Advanced Operating Systems: Lab 1 - I/O

Lecturelet 1

Dr Robert N. M. Watson

2020-2021

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 website

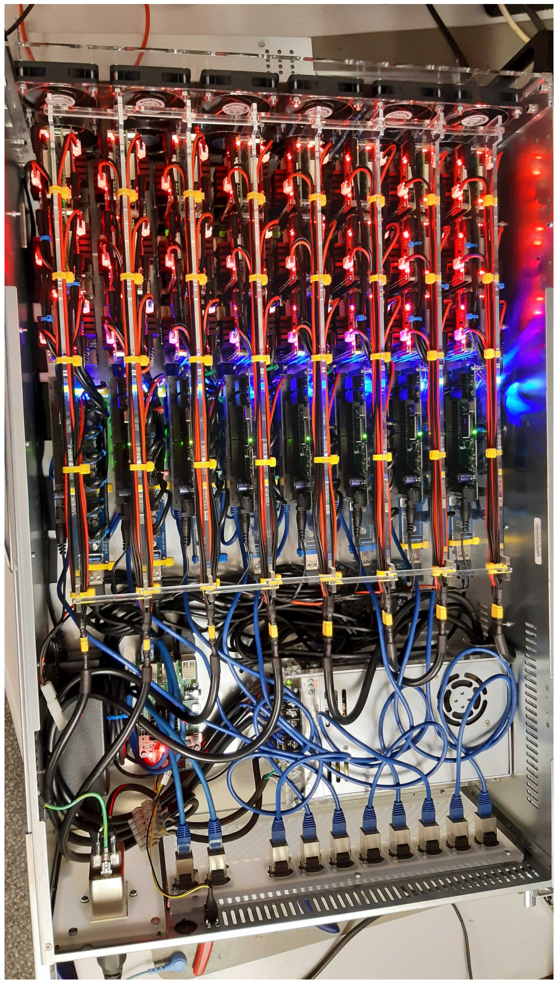
- We have provided several documents you will need from the module website:

• Lab Setup Guide	(Everyone)
• Lab 1 – I/O – Lab Description	(Everyone)
• Lab 1 – I/O – Part II Assignment	(Part II)
• Lab 1 – I/O – L41 Assignment	(L41)

Lab 1 – I/O

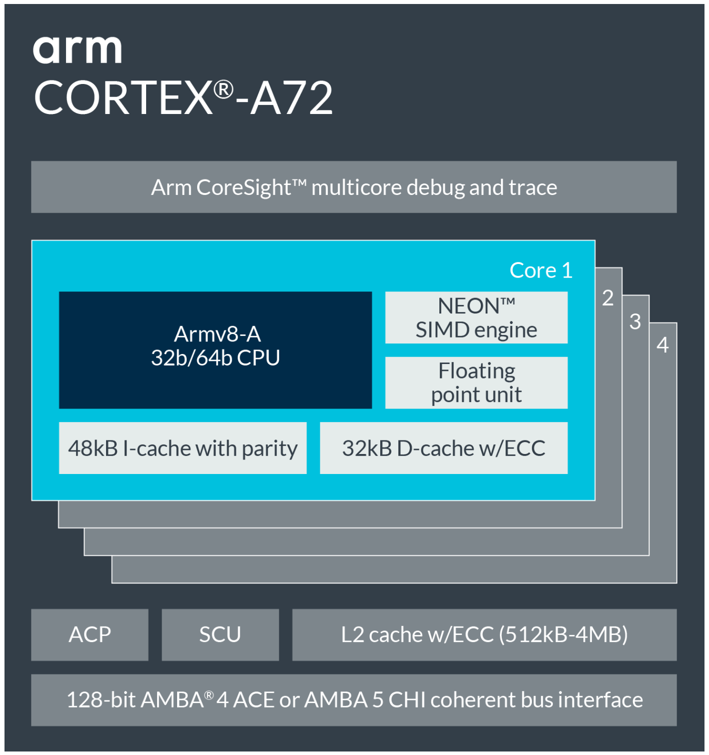
- This lab represents only a small number of assessed marks – it is intended as a learning exercise before we hit more complex topics:
  - Introduce our RPi4/FreeBSD environment
  - Explore user-kernel interactions via syscalls and traps
  - Engage with POSIX I/O and its implications
  - Measure the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

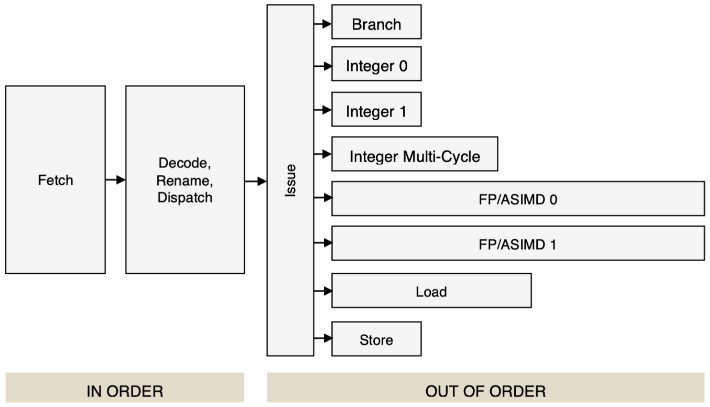
Our lab platform: RPi4s + FreeBSD 13.x



- 50x Raspberry Pi 4 boards
  - Broadcom BCM2711 SoC
  - 4x 64-bit A72 ARMv8-A cores
  - 8GB DRAM, 64G SD Card
- FreeBSD 13-STABLE (prerelease)
  - DTrace tracing tool
  - HWPMC counter framework
  - Bespoke benchmarks motivating OS and microarchitectural analysis
  - JupyterLab Notebook environment
- Remotely accessed via SSH + tunneling for JupyterLab

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





*** Our benchmarks use only the first core to simplify analysis**

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.

Per-Core:
L1 I-Cache: 48K

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.

Per-Core:
L1 D-Cache: 32K

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.

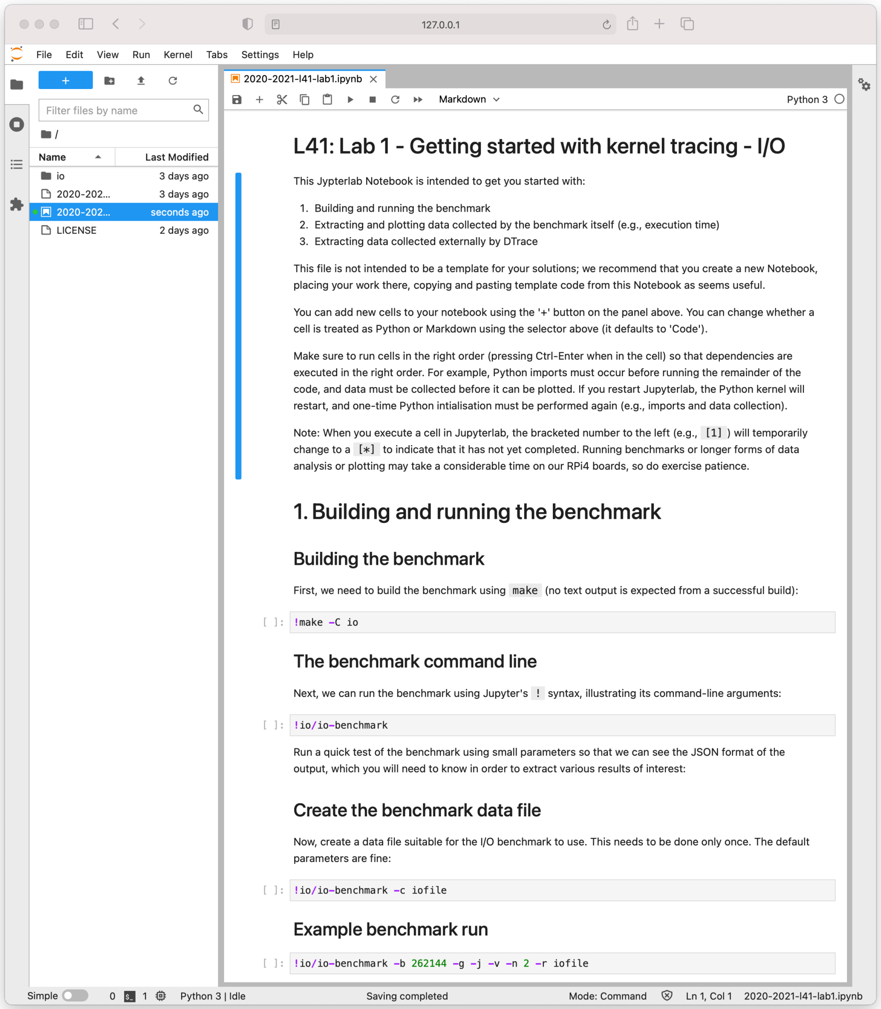
Shared:
L2 Cache: 1M

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

JupyterLab



L41: Lab 1 - Getting started with kernel tracing - I/O

This Jypterlab Notebook is intended to get you started with:

1. Building and running the benchmark
2. Extracting and plotting data collected by the benchmark itself (e.g., execution time)
3. Extracting data collected externally by DTrace

This file is not intended to be a template for your solutions; we recommend that you create a new Notebook, placing your work there, copying and pasting template code from this Notebook as seems useful.

You can add new cells to your notebook using the '+' button on the panel above. You can change whether a cell is treated as Python or Markdown using the selector above (it defaults to 'Code').

Make sure to run cells in the right order (pressing Ctrl-Enter when in the cell) so that dependencies are executed in the right order. For example, Python imports must occur before running the remainder of the code, and data must be collected before it can be plotted. If you restart Jupyterlab, the Python kernel will restart, and one-time Python intialisation must be performed again (e.g., imports and data collection).

Note: When you execute a cell in Jupyterlab, the bracketed number to the left (e.g., `[1]`) will temporarily change to a `[*]` to indicate that it has not yet completed. Running benchmarks or longer forms of data analysis or plotting may take a considerable time on our RPi4 boards, so do exercise patience.

1. Building and running the benchmark

Building the benchmark

First, we need to build the benchmark using `make` (no text output is expected from a successful build):

```
[ ]: !make -C io
```

The benchmark command line

Next, we can run the benchmark using Jupyter's `!` syntax, illustrating its command-line arguments:

```
[ ]: !io/io-benchmark
```

Run a quick test of the benchmark using small parameters so that we can see the JSON format of the output, which you will need to know in order to extract various results of interest:

Create the benchmark data file

Now, create a data file suitable for the I/O benchmark to use. This needs to be done only once. The default parameters are fine:

```
[ ]: !io/io-benchmark -c iofile
```

Example benchmark run

```
[ ]: !io/io-benchmark -b 262144 -g -j -v -n 2 -r iofile
```

- Web-based interactive Python(++) environment
  - "Notebooks" contain code, text, data, and plots
- Part II students will submit generated PDFs of their notebooks
- L41 students will use notebook output in their Lab Reports
- Some work will need to be done outside of JupyterLab due to Python-DTrace limitations
  - Kernel stack traces require command-line DTrace

JupyterLab – the UI

Executed cell (number)

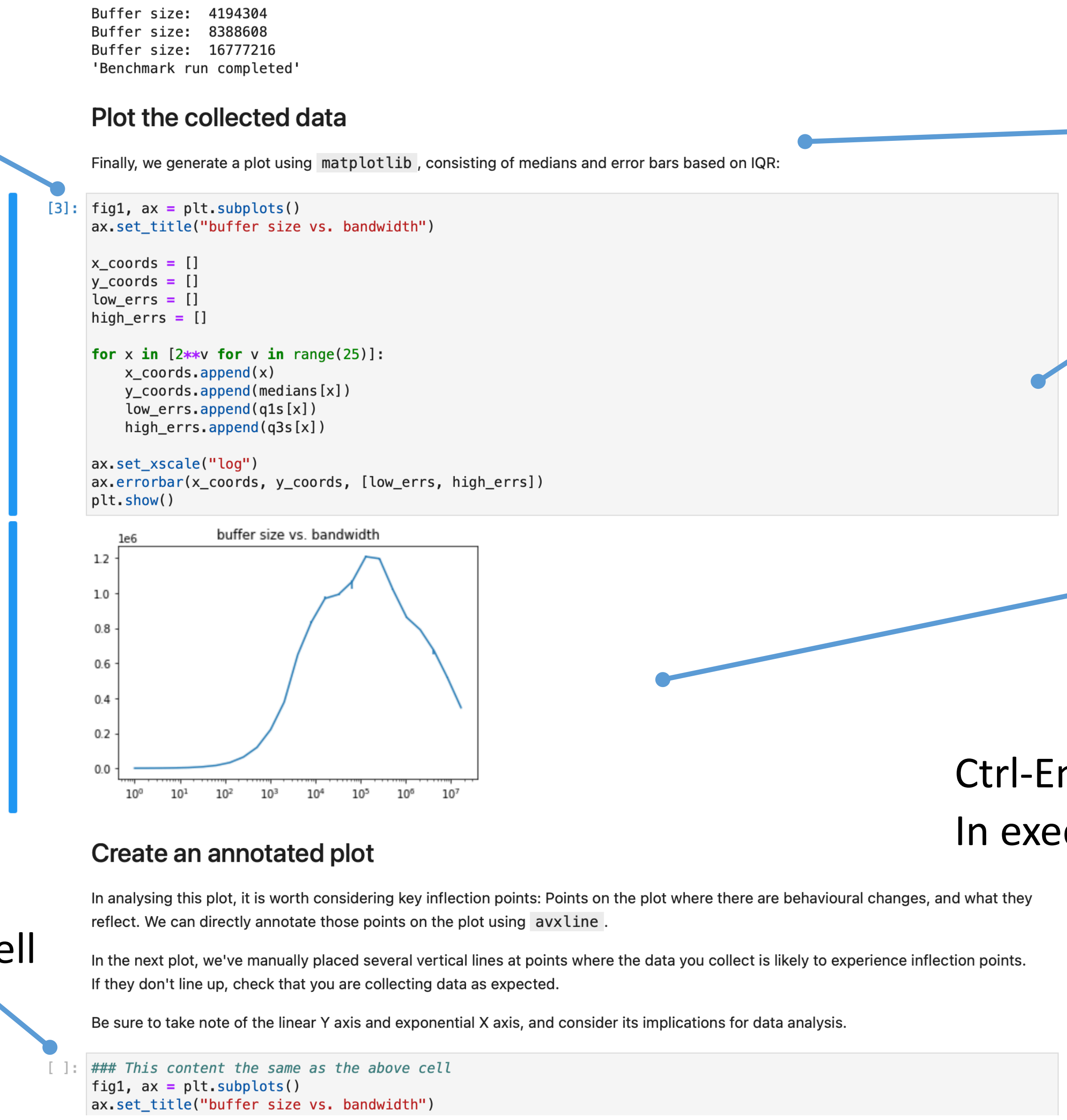
Current cell

Unexecuted cell (no number)

Markdown cell

Code cell

Cell output



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

Plot the collected data

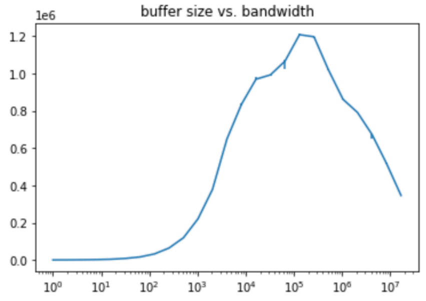
Finally, we generate a plot using `matplotlib`, consisting of medians and error bars based on IQR:

```
[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.

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

Ctrl-Enter in a cell executes it

In execution cells show [*]

Connecting to your board

- You will be contacted directly regarding your board assignment and login credentials
- The RPi4 nodes are accessible via SSH from within the CUDN (Cambridge University Data Network)
- If you are not directly connected, you can:
  - Use the UIS VPN
  - Use the CL VPN (if you have a CL account)
  - Connect via another system on the CUDN (e.g., ely)
- You will run all parts of the lab as the root user
- 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 two hypotheses:

1. System-call overhead is substantial. Reducing the number of system calls to perform the same work (e.g., by increasing I/O buffer sizes) will improve performance.
   Performance growth continues until we hit the system's peak I/O throughput, at which point it will stabilise.
2. The probe effect associated with DTrace is negligible.

io-benchmark

- Two operational modes used in this lab:

Create (`-c`)	Create a new benchmark data file
Read (`-r`)	Perform `read()`s against data file

- Adjust operational parameters:

Block size (`-b`)	Block size used for each I/O
Iterations (`-n`)	How many times to run
getrusage() (`-g`)	Report block-I/O statistics

- Output flags:

JSON (`-j`)	Generate machine-readable output
Quiet (`-q`)	Suppress all 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 :::
  - 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

How to contact us

- Preferred: Course slack
  - advopsys.slack.com
- Also possible: Email to the lecturer
  - robert.watson@cl.cam.ac.uk