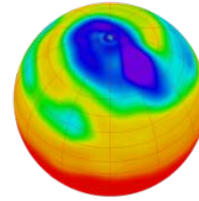




**National Centre for
Atmospheric Science**

NATURAL ENVIRONMENT RESEARCH COUNCIL



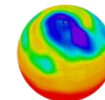
**Centre for Environmental
Data Archival**

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

Parallel processing large data



**National Centre for
Atmospheric Science**
NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental
Data Archival**

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL
NATURAL ENVIRONMENT RESEARCH COUNCIL

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
- The future of parallel data analysis

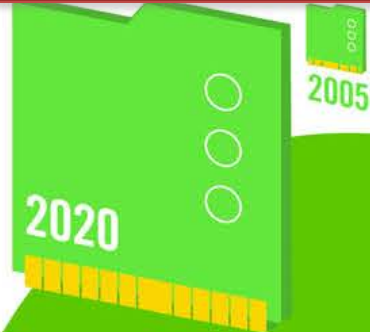
What is Big Data?

1 ZETTABYTE = 1,000,000,000 TERABYTES

40 ZETTABYTES

[43 TRILLION GIGABYTES]

of data will be created by 2020, an increase of 300 times from 2005

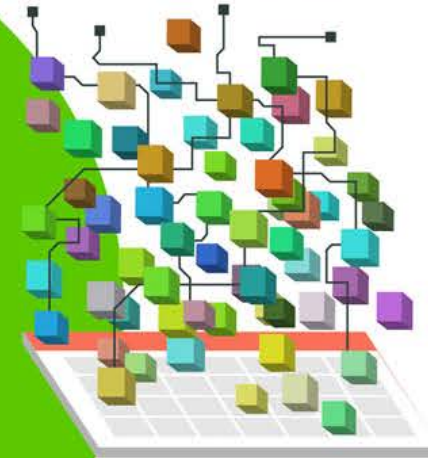


It's estimated that

2.5 QUINTILLION BYTES

[2.3 TRILLION GIGABYTES]

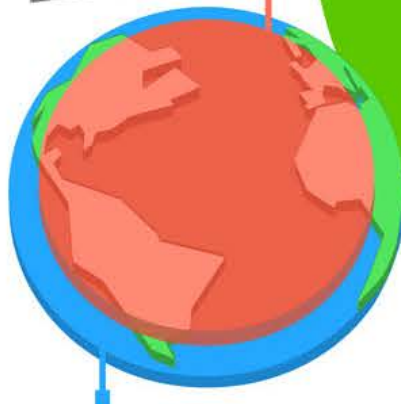
of data are created each day



Volume SCALE OF DATA

6 BILLION PEOPLE

have cell
phones



WORLD POPULATION: 7 BILLION



Most companies in the
U.S. have at least

100 TERABYTES

[100,000 GIGABYTES]

of data stored

T
F
O
D

From tr
history
stored,
and se
But wh
massive

As a le
break





Processing big data: the issues

- Parallel processing in the Environmental Sciences has historically focussed on running highly-parallelised 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
- The generation of 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



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.

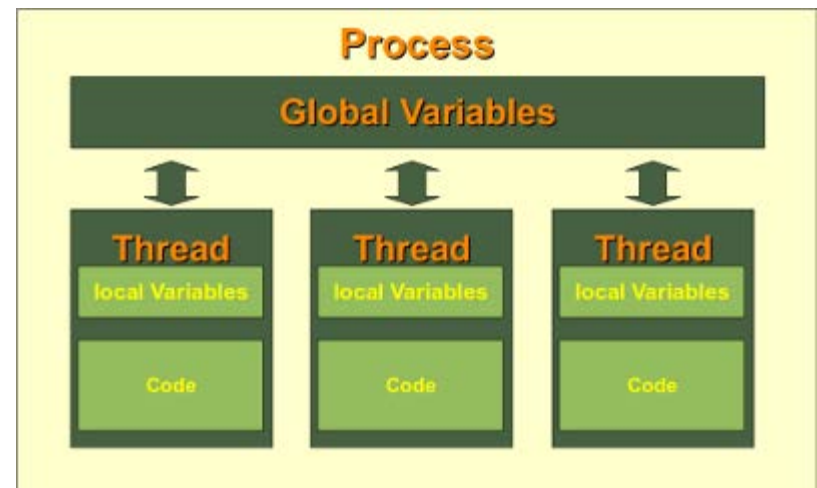
Race condition: A race condition occurs within concurrent environments. It is when a piece of code prevents code that is running elsewhere from accessing a shared resource, e.g., memory, and thus delays the other process.

How does my computer do so many things at once?

These days most computers, and even phones, have multiple processors.

However, 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 at the nanosecond scale where we humans can only keep track of milliseconds.



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

ental



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.



(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!

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

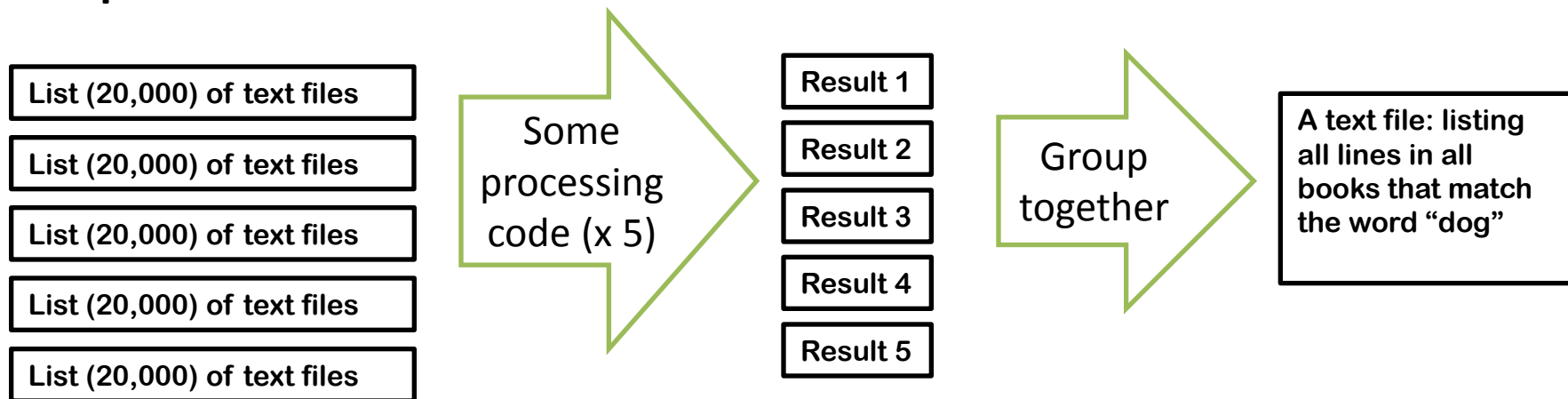
A text file: listing all lines in all books that match the word "dog"

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:

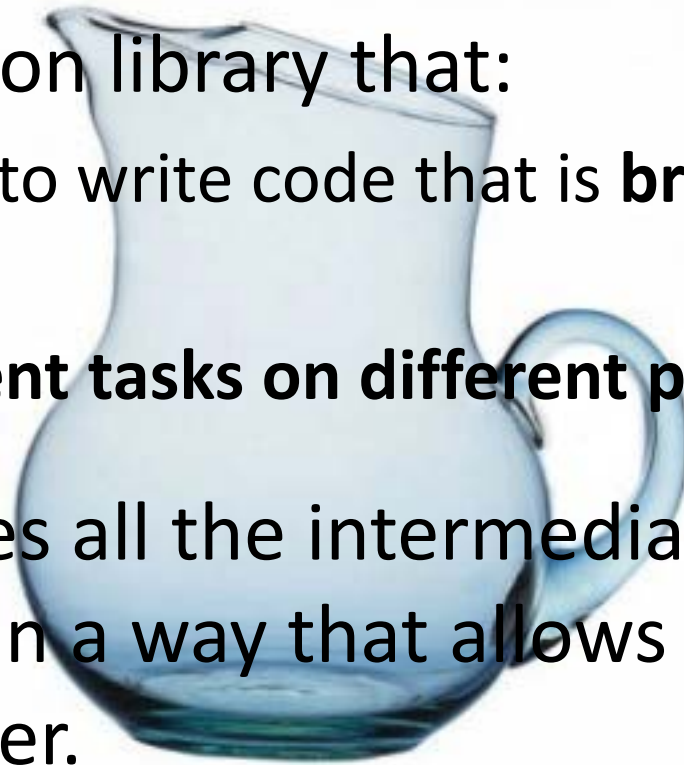


```
$ split -l 20000 -d list_of_files.txt # Writes to "x00", "x01", ..., "x04"
$ for i in x?; do grep_for_dog.sh $i & done
$ cat *_result.txt > output.txt
```



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

nmental





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.

JASMIN in pictures

JASMIN

jasmin-login1

SSH login gateway

jasmin-xfer1

Data transfers

Key:

General-purpose resources

Project-specific resources

Data centre resources

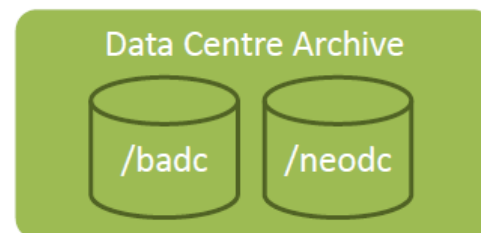
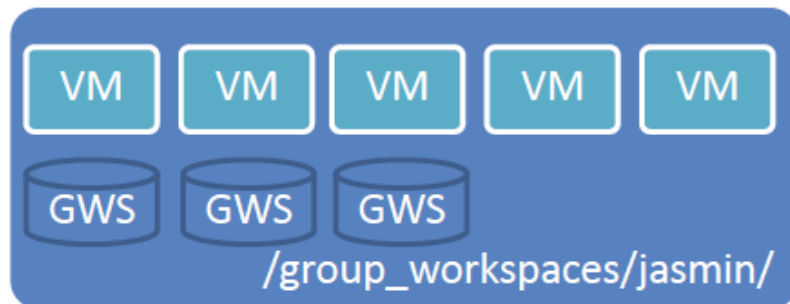
firewall

jasmin-sci1

Science/analysis

lotus.jc.rl.ac.uk

Batch processing cluster



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 CEDA 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.

Here, we provide real examples of how you can get up and running with LOTUS.



LOTUS: Specification (1)

Batch queues:

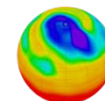
1. **lotus:** (8 Nodes with 2x6 Cores Intel 3.5GHz, 48G RAM, 10G Networking. 10Gb Std latency TCP MPI) = 96 Cores
2. **lotus-g:** (3..6 Nodes with 2x6 Cores Intel 3.0GHz 96G RAM, 10G Networking. 10Gb Gnodal low latency TCP MPI) = 36..72 cores
3. **lotus-smp:** (1 node with 4x12 cores AMD 2.6GHZ 256GB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



LOTUS: Specification (2)

- 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 community installed software
 - Environment modules

<http://proj.badc.rl.ac.uk/cedaservices/wiki/JASMIN/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!)

LOTUS: 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**
and then
2. **run in batch mode via LOTUS**

Parallel Processing Examples



Example 1: 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



Example 1: chaining tasks with Jug: calculating monthly means (2)

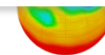
Basic Implementation in Python (PART 1):

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

Basic Implementation in Python (PART 2):

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

year = 2001
month = 1
all_days = []

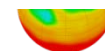
for day in range(1, 32):
    d = "%.4d-%.2d-%.2d" % (year, month, day)
    all_days.append(calcDailyMean(d))

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

mental





Example 1: 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
```

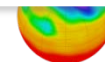
@TaskGenerator

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

@TaskGenerator

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

mental





Example 1: 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 1: chaining tasks with Jug: calculating monthly means (6)

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

```
$ jug_py27 execute monthly_mean.py &  
$ jug_py27 execute monthly_mean.py &  
$ jug_py27 execute monthly_mean.py &  
$ jug_py27 execute monthly_mean.py &  
  
$ jug_py27 status monthly_mean.py # will report on status
```




Example 1: chaining tasks with Jug: calculating monthly means (7)

Or run on LOTUS:

1. Create a simple shell script (“batch_jug.sh”) to execute multiple Jug processes (“n” is the number to run in parallel):

```
#!/bin/bash

n=$1
cd /group_workspace/jasmin/megalon

for i in `seq 1 $n`; do
    jug_py27 execute monthly_mean.py &
done
```



Example 1: chaining tasks with Jug: calculating monthly means (8)

Running on LOTUS (continued...):

2. Submit a job to the LOTUS queue:

- Reserve the number of parallel cores to run on (“n”)

```
$ n=8  
$ bsub -n $n ./batch_jug.sh $n
```

The LOTUS queue will allocate 8 cores to this job.

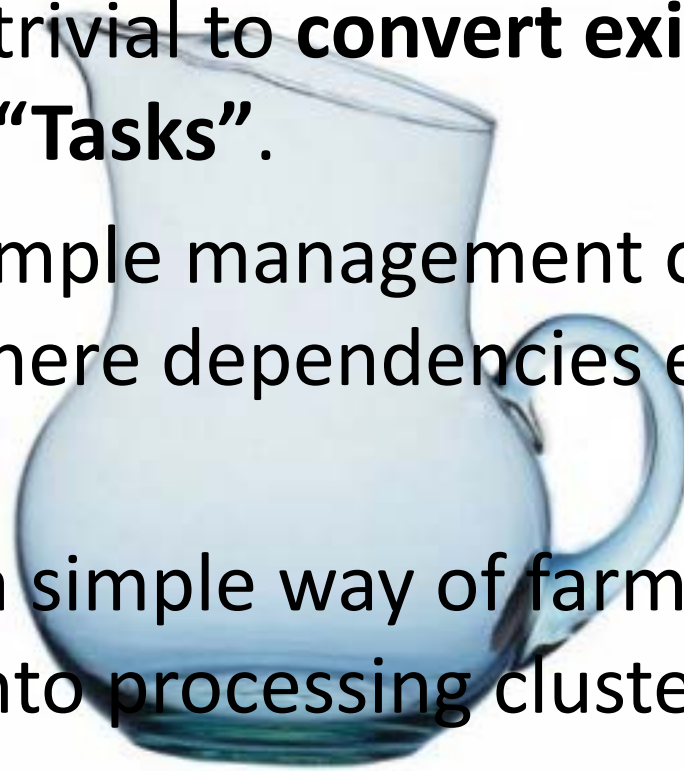
When 8 cores become available then Jug will carry out the tasks in order.

It will wait for tasks to complete when dependent tasks need to use their outputs.



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.



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:

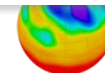
Script 1 (bash):

- 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

```
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)
```





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

Parallel Implementation using LOTUS:

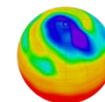
Script 1 (bash):

- 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



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. Issues might be:

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>

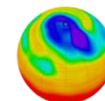


Running IPython Parallel within JASMIN

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 (1)

- 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



The future of parallel data analysis (2)

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/