIN/LOTUS

LOTUS User Guide:

http://www.ceda.ac.uk/help/users-guide/lotus/

Jug:

http://pythonhosted.org/Jug/

Parallel processing:

https://computing.llnl.gov/tutorials/parallel_comp/

National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL

Centre for Environmental Data Archival
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL