Advanced Operating Systems:
Lab 1 - I/O

Lecturelet 1

Prof. Robert N. M. Watson

2021-2022

Lab objectives

In the labs, you will:

- Utilise systems research methodology and practice

- Explore real-world systems artefacts through performance and functional evaluation/analysis

- Develop scientific writing skills **(L41 only)**

Documents on the module/unit websites

- We have provided several documents you will need from the module website, which apply to all labs:
  - Lab Setup Guide   (Everyone)
  - DTrace Quick Start   (Everyone)
  - Lab Reports   (L41 only)

- These documents are specific to Lab 1:
  - Lab 1 – I/O – General Information   (Everyone)
  - Lab 1 – I/O – Part II Assignment   (Part II only)
  - Lab 1 – I/O – L41 Assignment   (L41 only)

- Please ensure that you are completing the correct assignment for your course – they are different!

Lab 1 – I/O

- This lab represents only a small number of assessed marks – it is intended as a learning exercise before we hit more complex topics:
    - Introduce our RPi4/FreeBSD environment
    - Explore user-kernel interactions via syscalls and traps
    - Engage with POSIX I/O and its implications
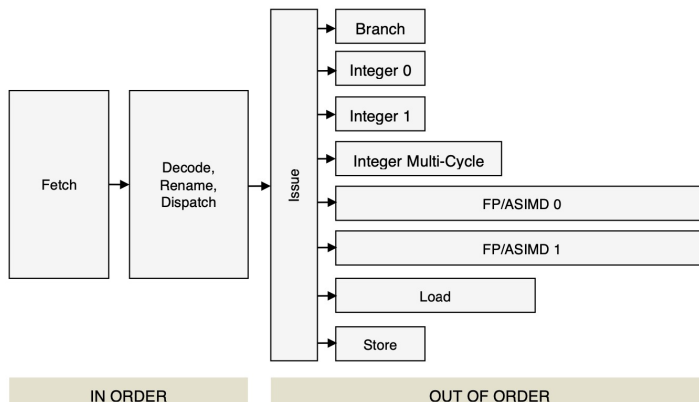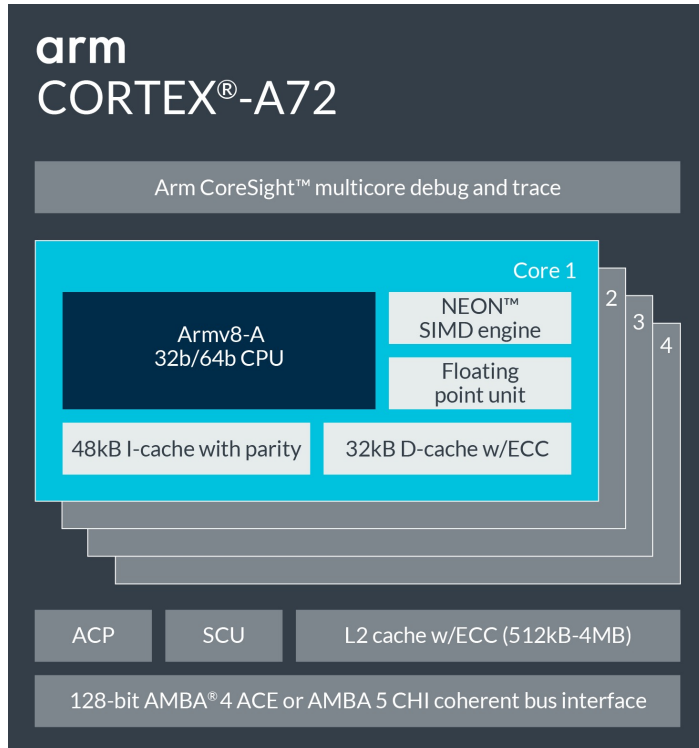   - Measure the probe effect

Our lab platform: RPi4s + FreeBSD 13.x
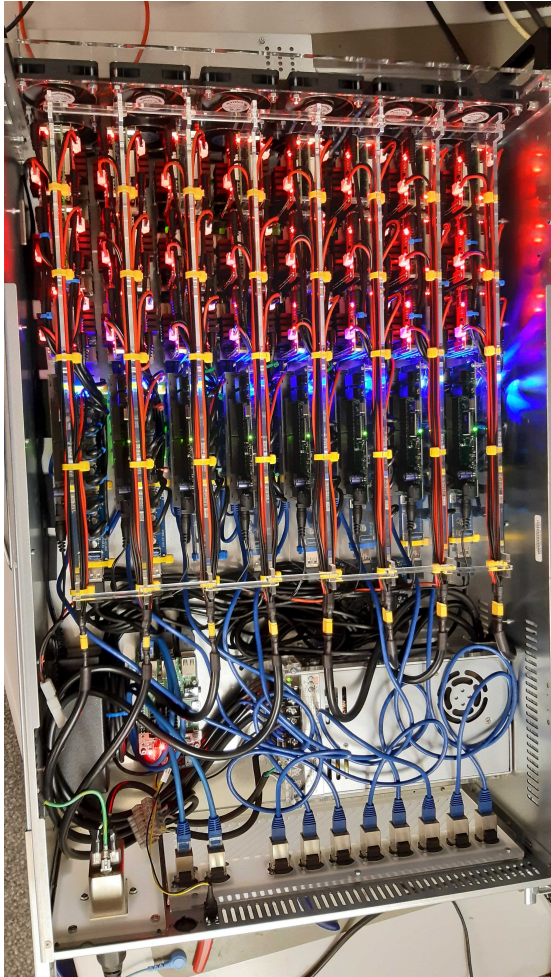
- 50x Raspberry Pi 4 boards
  - Broadcom BCM2711 SoC
  - 4x 64-bit A72 ARMv8-A cores
  - 8GB DRAM, 64G SD Card

- FreeBSD 13-STABLE
  - DTrace tracing tool
  - HWPMC counter framework
  - Bespoke benchmarks motivating OS and microarchitectural analysis
  - JupyterLab Notebook environment

- Access remotely via SSH + port forwarding for JupyterLab interface

High-density Cortex A-72 slide
(Some of this information will be useful only for later labs)

arm
CORTEX®-A72

Arm CoreSight™ multicore debug and trace

Core 1

Armv8-A
32b/64b CPU

NEON™
SIMD engine

Floating
point unit

48kB I-cache with parity

32kB D-cache w/ECC

ACP

SCU

L2 cache w/ECC (512kB-4MB)

128-bit AMBA® 4 ACE or AMBA 5 CHI coherent bus interface

The L1 memory system consists of separate instruction and data caches.

The L1 instruction memory system has the following features:

- 48KB 3-way set-associative instruction cache.
- Fixed line length of 64 bytes.
- Parity protection per 16 bits.
- Instruction cache that behaves as Physically-indexed and physically-tagged (PIPT).
- Least Recently Used (LRU) cache replacement policy.
- MBIST support.

The L1 data memory system has the following features:

- 32KB 2-way set-associative data cache.
- Fixed line length of 64 bytes.
- ECC protection per 32 bits.
- Data cache that is PIPT.
- Out-of-order, speculative, non-blocking load requests to Normal memory and non-speculative, non-blocking load requests to Device memory.
- LRU cache replacement policy.
- Hardware prefetcher that generates prefetches targeting both the L1 data cache and the L2 cache.
- MBIST support.

The features of the L2 memory system include:

- Configurable L2 cache size of 512KB, 1MB, 2MB and 4MB.
- Fixed line length of 64 bytes.
- Physically indexed and tagged cache.
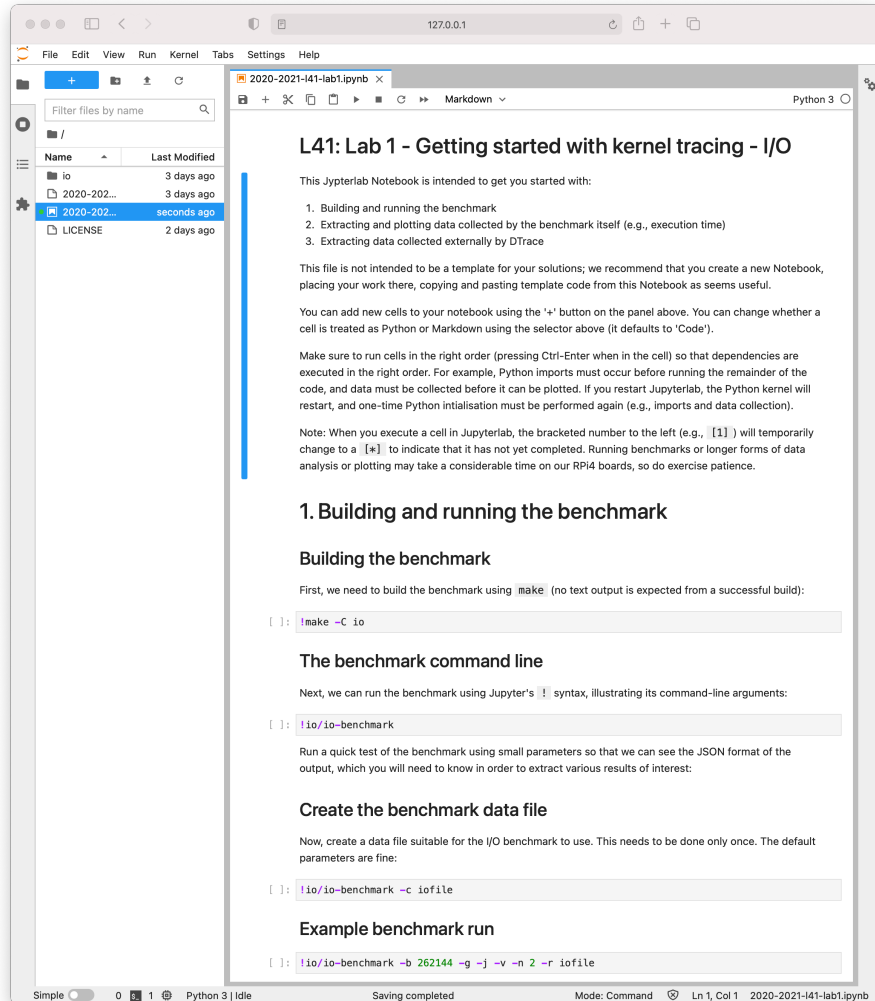- 16-way set-associative cache structure.

The MMU has the following features:

- 48-entry fully-associative L1 instruction TLB.
- 32-entry fully-associative L1 data TLB for data load and store pipeline
- 4-way set-associative 1024-entry L2 TLB in each processor.
- Intermediate table walk caches.
- The TLB entries contain a global indicator or an Address Space Identifier (ASID) to permit context switches without TLB flushes.
- The TLB entries contain a Virtual Machine Identifier (VMID) to permit virtual machine switches without TLB flushes.

Per-Core:
L1 I-Cache: 48K

Per-Core:
L1 D-Cache: 32K

Shared:
L2 Cache: 1M

Per-Core:
MMU
I-TLB: 48, D-TLB: 32,
L2-TLB: 1024

Branch
Integer 0
Integer 1
Integer Multi-Cycle
FP/ASIMD 0
FP/ASIMD 1
Load
Store

Fetch

Decode,
Rename,
Dispatch

Issue

IN ORDER

OUT OF ORDER

* Our benchmarks use only the first core to simplify analysis

JupyterLab

- Web-based interactive Python(++) environment
  - Runs on the RPi4, with UI reached via a web browser
  - "Notebooks" contain code, text, data, and plots
- Part II: Submit generated PDFs of their notebooks
- L41: Use notebook output in Lab Reports; don't submit
- Data analysis + plotting is best done within JupyterLab
 - … but you might find the DTrace command-line client easier to work with while doing exploration

JupyterLab – the UI

```
Buffer size:  4194304
Buffer size:  8388608
Buffer size:  16777216
'Benchmark run completed'
```

Plot the collected data

Finally, we generate a plot using `matplotlib`, consisting of medians and error bars based on IQR:

```python
[3]: fig1, ax = plt.subplots()
     ax.set_title("buffer size vs. bandwidth")

     x_coords = []
     y_coords = []
     low_errs = []
     high_errs = []

     for x in [2**v for v in range(25)]:
         x_coords.append(x)
         y_coords.append(medians[x])
         low_errs.append(q1s[x])
         high_errs.append(q3s[x])

     ax.set_xscale("log")
     ax.errorbar(x_coords, y_coords, [low_errs, high_errs])
     plt.show()
```
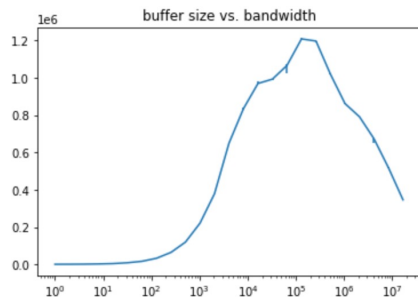

Create an annotated plot

In analysing this plot, it is worth considering key inflection points: Points on the plot where there are behavioural changes, and what they reflect. We can directly annotate those points on the plot using `avxline`.

In the next plot, we've manually placed several vertical lines at points where the data you collect is likely to experience inflection points. If they don't line up, check that you are collecting data as expected.

Be sure to take note of the linear Y axis and exponential X axis, and consider its implications for data analysis.

```python
[ ]: ### This content the same as the above cell
     fig1, ax = plt.subplots()
     ax.set_title("buffer size vs. bandwidth")
```

Executed cell (number)

Current cell

Unexecuted cell (no number)

Markdown cell

Code cell

Cell output

Ctrl-Enter in a cell executes it
In execution cells show [*]

8

Connecting to your board

- You will be contacted regarding your board assignment and how to collect login credentials
- The RPi4 nodes are accessible via SSH from within the CUDN (Cambridge University Data Network)
    - We expect this to apply to almost all students
- If you are not directly connected, you can:
    - Use the UIS VPN
    - Use the CL VPN (if you have a CL account)
    - Hop using SSH via another system on the CUDN (e.g., ely)
- You will run all parts of the lab as the root user
    - Exercise suitable care; we can re-image toasted boards, or assign you a spare, but data you may have on the board will be lost
- Please get in touch directly if you are having problems accessing your RPi4 board remotely

Web access over SSH

- In addition to logging in via SSH, you will also use SSH to port forward the JupyterLab web interface; e.g.,

ssh –L8888:127.0.0.1:8888 root@rpi4-000.advopsys.cl.cam.ac.uk

- This command allows software on your notebook/workstation to connect to 127.0.0.1:8888 and be transparently connected to the same port on the remote system
    - I.e., by connecting to http://127.0.0.1:8888
- JupyterLab will print out the URL to use it starts

Lab 1: Hypotheses

You will test and explore three hypotheses:

1. System-call overhead is substantial; structuring application requests to use larger buffer sizes will improve performance by amortizing that overhead.

2. Perform growth continues until we hit the system's peak I/O throughput, at which point performance will stabilise.

3. DTrace's probe effect associated is negligible.

These will be pursued differently in Part II and L41.

io-benchmark – read a file in blocks

- Two operational modes used in this lab:
    - Create (`create`)   Create a new benchmark data file
    - Read (`read`)   Perform `read()`s against data file
- Adjust operational parameters:
    - Block size (`-b`)   Block size used for each I/O
    - Total size (`-t`)   Total size across all blocks (must be an integer multiple of block size.)
    - Iterations (`-n`)   How many times to run
    - getrusage() (`-g`)   Report selected kernel statistics
- Output flags:
    - JSON (`-j`)   Generate machine-readable output
   - Verbose (`-v`)   Verbose output

Notes on the execution environment

- **/data** is a suitable directory tree to store your data in
- **/usr/src/sys** contains synchronized kernel source code
- You are running as root – please be careful not to hose the board you've been assigned
  - We can remotely re-image, but your data will be lost
- DTrace can have a significant impact on performance for some scripts – e.g., instrumenting ":::" (all probes)
  - Try not to render your board unresponsive, if possible
  - We can remotely reset, but it risks data loss
- Please back up your data to your personal machine

Plotting, exploring, and explaining graphs (1/2)

- **Graphs are an essential part of your lab submissions**
- Graphs make patterns in data accessible visually
  - They represent **hypotheses** in data exploration
  - They make **arguments** in data presentation and explanation
- When explaining graphs, focus on trends, inflection points, and surprising artifacts
  - Partition graph into regions of similar behaviour
  - Label and annotate inflection points
  - Explain why different partitions behave the way they do
- Quality of presentation is really important
  - Ensure that they are clearly labeled – axes, legend, etc.
  - Think carefully about what axes and scales to use
  - Visual comparison is key – present data on the same plot, or in stacked plots, if you want to invite comparison
    - E.g., ensure that the reader can **see** the relationship in your plots

Plotting, exploring, and explaining graphs (2/2)
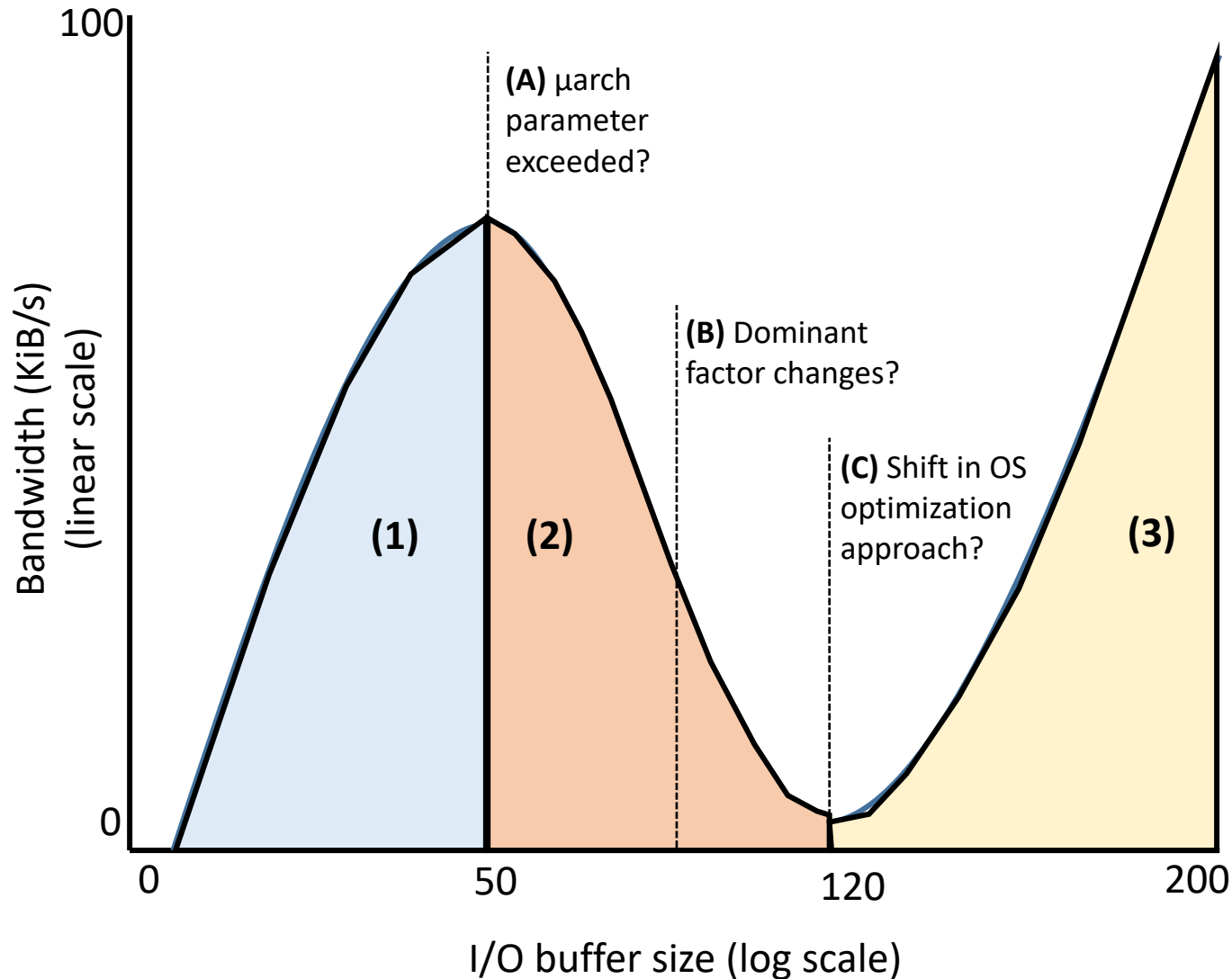
Why does (1) rise in the way that it does?

What happens at (A)?

Why does (2) sink in the way that it does?

What happens at (B)?

Why does (3) rise in the way that it does?

What happens at (C)?

How to contact us

- Preferred: Course slack
  - advopsys.slack.com

- Also possible: Email to the lecturer
  - robert.watson@cl.cam.ac.uk