F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 — 2025/26

[0] No proprietary software has been used in producing these slides

Outline

Lecture 3: Memory Hierarchy

Memory Hierarchy: Introduction

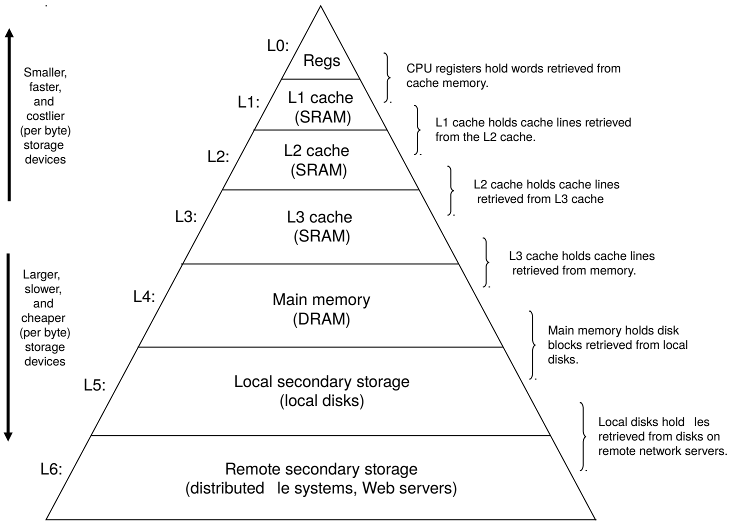
- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a **memory hierarchy**.

[0] Lecture based on Bryant & O'Hallaron, 3rd edition, Chapter 6

Memory Hierarchy

- Our view of the main memory so far has been a **flat** one, ie.
- access time to all memory locations is constant.
- In modern architecture this is **not** the case.
- In practice, a memory system is a **hierarchy of storage devices** with different capacities, costs, and access times.
- CPU registers hold the most frequently used data.
- Small, fast cache memories nearby the CPU act as staging areas for a subset of the data and instructions stored in the relatively slow main memory.
- The main memory stages data stored on large, slow disks, which in turn often serve as staging areas for data stored on the disks or tapes of other machines connected by networks

Caches and Memory Hierarchy


Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0: Regs
L1: L1 cache (SRAM)
L2: L2 cache (SRAM)
L3: L3 cache (SRAM)
L4: Main memory (DRAM)
L5: Local secondary storage (local disks)
L6: Remote secondary storage (distributed le systems, Web servers)
CPU registers hold words retrieved from cache memory.
L1 cache holds cache lines retrieved from the L2 cache.
L2 cache holds cache lines retrieved from L3 cache
L3 cache holds cache lines retrieved from memory.
Main memory holds disk blocks retrieved from local disks.
Local disks hold les retrieved from disks on remote network servers.

Discussion

As we move from the top of the hierarchy to the bottom, the devices become **slower, larger, and less costly** per byte.

The main idea of a memory hierarchy is that **storage at one level serves as a cache for storage at the next lower level**.

Using the different levels of the memory hierarchy efficiently is crucial to achieving high performance.

Access to levels in the hierarchy can be explicit (for example when using OpenCL to program a graphics card), or implicit (in most other cases).

The importance of the memory hierarchy

- For the programmer this is important because data access times are very different:
  - Register: **0 cycles**
  - Cache: **1–30 cycles**
  - Main memory: **50–200 cycles**
- We want to store data that is frequently accessed **high in the memory hierarchy**

Locality

- **Principle of Locality**: Programs tend to use data and instructions with addresses near or equal to those they have used recently
- **Temporal locality:** Recently referenced items are likely to be referenced again in the near future.
- **Spatial locality:** Items with nearby addresses tend to be referenced close together in time

Locality Example: sum-over-array

```
ulong count; ulong sum;
for (count = 0, sum = 0; count<n; count++)
   sum += arr[count];
res1->count = count;
res1->sum = sum;
res1->avg = sum/count;
}
```

- **Data references**
  - Reference array elements in succession (stride-1 reference pattern). **spatial locality**
  - Reference variable sum each iteration. **temporal locality**
- **Instruction references**
  - Reference instructions in sequence. **spatial locality**
  - Cycle through loop repeatedly. **spatial locality**

Importance of Locality

Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer!

Which of the following two version of sum-over-matrix has better locality (and performance):

Traversal by rows:

```
int i, j;  ulong  sum;
for (i = 0; i<n; i++)
  for (j = 0; j<n; j++)
    sum += arr[i][j];
```

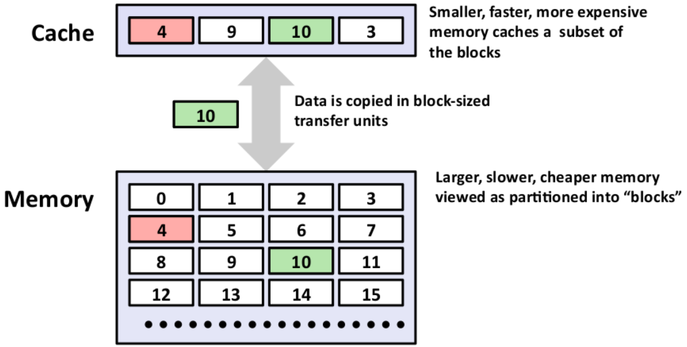
Traversal by columns:

```
int i, j; ulong  sum;
for (j = 0; j<n; j++)
  for (i = 0; i<n; i++)
  sum += arr[i][j];
```

Caches

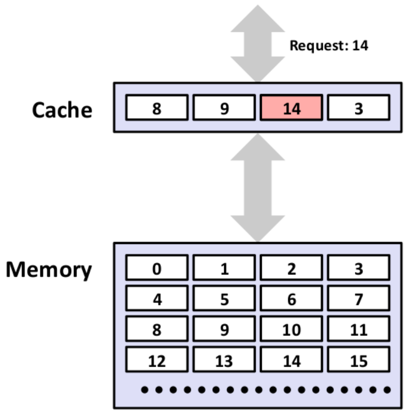
- **Cache:** A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level $k + 1$.
- Why do memory hierarchies work?
  - Because of locality, programs tend to access the data at level k more often than they access the data at level $k + 1$.
  - Thus, the storage at level $k + 1$ can be slower, and thus larger and cheaper per bit.
- **Big Idea:** The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

General Cache Concepts


Cache
4
9
10
3
Smaller, faster, more expensive memory caches a subset of the blocks
10
Data is copied in block-sized transfer units
Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper memory viewed as partitioned into "blocks"

[0] From Bryant and O'Hallaron, Ch 6

General Cache Concepts: Hit

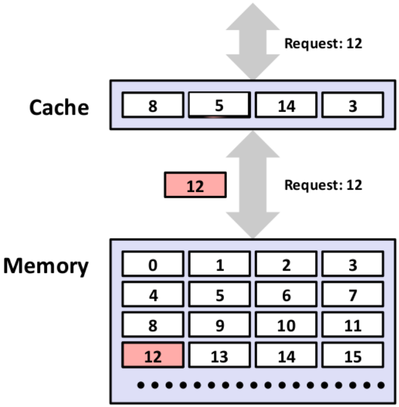


Data in block b is needed

Block b is in cache:
Hit!

[0]From Bryant and O'Hallaron, Ch 6

General Cache Concepts: Miss



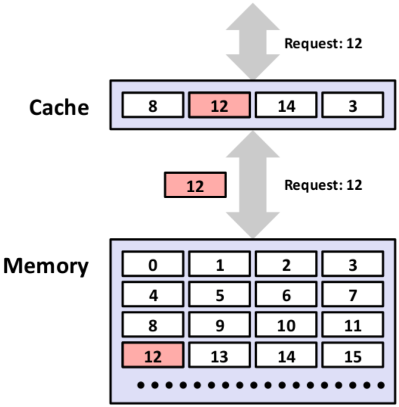
Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from memory

[0] From Bryant and O'Hallaron, Ch 6

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from memory

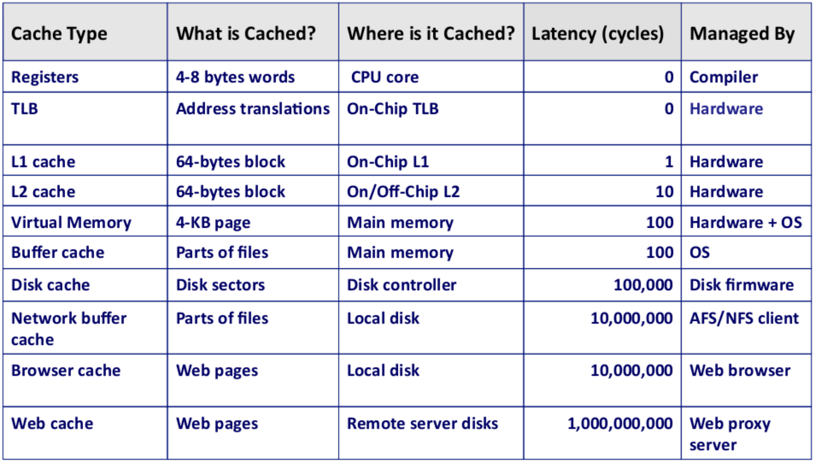
Block b is stored in cache
- Placement policy: determines where b goes
- Replacement policy: determines which block gets evicted (victim)

[0]From Bryant and O'Hallaron, Ch 6

Types of Cache Misses

- **Cold (compulsory) miss:**
  - Cold misses occur because the cache is empty.
- **Conflict miss:**
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - ⋆ E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - ⋆ E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.
- **Capacity miss:**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of Caching in the Memory Hierarchy



Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

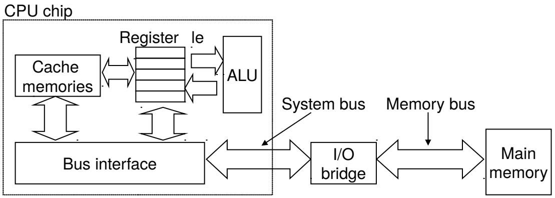
[0] From Bryant and O'Hallaron, Ch 6

Summary

- The speed gap between CPU, memory and mass storage continues to widen.
- Well-written programs exhibit a property called locality.
- Memory hierarchies based on caching close the gap by exploiting locality.

Principles of Caches

- Cache memories are small, fast SRAM-based memories managed automatically in hardware.
  - Hold frequently accessed blocks of main memory
- CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory.
- Typical system structure:



ARM Cortex A7 Cache Hierarchy

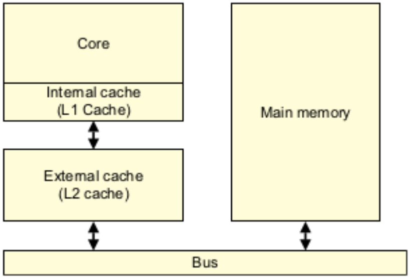


Figure 8-1 A basic cache arrangement

A cache is a small, fast block of memory that sits between the core and main memory. It holds copies of items in main memory. Accesses to the cache memory happen significantly faster than those to main memory. Because the cache holds only a subset of the contents of main memory, it must store both the address of the item in main memory and the associated data. Whenever the core wants to read or write a particular address, it will first look for it in the cache. If it finds the address in the cache, it will use the data in the cache, rather than having to perform an access to main memory. This significantly increases the potential performance of the system, by reducing the effect of slow external memory access times. It also reduces the power consumption of the system. NB: In many ARM-based systems, access to external memory will take 10s or 100s of cycles.

ARMv7-A Memory Hierarchy

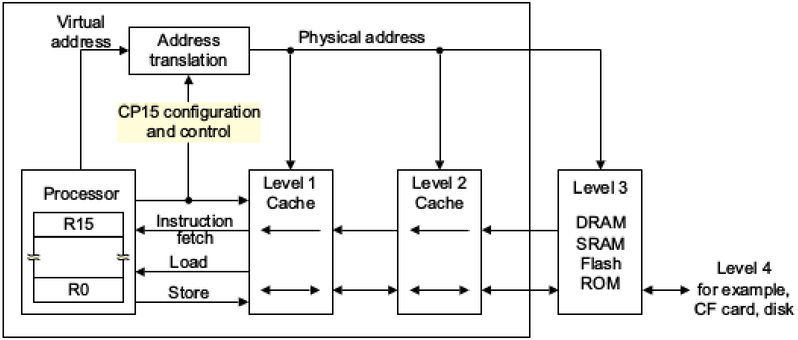


Figure A3-6 Multiple levels of cache in a memory hierarchy

See ARM Architcture Reference, Ch A3, Fig A3.6, p.157

Caching policies: direct mapping

- The caching policy determines how to map addresses (and their contents) in main memory to locations in the chache.
- Since the cache is much smaller, several main memory addresses will be mapped to the same cache location.
- The role of the caching policy is to avoid such clashes as much as possible, so that the cache can be used for most memory read/write operations.
- The simplest caching policy is a **direct mapped cache**:
  - each location in main memory always maps to a single location in the cache
  - this policy is simple to implement, and therefore requires little hardware
  - a weakness of the policy is, that if two frequently used memory addresses map to the same cache address, this results in a lot of cache misses ("**cache thrashing**")

Direct mapped cache

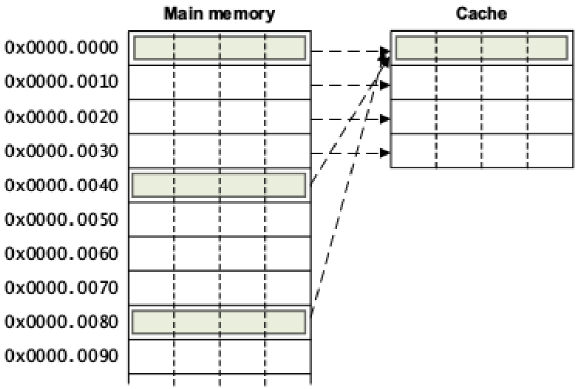


Figure 8-4 Direct mapped cache operation

[0]See ARM Programmer's Guide, Ch 8, Fig 8.4, p 113

Caching policies: set-associative

- To eliminate the weakness of the direct-mapped caches, a more flexible **set-associative** cache can be used.
- With this policy, one memory location can map to one of several *ways* in the cache.
- Conceptually, each way represents a slice of the cache.
- Therefore, a main memory address can be mapped to any of these slices in the cache.
- Inside one such slice, however, the location is fixed.
- If the system uses *n* such slices (“ways”) it is called an *n*-way associative cache.
- This avoids cache thrashing in cases where no more than *n* frequently used variables (memory locations) occur.

NB: The ARM Cortex A7 uses a 4-way set associative data cache, with cache size of 32kB, and a cache line size of 8 words.

Set-associative cache

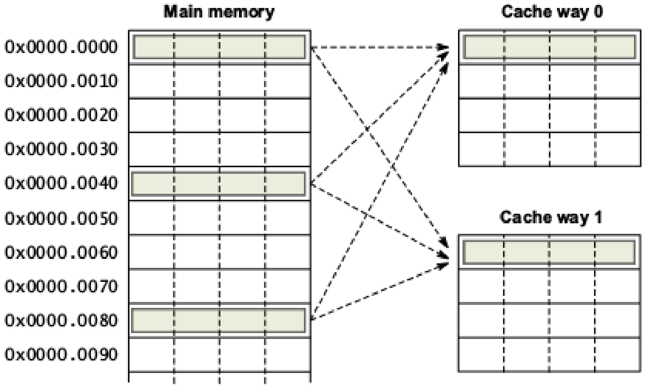
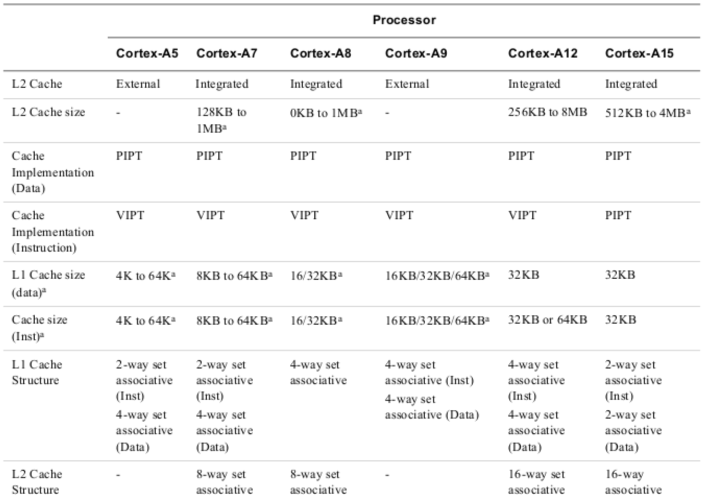


Figure 8-6 A 2-way set-associative cache

[0] See ARM Programmer's Guide, Ch 8, Fig 8.5, p 115

ARM cache features

Table 8-1 Cache features of Cortex-A series processors



	Processor					
	Cortex-A5	Cortex-A7	Cortex-A8	Cortex-A9	Cortex-A12	Cortex-A15
L2 Cache	External	Integrated	Integrated	External	Integrated	Integrated
L2 Cache size	-	128KB to 1MB[a]	0KB to 1MB[a]	-	256KB to 8MB	512KB to 4MB[a]
Cache Implementation (Data)	PIPT	PIPT	PIPT	PIPT	PIPT	PIPT
Cache Implementation (Instruction)	VIPT	VIPT	VIPT	VIPT	VIPT	PIPT
L1 Cache size (data)[a]	4K to 64K[a]	8KB to 64KB[a]	16/32KB[a]	16KB/32KB/64KB[a]	32KB	32KB
Cache size (Inst)[a]	4K to 64K[a]	8KB to 64KB[a]	16/32KB[a]	16KB/32KB/64KB[a]	32KB or 64KB	32KB
L1 Cache Structure	2-way set associative (Inst) 4-way set associative (Data)	2-way set associative (Inst) 4-way set associative (Data)	4-way set associative	4-way set associative (Inst) 4-way set associative (Data)	4-way set associative (Inst) 4-way set associative (Data)	2-way set associative (Inst) 2-way set associative (Data)
L2 Cache Structure	-	8-way set associative	8-way set associative	-	16-way set associative	16-way associative

ARM cache features

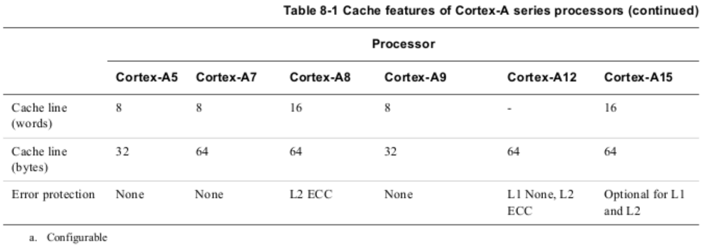


Table 8-1 Cache features of Cortex-A series processors (continued)

	Processor					
	Cortex-A5	Cortex-A7	Cortex-A8	Cortex-A9	Cortex-A12	Cortex-A15
Cache line (words)	8	8	16	8	-	16
Cache line (bytes)	32	64	64	32	64	64
Error protection	None	None	L2 ECC	None	L1 None, L2 ECC	Optional for L1 and L2

a. Configurable

ARM Cortex A7 Structure

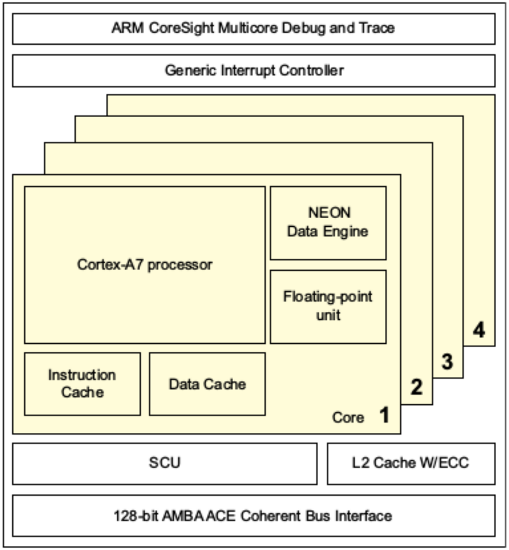


Figure 2-4 Cortex-A7 processor

Example: Cache friendly code

See the background reading material on the web page:
Web aside on blocking in matrix multiplication

Summary: Memory Hierarchy

- In modern architectures the main memory is arranged in a **hierarchy of levels** ("memory hierarchy").
- Levels higher in the hierarchy (close to the processor) have fast access time but small capacity.
- Levels lower in the hierarchy (further from the processor) have slow access time but large capacity.
- Modern systems provide hardware (**caches**) and software (paging; configurable caching policies) support for managing the different levels in the hierarchy.
- The simplest caching policy uses **direct mapping**
- Modern ARM architectures use a more sophisticated **set associative** cache, that reduces "cache thrashing".
- For a programmer it's important to be aware of the impact of **spatial and temporal locality** on the performance of the program.
- Making good use of the cache can reduce runtime by a factor of ca. 3 as in our example of blocked matrix multiplication.