Advanced Operating Systems: Lab 2 - IPC

Lecturelet 2

Prof. Robert Watson

2026-2027

Lab 2 objectives

- Use, and extend, the skills developed in Lab 1
- Trace user-kernel interactions via syscalls and traps
- Explore the performance of pipe and shared memory IPC
- Use DTrace and hardware performance counters (HWPMC) to analyse these properties

Overall: Let's apply our newly learned skills to some more interesting root-cause analysese of complex OS behaviour …

New documents

- *Advanced Operating System: Hardware Performance Counters (HWPMC)*
  - Introduction to performance counters in this lab
  - You may wish to refer to the ARMv8-A and A72 manuals (or not)

- *2x lab documents:*
  - *Advanced Operating Systems: Lab 2 – IPC – General information*
  - *Advanced Operating Systems: Lab 2 – IPC – Assignment*

Please make sure you have also read the document, *Advanced Operating Systems: Lab Exercises*, which provides information on our expectations, assessment, and marking

Lab Exercises document

Style and presentation

- **Presentation**: Lab exercise answers should be clearly written, spell checked, and formatted to make them easy for the reader to follow.
- **Focus**: Given the potential open endedness of the questions, it is important to keep answers focused. Don't write a journal of all the things you tried on the way to the answer; instead, focus on key points that are **important** or **interesting**, considering where possible root-cause analyses.
- **Length**: No specific maximum length is prescribed, but if excessive plots, data, script source code, and/or other elements are included, then they will be disregarded for the purposes of marking. Submissions should be succinct, answering the question in adequate detail, but avoiding detours or unnecessary detail or elaboration.
- **Figures and data tables**: Do include succinct and clearly labeled figures and summary statistics where they improve clarity of explanation, but avoid presenting voluminous data sets unless they are essential to your explanation.
- **Graphs**: Particular attention should be paid to graphs that present the results: axes must be labelled, scales should be selected with care to avoid misunderstanding, and if, for example, there are clear artefacts of interest, then an additional graph may be appropriate to explore those in greater detail. Where comparisons of graphs are important to the argument being made, it may be useful to align the plots as well as share axis ranges and scales to make the visiual comparison easier.
- **Graph descriptions**: All graphs must be described in the body of the text, and if not adjacent to that text, must also have a suitable (but brief) descriptive caption. Don't describe every detail of the graph: Instead help the reader understand how the graph supports the argument you are making.
- **Error**: In general, it will be important to include error bars or other error information, and explain when confidence intervals have been used.

Please do make sure to review this document, as it contains important guidance on what submitted work should include, as well as on the marking criteria.

Rough framing

- Inter-Process Communication (IPC) is an essential component to using the Process Model
  - Isolated boxes that can't talk to anything aren't very useful
- IPC design considers both semantics and performance:
  - Message passing vs shared memory?
  - Byte stream vs discrete messages?
  - Synchronous vs. asynchronous?
  - Portability to other OSes, communication semantics?
- Many years of research into two intertwined question:
  - What is the best IPC API?
  - How can we make it perform well?
- Once there is a defined API .. OS designers try to find the most efficient implementation

Use DTrace and performance counters to compare the behaviour of two distinct IPC types in UNIX: Pipes and shared memory IPC

Optimising IPC using kernel page borrowing

- Pipe API specifies **copy semantics**
  - Two endpoints both represented by file descriptors, one to send, the other two receive
  - Once a write(2) call returns, changes to memory in the sender do not affect data received in the recipient via read(2)
  - Practical implementation – copy two times
    1. From userspace sender buffer to kernel buffer (copyin(9))
    2. From kernel buffer to userspace recipient buffer (copyout(9))
  - But memory copying is known to be expensive with both historic and contemporary microarchitectures
- In 1996, John Dyson implemented VM optimisations for bulk pipe data transfer for FreeBSD
  - Remove sender copy by “borrowing” pages for “large” sends
  - Later also adopted in macOS; similar optimisations elsewhere
- Our lab will compare **pipes**, with implicit virtual-memory use, to explicit **shared memory IPC**

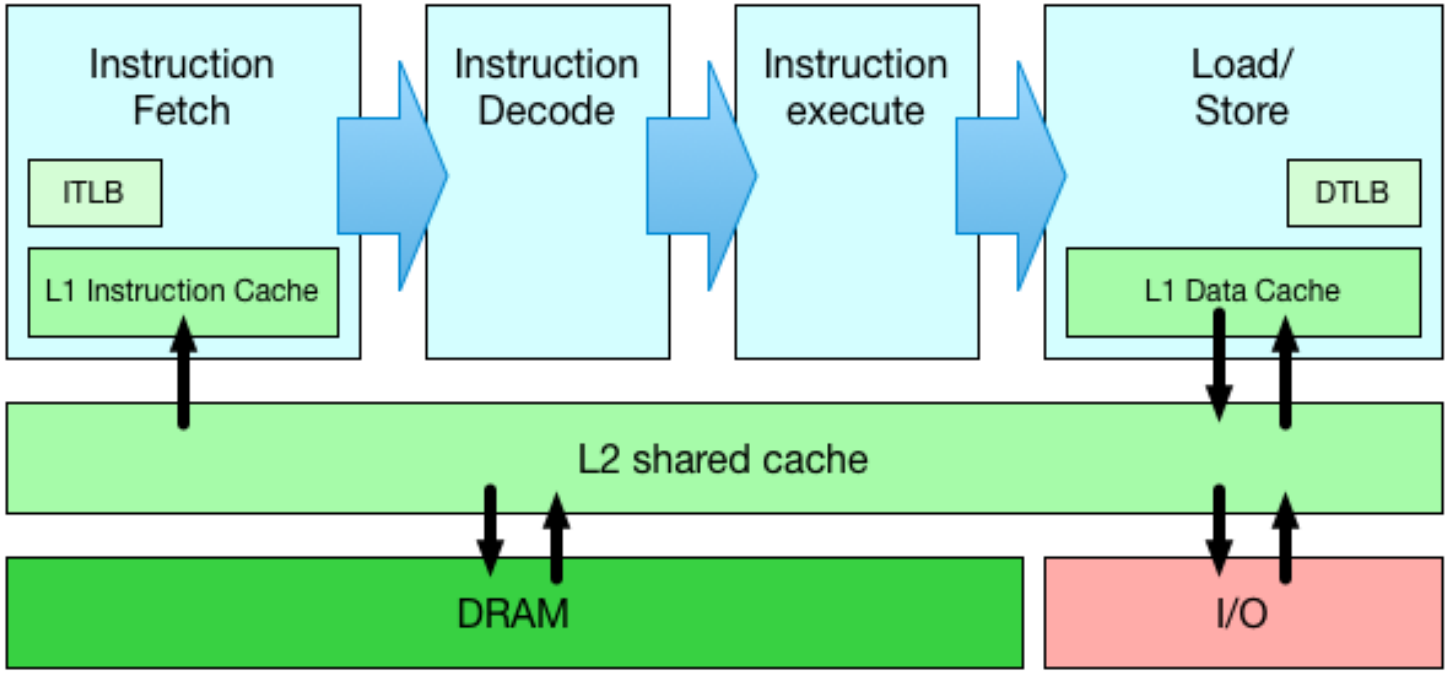
Hardware performance counters

Hardware performance counters (1/2)

- Seems simple enough:
  - Source code compiles to instructions
  - Instructions are executed by the processor
  - Fewer instructions → better performance?
- But some instructions take longer than others:
  - Register-register operations generally single-cycle (or less)
  - Multiply and divide may depend on the specific numeric values
  - Floating point may take quite a while
  - Loads/stores cost different amounts depending on TLB/cache use
- **Instruction count is [frequently] not a good way to understand computational expense**

Sketch of ARM Cortex A-8 memory hierarchy
(This is not the CPU you are using, just an illustration!)

- **Architectural** refers to an ISA-level view of execution
- **Micro-architectural** refers to behaviours below the ISA



- The performance of software depends on interactions throughout the hardware design, not just the pipeline.
- Yet architectural events are very important: They tell us what the software actually asked for.

* This is a very, very rough sketch indeed!

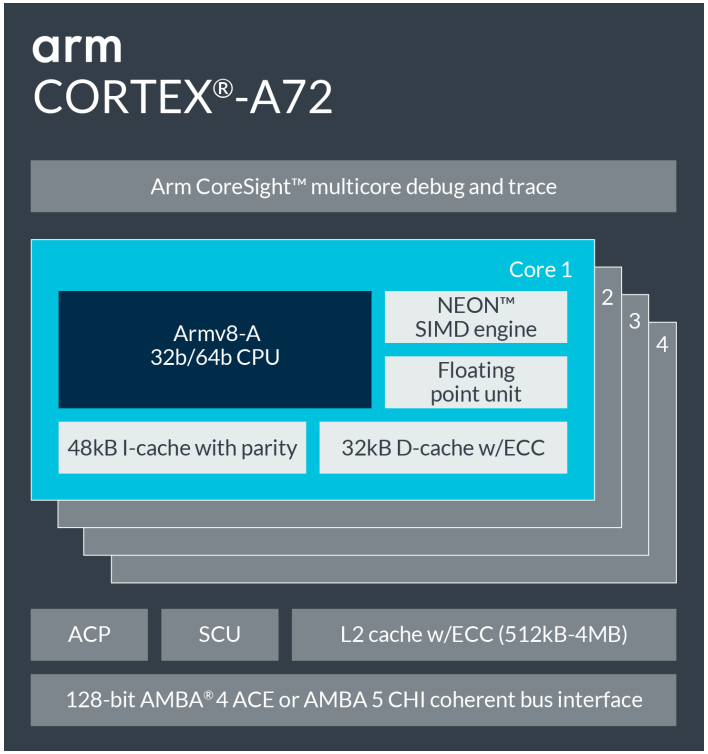
Hardware performance counters (2/2)

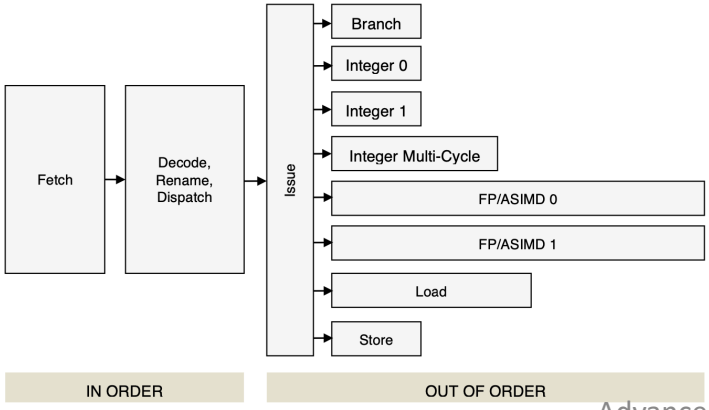
- Optimisation must take into account both architectural (algorithmic) behavior and microarchitectural behaviour
  - Count the actual numbers of instructions, branches taken, and so on to analyse software behaviour
  - Directly measure effects such as cache misses, branch-predictor misses, slower memory accesses, etc.
  - TLB/cache effects are hard to predict as they vary with memory footprint, memory subsystem, and also heuristics such as prefetching
- Hardware performance counters let us directly investigate architectural and micro-architectural events
  - **Architectural events**: #instructions, #load instructions, #store instructions, #branch returns, etc.
  - **Microarchitectural**: #bus accesses, #cache misses, #DRAM traffic, #branch mispredicts...

Reminder: High-density Cortex A-72 slide

(Some of this information will be useful only for later labs)

Unchanged from Lecture





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

Per-Core: L1 I-Cache: 48K

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 D-Cache: 32K

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.

Shared: L2 Cache: 1M

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.

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

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.

Using performance counters

- Recall:
  - **Architectural counters**: What software asks the hardware to do
  - **Microarchitectural counters**: How the hardware implements it
- Optimising software using performance counters is subtle
  - Often counter use leads to important micro-optimisations (e.g., “Cache lines are thrashing → lay out memory better”)
  - But must consider whether algorithmic optimisation is preferable to microarchitecture-centric tuning
- A few considerations when analysing “work”:
  - It may be preferable to ask the hardware to do less work, “inefficiently”, than to do more work, “efficiently”
  - It is hard to know whether a change is important (e.g., doubling TLB misses might be critical .. Or irrelevant .. Context is required)
  - Microarchitecturally-aware optimisations may tune well for one specific microarchitecture, yet perform badly on another
- **Microarchitecturally motivated optimisations must be carefully evaluated, ideally across >1 microarchitectures**

The benchmark

The benchmark (1/2)

```
root@rpi4-000:/data # ipc/ipc-benchmark
ipc-benchmark [-Bgjqsv] [-b buffersize] [-i pipe|local|tcp|shmem]
    [-n iterations] [-p tcp_port] [-P arch|dcache|instr|tlbmem]
    [-t totalsize] mode

Modes (pick one - default 2thread):
    2thread     IPC between two threads in one process
    2proc       IPC between two threads in two different processes
    describe    Describe the hardware, OS, and benchmark configurations

Optional flags:
    -B                          Run in bare mode: no preparatory activities
    -g                          Enable getrusage(2) collection
    -i pipe|local|tcp|shmem     Select pipe, local sockets, TCP, or shared memory
                                (default: pipe)
    -j                          Output as JSON
    -p tcp_port                 Set TCP port number (default: 10141)
    -P arch|dcache|instr|tlbmem  Enable hardware performance counters
    -q                          Just run the benchmark, don't print stuff out
    -s                          Set send/receive socket-buffer sizes to buffersize
    -v                          Provide a verbose benchmark description
    -b buffersize               Specify the buffer size (default: 131072)
    -n iterations               Specify the number of times to run (default: 1)
    -t totalsize                Specify the total I/O size (default: 16777216)
```

- Simple, bespoke IPC benchmark: pipes, sockets, TCP, and shared memory
- Adjust user and kernel buffer sizes

The benchmark (2/2)

- Use only one of its operational modes:

  `2proc` IPC between two processes

- Adjust IPC parameters:

  `-b` *`buffersize`* Set user IPC buffer size

  `-i` *`pipe`* **`or`** *`shmem`* Use pipes or shared memory

  `-P` *`mode`* Configure HWPMC

- Output flags:

  `-g` Display getrusage(1) statistics

  `-j` Output as JSON

  `-v` Verbose output (more configuration detail)

Performance counter modes

- Normally we run an external tool to use counters, such as FreeBSD's pmcstat(8) or Linux's perf tool
  - We have adapted the benchmark to use libpmc
  - We use only counting mode, not sampling mode
- The A-72 has six counter registers that can be used at once
  - We always enable **instruction counting** and **cycle counting**
  - The other 4 are used for specific groups of counters:

-P *mode*	Category
arch	Architectural (ISA-level) statistics (some speculative*)
dcache	L1-D and L2 cache statistics
instr	L1-I and branch-prediction statistics
tlbmem	D-TLB / I-TLB and memory access/bus access statistics

- We recommend using the **arch** and **tlbmem** counter sets
- The probe effect affects hardware counters, too!

*Non-speculative counters can be quite expensive in the microarchitecture for superscalar processors, so Arm has chosen not to provide architectural counters

root@rpi4-046:/data # **ipc/ipc-benchmark -g -i pipe -j -P arch -v 2proc**

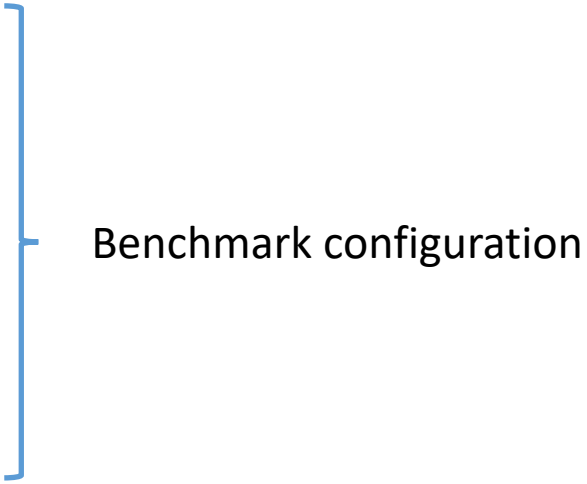

```
{"hardware_configuration": {
   "hw.machine": "arm64",
   "hw.model": "ARM Cortex-A72 r0p3",
   "hw.ncpu": 4,
   "hw.physmem": 8419033088,
   "hw.pagesizes": [{"pagesize": 4096},
    {"pagesize": 2097152}, {"pagesize": 1073741824}],
   "hw.cpufreq.arm_freq": 600000000
 },  "os_configuration": {
   "kern.ostype": "FreeBSD",
   "kern.osrelease": "13.0-STABLE",
   "kern.ident": "ADVOPSYS",
   "kern.hostname": "rpi4-000"
 },  "network_ipc_configuration": {
   "kern.ipc.pipe_mindirect": 8192,
   "kern.ipc.maxsockbuf": 33554432,
   "ifnet.name": "lo0",
   "ifnet.mtu": 16384,
   "net.inet.tcp.cc.algorithm": "newreno",
   "net.isr.bindthreads": 1,
   "net.isr.defaultqlimit": 256
 },
```


Hardware configuration

OS configuration

Network/IPC configuration

```
"benchmark_configuration": {
  "buffersize": 131072,
  "totalsize": 16777216,
  "msgcount": 128,
  "mode": "2proc",
  "ipctype": "pipe",
  "pmctype": "arch",
  "iterations": 1
},
```

```
"benchmark_samples": [
  {
    "bandwidth": 609733.59,
    "time": "0.026870752",
    "stime": "0.023513",
    "utime": "0.000165",
    "msgsnd": 128,
    "msgrcv": 256,
    "nvcsw": 523,
    "nivcsw": 0,
    "INST_RETIRED": 7807526,
    "CPU_CYCLES": 10659620,
    "LD_SPEC": 2776279,
    "ST_SPEC": 1675676,
    "EXC_RETURN": 458,
    "BR_RETURN_SPEC": 135871,
    "CYCLES_PER_INSTRUCTION": 1.365301
  }
 ]
}
```


Performance / wallclock time

Sampled execution time in userlevel/kernel

getrusage(2) statistics

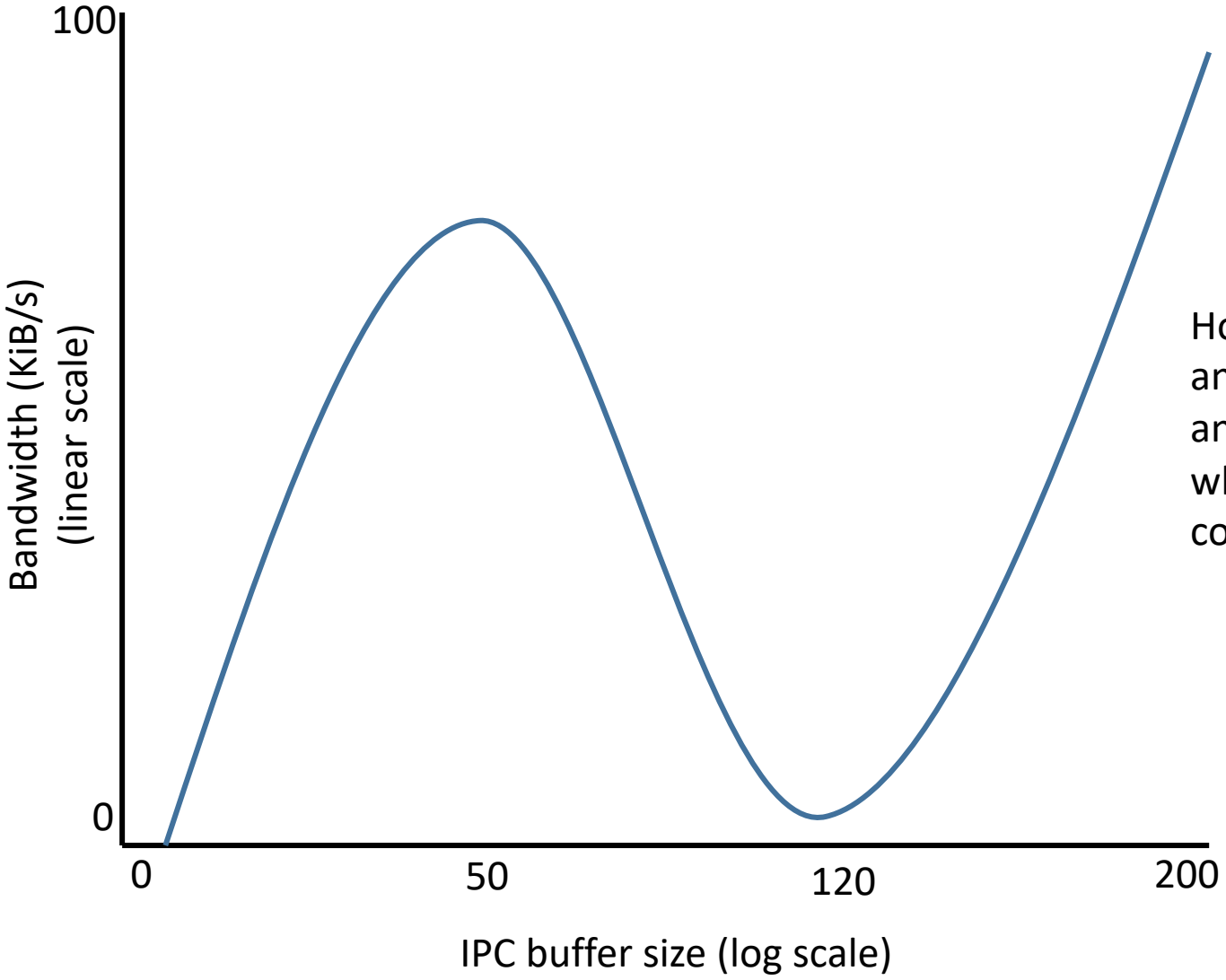
Hardware performance counters
(and derived metrics)

Plots

Plotting, exploring, and explaining graphs (1/3)

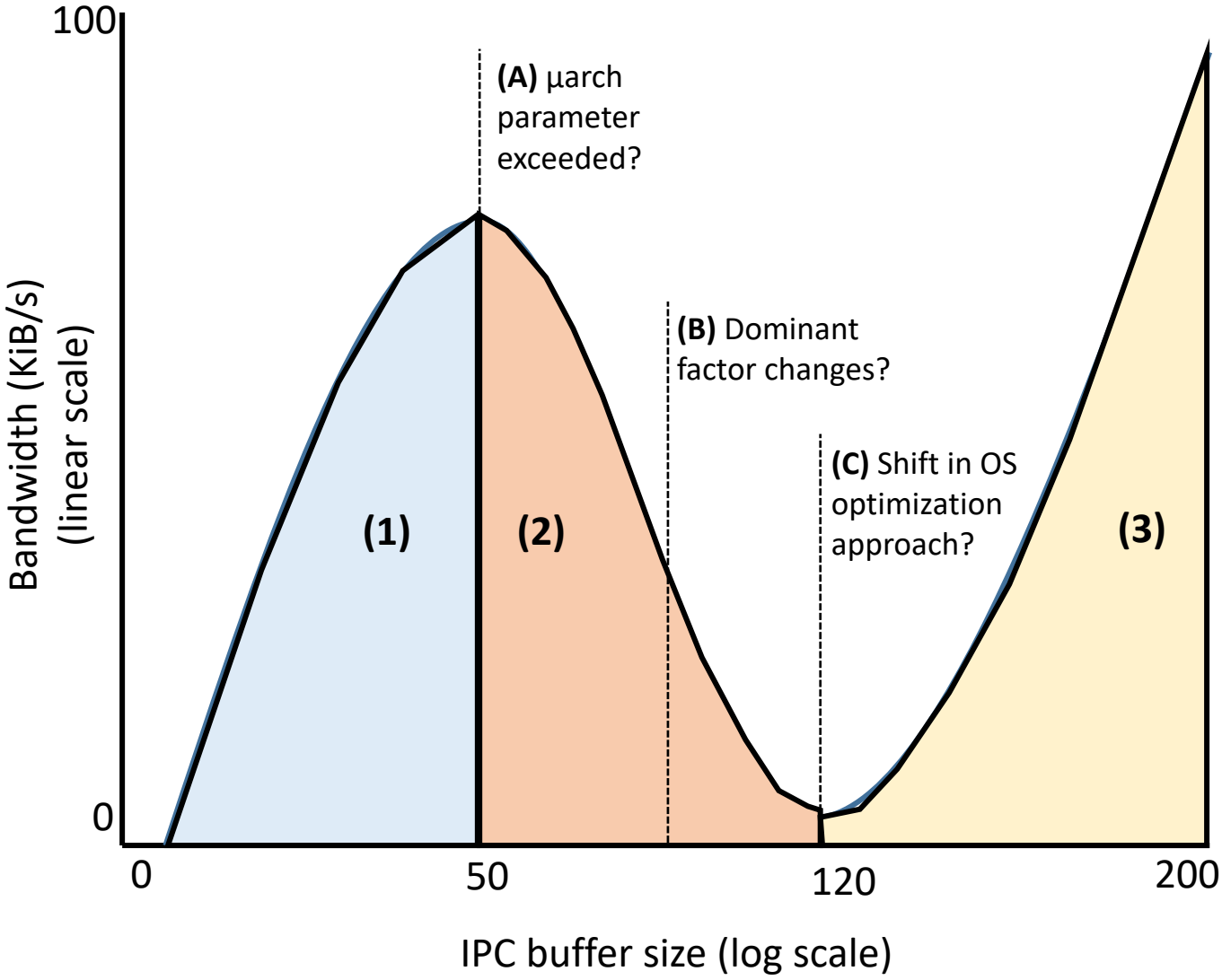
- **Plots are an essential part of your lab submissions**
- Plots make patterns in data accessible visually
  - They represent **hypotheses** in data exploration
  - They make **arguments** in data presentation and explanation
- When explaining graphs, focus on **trends**, **inflection points**, and **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 critical
  - 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 (2/3)



How should we explore and explain this graph – and how do we know what more data to collect to do so?

Plotting, exploring, and explaining graphs (3/3)



Partitioning plot and analysing their inflection points can be valuable:

- 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)?

You will sometimes want to consider both first and second derivatives.

Conclusion

A few concluding thoughts

- You are now (fairly) familiar with:
  - DTrace as an instrumentation tool
  - JupyterLab as a data collection, analysis, presentation tool
- You will now pick up new skills:
  - Further DTrace experience – e.g., system-call provider, profile provider, perhaps scheduling provider, etc.
  - Performance counter experience (can be hard to interpret…)
- When gathering and analysing data:
  - Start with short runs (even –n 1) to allow quick iteration
  - Plot data to understand its behaviour
  - Pay attention to inflection points, regions of commonality
  - Mark up graphs with key hardware, software thresholds
  - Remember that the cache/TLB footprint of a workload will (almost certainly) not be the benchmark buffer size
- We are now doing comparative analysis…

How to contact us

- Attend the lab!

- Course slack outside of lab hours, or if unable to join
  - advopsys.slack.com

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