Advanced Operating Systems:
Lab 2- IPC

Lecturelet 2

Dr Robert Watson

2020-2021

Lab 2 objectives

- Consolidate and extend skills developed in Lab 1

- Continue to gain experience tracing user-kernel interactions via system calls and traps

- Explore the performance of varying IPC models and buffer sizes

- Use DTrace and hardware performance counters (HWPMC) to analyse these properties

- Generate data to complete the second lab assignment

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)

- *Advanced Operating Systems: Lab 2 – IPC*
- *Part II - Advanced Operating Systems: Lab 2 – IPC*
- *L41 - Advanced Operating Systems: Lab 2 – IPC*

- **Important**: The two assignments are substantially more different than they were in Lab 1
- However, L41 students might find the Part II assignment useful to think about potential analysis strategies

Rough framing

- Inter-Process Communication (IPC) is an essential component to using the Process Model
  - Isolated boxes that can't talk to anyone aren't very useful
- There are many design dimensions to an IPC primitive relating to application semantics and performance
  - Message passing vs shared memory?
  - Stream vs datagram?
  - Synchronous vs. asynchronous?
  - Portability to other OSes, communication semantics?
- You will compare two such primitives to understand its performance behaviour
- Use DTrace and hardware performance counters to explain .. Surprising? .. performance artifacts

The benchmark

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

Modes (pick one - default 1thread):
    1thread                IPC within a single thread
    2thread                IPC between two threads in one process
    2proc                  IPC between two threads in two different processes

Optional flags:
    -B                     Run in bare mode: no preparatory activities
    -g                     Enable getrusage(2) collection
    -i pipe|local|tcp      Select pipe, local sockets, or TCP (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 and sockets
- Statically linked
- Adjust user and kernel buffer sizes

The benchmark (2)

- Use only one of its operational modes:
  - 2thread   IPC between two threads in one process

- Adjust IPC parameters:
  - `-b size`   Set user IPC buffer size
  - `-i pipe`   Use pipe() IPC   (**L41 only**)
  - `-i local`   Use socketpair() IPC
  - `-P mode`   Configure HWPMC
  - `-s`   Set socket-buffer size   (**Part II only**)

- Output flags:
  - `-g`   Display getrusage(1) statistics
  - `-j`   Output as JSON
  - `-v`   Verbose output (more configuration detail)

Hardware performance counters (1/2)

- Seems simple enough:
  - Source code compiles to instructions
  - Instructions are executed by the processor

- 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

Hardware performance counters (2/2)

- Optimisation is therefore not just about reducing instruction count
  - Optimisation must take into account micro-architectural effects
  - TLB/cache effects tricky as they vary with memory footprint
  - How can we tell when the cache overflows?

- Hardware performance counters let us directly investigate architectural and micro-architectural events
  - #instructions, #memory accesses, #cache misses, DRAM traffic...

Performance counter modes

- We have adapted the benchmark to use libpmc
- We use only counting mode, not sampling mode
- The A-72 supports up to six counters enabled at a time
  - We always enable instruction 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

- You will need to run the benchmark for each counter set
  - … But it is reasonable to limit to one iteration each
- 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@rpi4-000:/data # ipc/ipc-benchmark -g -i local -j -P arch -v 2thread
{
  "host_configuration": {
    "hw.machine": "arm64",
    "hw.model": "ARM Cortex-A72 r0p3",
    "hw.ncpu": 4,
    "hw.physmem": 8419667968,
    "hw.pagesizes": {
      "pagesize": 4096,
      "pagesize": 2097152,
      "pagesize": 1073741824
    },
    "hw.cpufreq.arm_freq": 600000000
  },
  "benchmark_configuration": {
    "buffersize": 131072,
    "totalsize": 16777216,
    "msgcount": 128,
    "mode": "2thread",
    "ipctype": "local",
    "pmctype": "arch",
    "iterations": 1
  },
```

System configuration

Benchmark configuration

```json
"benchmark_samples": [
  {
    "bandwidth": 4944.31,
    "time": "3.313705598",
    "INST_RETIRED": 1187796464,
    "CPU_CYCLES": 1448312762,
    "LD_SPEC": 414032426,
    "ST_SPEC": 228205020,
    "EXC_RETURN": 1484103,
    "BR_RETURN_SPEC": 24429746,
    "CYCLES_PER_INSTRUCTION": 1.219327,
    "utime": "0.007958",
    "stime": "3.305383",
    "msgsnd": 128,
    "msgrcv": 2048
  }
]
}
```

Performance / wallclock time

Hardware performance counters
(and derived metrics)

Sampled execution time in userlevel/kernel

Getrusage(2) statistics

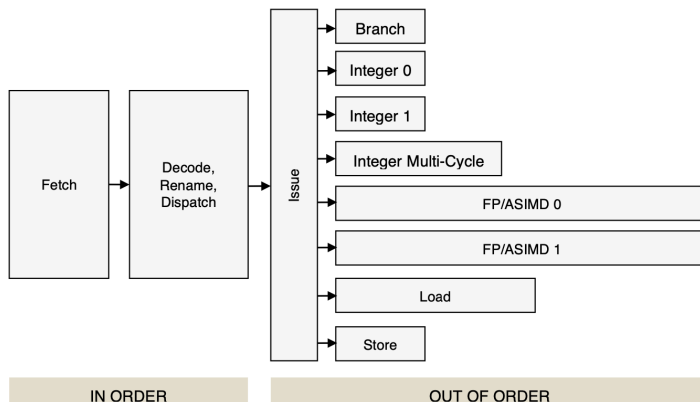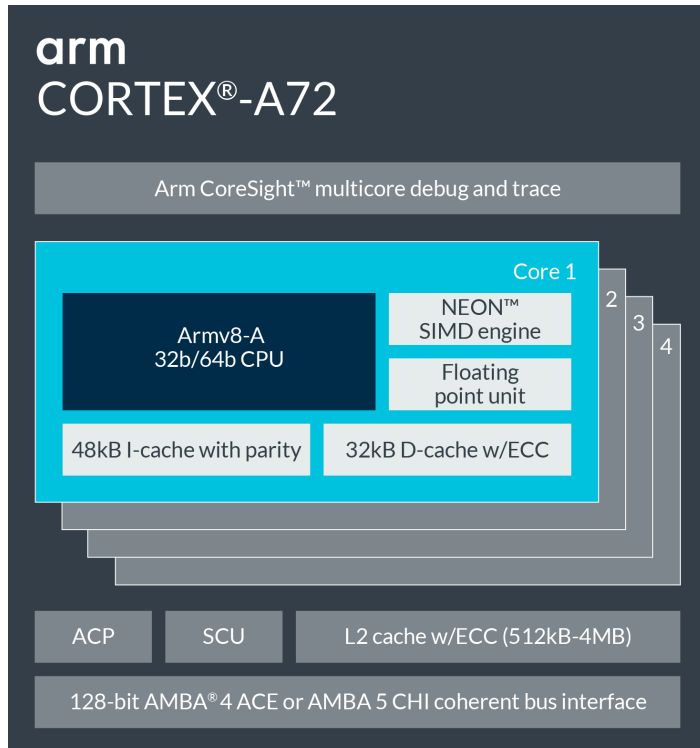# Sketch of ARM Cortex A-8 memory hierarchy
(This is not the CPU you are using, just an illustration!)

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

This is a very, very rough sketch indeed!
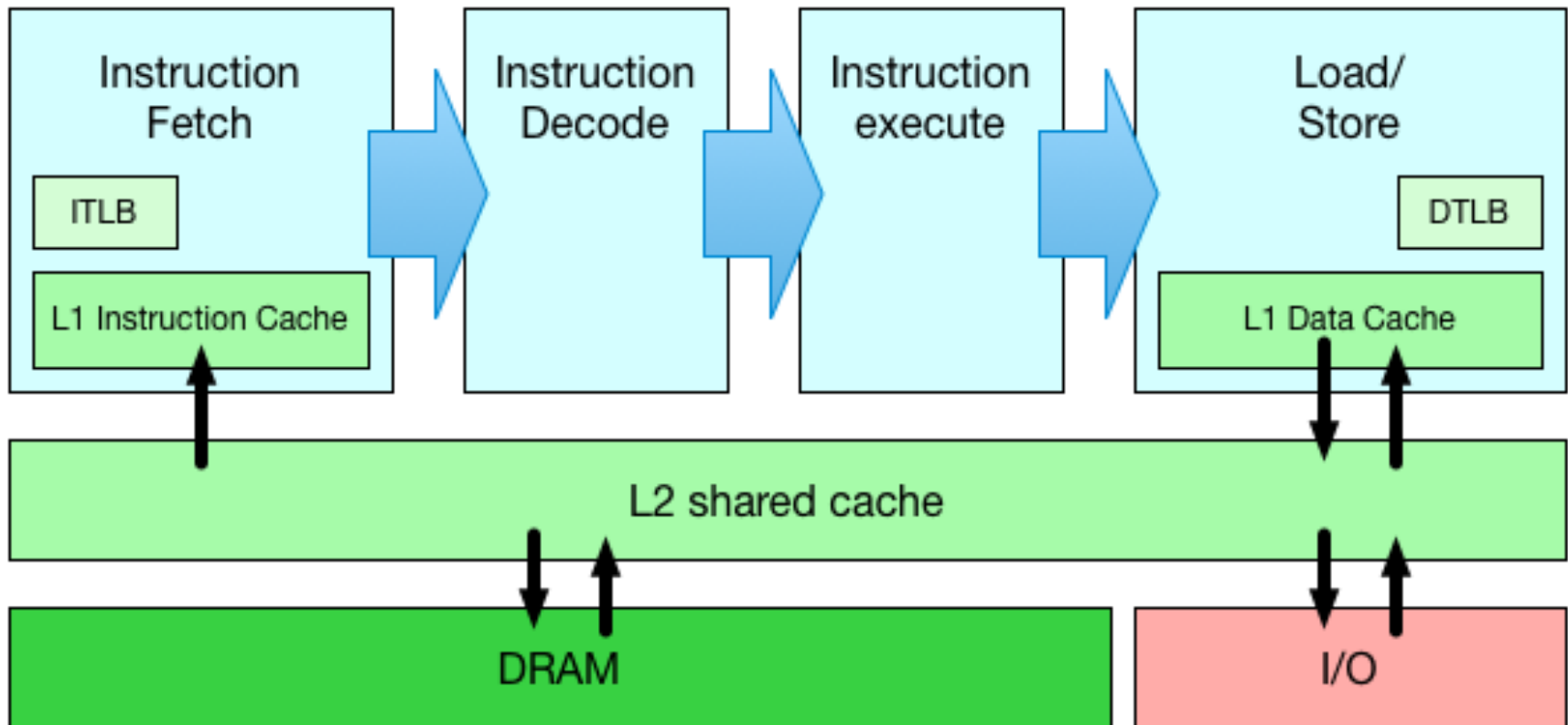
Reminder: High-density Cortex A-72 slide
(Some of this information will be useful only for later labs)

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

* Our benchmarks use only the first core to simplify analysis

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., perhaps the system-call provider, scheduling provider, etc.
  - Performance counter experience (can be hard to interpret…)
- When 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…