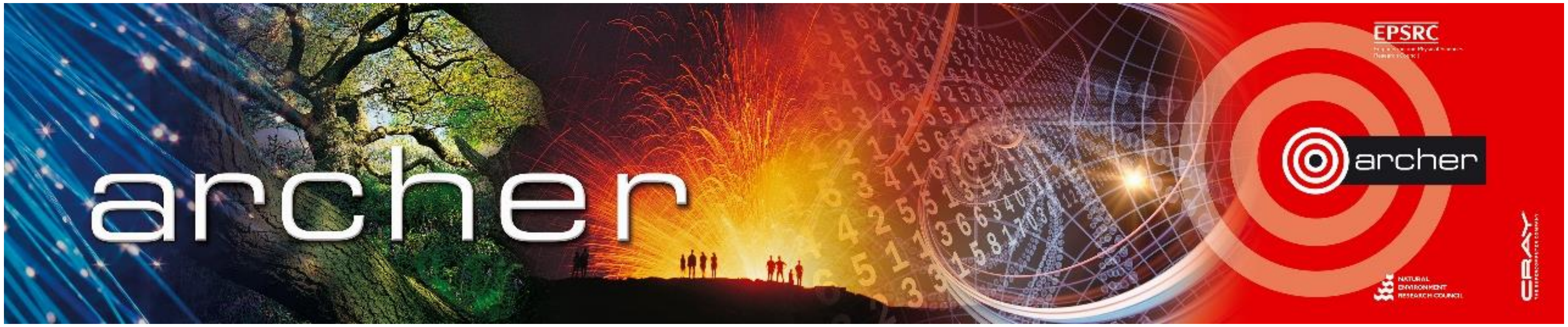


Welcome!

Virtual tutorial starts at 15:00 BST

Performance Analysis on ARCHER Using CrayPAT

ARCHER Virtual Tutorial, Wed 10th May 2017

Gordon Gibb; g.gibb@epcc.ed.ac.uk

Reusing this material

Outline

- Motivations
- Overview of CrayPAT
- Description of example CFD code
- Using perftools-lite
- Using perftools for sampling and tracing
- Apprentice2 GUI

Motivations – What is Profiling?

- Examine the behaviour of the code

- Pick out any subroutines/functions that cause slowdown or have unusual behaviour

- Two types:
  1. Sampling (periodically queries running code to determine what function the code is in)
  2. Tracing (adds instructions into the code that report when entering/leaving functions, and various statistics)

Sampling

- What is sampling?
  - Every so often (100 Hz default), look at the call stack of the Program
  - Record which function is being executed (+ callers etc.)

- A good starting point if you know nothing about the behaviour of a program

- Low overhead (~1%)
- Very easy to set up & run

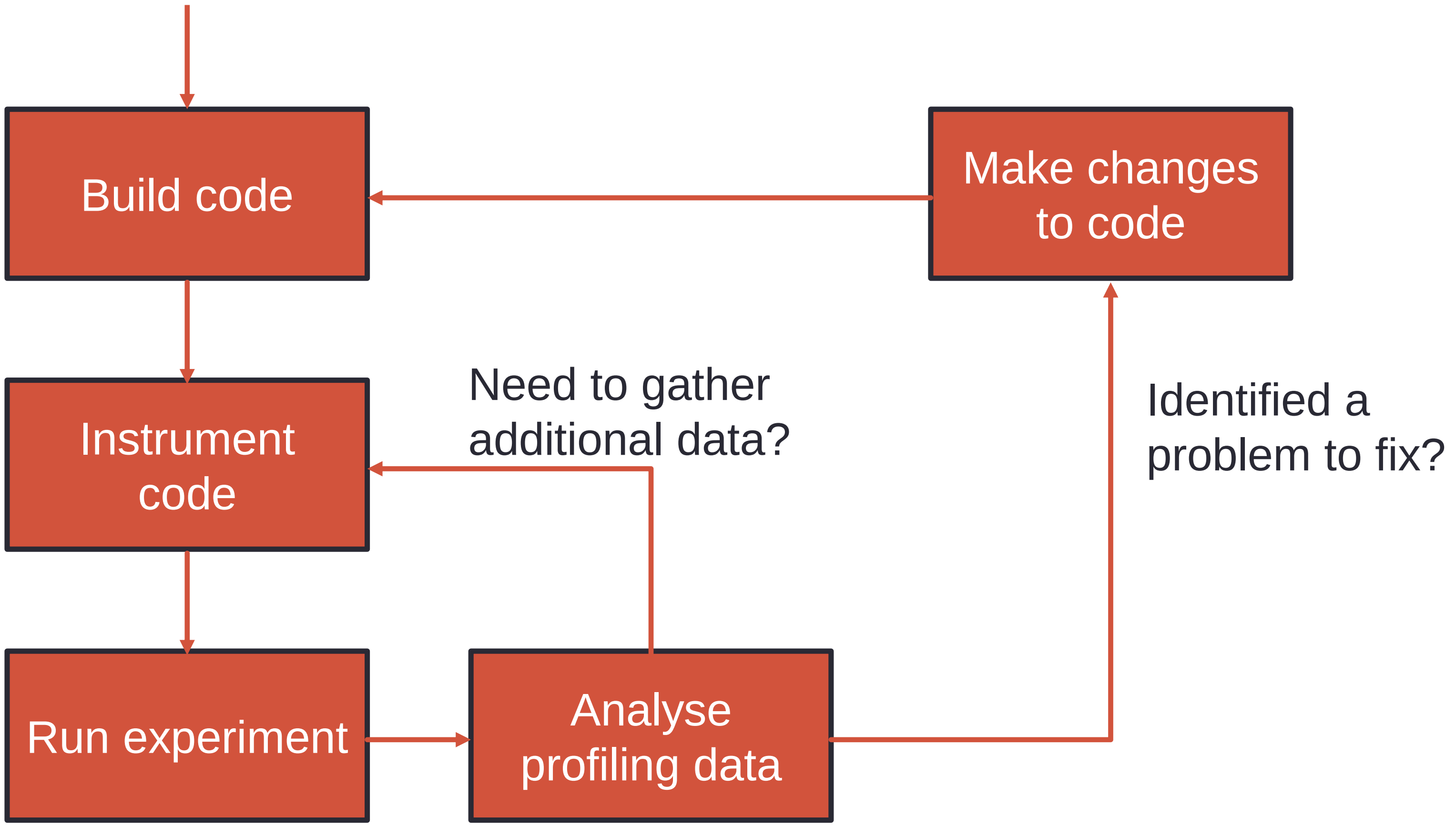
Tracing

- What is tracing?
    - ‘Trace intercept routines’ inserted at entry and exit of routines
        - Records amount of time spend in each call of a function
        - Exact sequence of events in a program execution
        - Allows for checking state of hardware counters
- Possible to generate endless detail about program execution
- Moderate overhead (~5-10%), depending on what you choose to trace
- Balance between detailed measurement and disturbing the experiment

Motivations - Why Profile?

- For developers:
  - Understand what the most time-consuming parts of the program are
  - Understand communication patterns and problems
    - E.g. load imbalance, synchronisation costs
  - Tool to help direct development efforts to give maximum benefits
- For users?
  - Understand why your program performs in a certain way
  - Help with choice of appropriate parameters, MPI processes…

Motivations – What is Profiling?



Picking an Example to Analyse

- Profiling generates a lot of extra data, and can cause your code to run more slowly
  - Need to choose a reasonably short example, but:
    - Program execution must be representative of a production run
    - Must be long enough to hide start-up and finalisation costs
    - Should include all the I/O of a normal job

- A good choice is something like a benchmark problem that takes a few minutes to run on a node/handful of nodes

Profilers

- In this course we will consider CrayPAT, however others exist:
  - Scalasca
  - Alinea MAP
  - Etc…

- With each tool you
  1. Instrument your code (typically during building)
  2. Run your code
  3. Analyse results

CrayPAT – pros and cons

+ Various levels of detail
+ Extreme customisibility for expert users

- Only available on Cray Platforms
- GUI is not particularly useful

Overview of CrayPAT

- Tools
  - pat_build
    - Instruments existing binaries for profiling
  - pat_report
    - Report generator
  - Apprentice2
    - GUI for analysing profiling data

CrayPAT Module layout

- Recently Cray updated their CrayPAT modules
- There are now three modules
  - perftools-base
  - perftools-lite
  - perftools
- The perftools-base module must be loaded before loading the other two modules

CrayPAT Module layout

- perftools-base
  - Contains man pages, and the Apprentice2 GUI tool.
- perftools-lite
  - Automatically gather profiling (sampling) data during program execution
  - Basic reporting dumped into standard output at end of run
  - Generate CrayPAT data files for further analysis (if needed)
- perftools
  - The full set of tools allowing both sampling and tracing

Example Test Code - CFD

- In this webinar I will use a simple MPI code to demonstrate parallel performance analysis
- A computational fluid dynamics (CFD) code is employed, which calculates the flow of fluid within a cavity with an inlet in one side, and an outlet on another.
- The code can calculate the inviscid or viscid fluid flow.

Example Test Code - CFD

- Solves Poisson's Equation for the streamfunction:

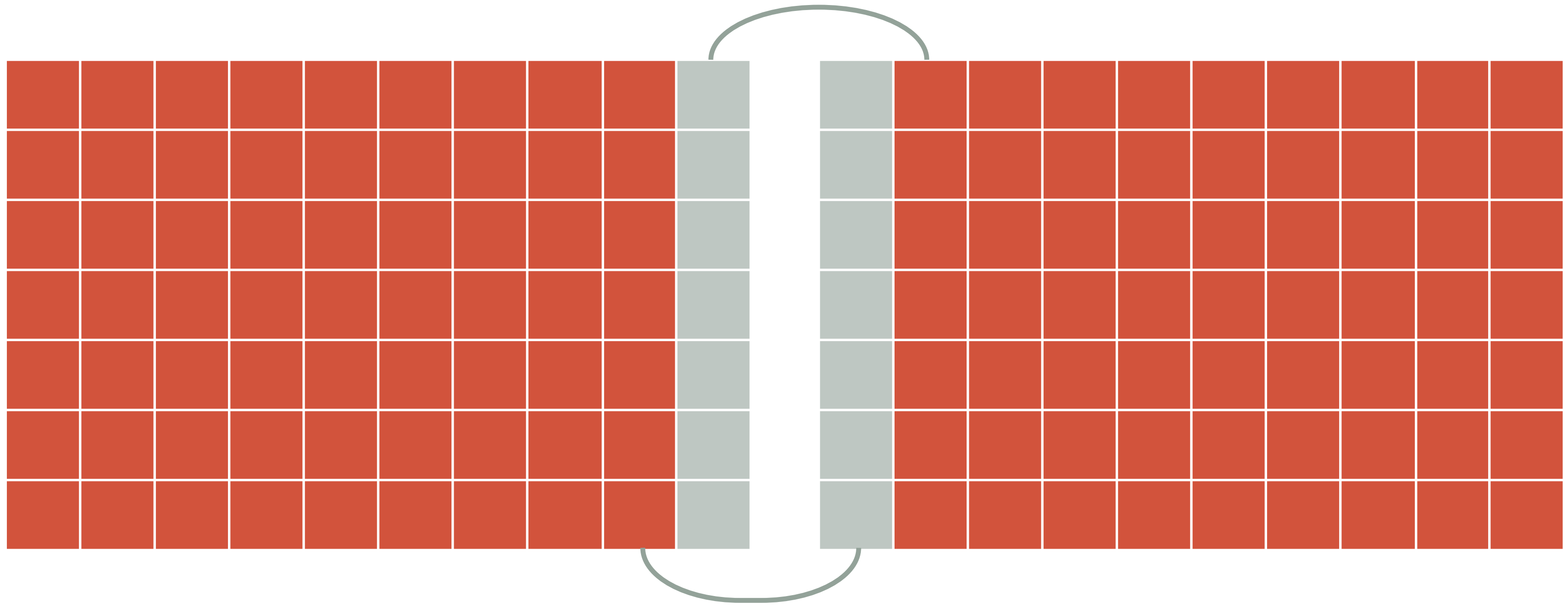
$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\zeta$$

$$u_x = \frac{\partial \psi}{\partial y}; \; u_y = -\frac{\partial \psi}{\partial x}; \; \zeta = \frac{\partial u_y}{\partial x} - \frac{\partial u_x}{\partial y}$$

- Available in both C and Fortran

Example Test Code - CFD

- Parallelised in the x (C) or y (Fortran) directions



- Halos transferred via MPI_Sendrecv

Example Test Code - CFD

- The code can be found on the course webpages

- To run it, use

```
aprun –n [nprocs] ./cfd <scale> <numiter> <Re>
```

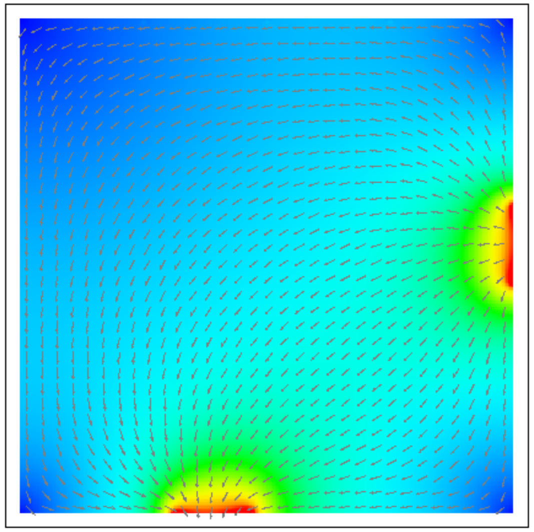
Where

- nprocs is the number of MPI processes
- scale scales the size of the box (32 x scale cells)
- numiter is the number of iterations
- Re (optional) is the Reynolds number ($0 \geq Re < 3.7$)

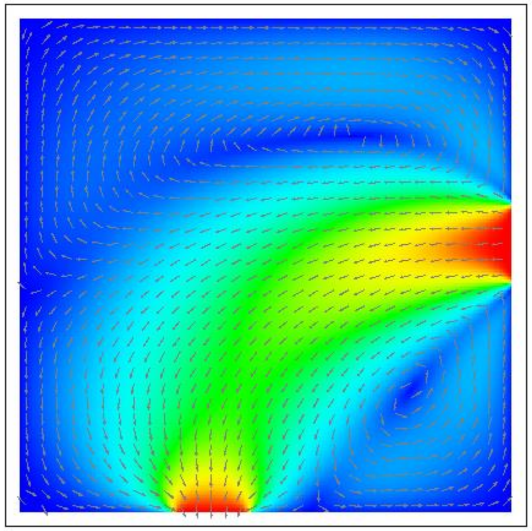
Example Test Code - CFD

- The output can be visualised using:

```
$ gnuplot –persist cfd.plt
```



$R_e = 0$



$R_e = 3.0$

Using perftools-lite

- Load the appropriate modules:

```
$ module load perftools-base
$ module load perftools-lite
```

- Build executable as normal

```
$ make clean; make
```

- Look for message at end:

```
INFO: creating the CrayPat-instrumented executable 'cfd'
(sample_profile) ...OK
```

Using perftools-lite

- Run your program
  - Usual PBS job submission script

- Basic profiling data appears at the end of job output
  - Overall job info
  - Top 10 most time-consuming functions
  - I/O, memory information
  - Report also saved in *.rpt file
  - A CrayPAT performance data file *.ap2 also created for further analysis

Demonstration

Using CrayPAT – More in-depth sampling

- Load the CrayPAT modules:

```
$ module load perftools-base
$ module load perftools
```

- Build executable as normal

```
$ make clean; make
```

- Instrument the binary using pat_build

```
$ pat_build ./cfd
```

Using CrayPAT - Sampling

- N.B. Linking must be separate from the compile stage
- Instrumentation creates a new binary cfd+pat
- Modify the job submission script to run this new binary, then submit the job

```
$ qsub submit.pbs
```

Using CrayPAT - Sampling

- Once the job has completed, it will have created an additional file: cfd+pat+<number>.xf

- Generate a human-readable report using pat_report

```
$ pat_report cfd+pat+<number>.xf
```

(You can put this information into a file by using the argument '–o <file>')

Demonstration

Using CrayPAT - Sampling

Pat_report also produces two other files; an .ap2 file, and an .apa file:

- The ap2 file acts as an input to the Apprentice2 graphical interface for viewing performance statistics (see later on)

```
$ app2 <file>.ap2
```

- The apa file contains suggested configuration options for a traced experiment

Using CrayPAT - Tracing

- Pat_build options:
  - For full list see man pat_build
  - Tracegroups(-g)
    - e.g. mpi, lapack, omp
  - Tracing user functions
    - -w enables tracing user functions
    - -T trace specific functions
    - -u trace all visible user functions (use with extreme caution!)
- Complex to set up
  - Except if you only want to trace e.g. MPI library calls
  - This is where APA helps

Using CrayPAT - Tracing

- Automated Profiling Analysis (APA)
  - From the sampling experiment report generation, a *.apa file was generated containing recommended options for pat_build to set up a tracing experiment

- Usage: pat_build –O [filename].apa

- Defaults:
  - Trace MPI calls
  - Gather default hardware counter group
  - Trace user functions with > 1% of samples, up to limit of 200
  - Very small functions (< 200 bytes) not traced to limit overhead

Using CrayPAT - Tracing

- Instrument the binary for tracing using the .apa file as an input to pat_build

```
$ pat_build -O cfd+pat+<number>.apa
```

- Modify the job submission script to use the new binary then submit the job

```
$ qsub submit.pbs
```

- View the results data using pat_report as before

```
$ pat_report cfd+apa+<number>.xf
```

- Then use Apprentice2 if desired

```
$ app2 cfd+apa+<number>.ap2
```

Demonstration

Using CrayPAT - pat_report

- pat_report options
  - Report generation is (almost) endlessly customisable
  - There are several pre-defined reports that are a good place to start:
    - -O profile (default) – list of most expensive functions
    - -O calltree / callers – top-up / bottom up function calls
    - -O ca+src – as above, with line numbers
    - -O load balance – displays min/mean/max across Pes
  - Each table in the report lists which options are needed to generate it:
    - e.g. Tableoption:

```
-O profile
Options implied by table option:
-d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE
```

Using CrayPAT - pat_report

- Implied options are a good starting point for customisation
  - See man pat_report for full list of options

- Each table also suggests options for related tables, and additional pat_report flags

- Also, check the ‘Observations and suggestions’ section

Using CrayPAT

- You can use the report to determine if you would like to change the tracing you want to carry out, and repeat tracing with different options as necessary until you have gathered the necessary information

- More information on CrayPAT can be found using the commands

```
$ pat_help
$ man intro_pat
$ man pat_build
$ man pat_report
```

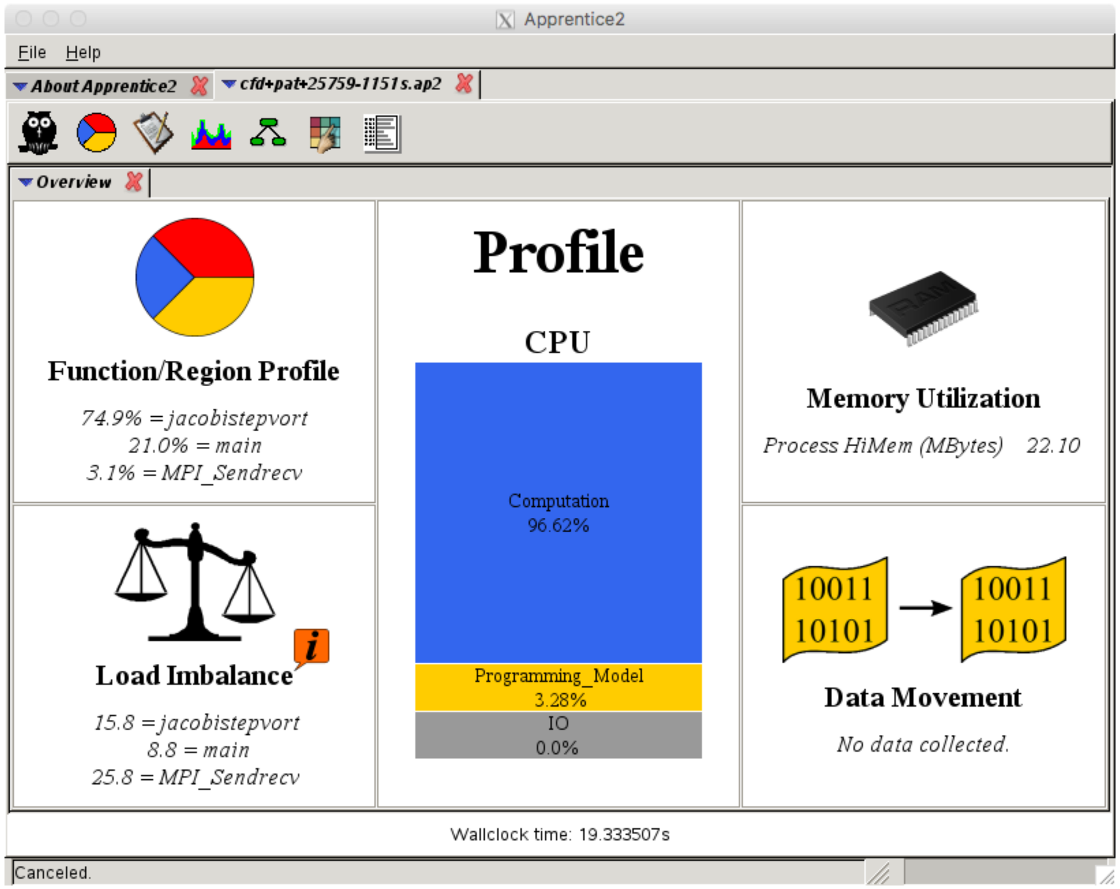
Using CrayPAT – Apprentice2

- CrayPAT includes a GUI called Apprentice2
  - Reads the portable *.ap2 file format
  - Graphical view of the calltree
  - Chart views of selected data
  - Hardware counters, activity graphs
  - Application trace available by setting PAT_RT_SUMMARY=0 before running your application
  - Warning – v. large trace files (MBs -> GBs!)
- • Can be run directly from ARCHER via X-windows

```
app2 &
```

- • Or binaries available for Mac & Windows: /opt/cray/perftools/6.2.2/share/desktop_installers/

Using CrayPAT – Apprentice2



Profile

CPU

Computation 96.62%

Programming_Model 3.28%

IO 0.0%

Function/Region Profile

74.9% = jacobistepvort
21.0% = main
3.1% = MPI_Sendrecv

Memory Utilization

Process HiMem (MBytes) 22.10

Load Imbalance

15.8 = jacobistepvort
8.8 = main
25.8 = MPI_Sendrecv

Data Movement

No data collected.

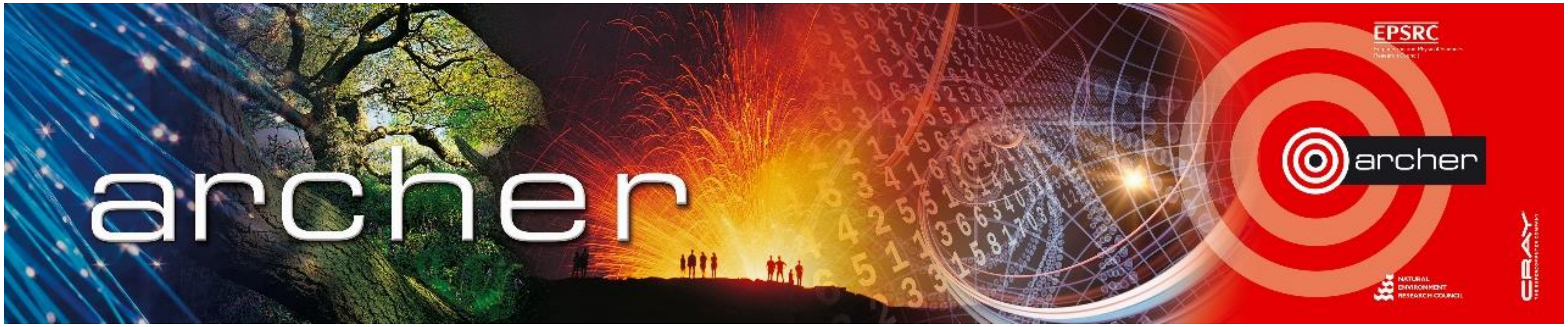
Wallclock time: 19.333507s

Canceled.

Demonstration

The End

- Questions?



Goodbye!

Virtual tutorial has finished