

Topic 4: Processor Fundamentals

Why real CPUs are fast: caches, split memory paths and the limits of multi-core.

Optional stretch material. This goes beyond the 9618 syllabus, so you will not be asked about it directly in the exam. Its purpose is to deepen ideas you already know. **Builds on:** 4.1 CPU architecture. Try the challenge questions on the last pages before you read the answers.

Caches and the principle of locality

A CPU can do an addition in under a nanosecond, but main memory takes around 50–100 ns to respond. Caches bridge the gap by exploiting **locality**: **temporal** (data used recently is likely to be used again) and **spatial** (data near recently used data is likely to be used soon). So memory is moved in **cache lines** (typically 64 bytes), not single bytes.

Level	Typical size	Typical access time
Registers	Under 1 KB	Under 1 ns
L1 cache	32–64 KB per core	About 1 ns
L2 / L3 cache	256 KB – tens of MB	About 3–40 ns
Main memory (DRAM)	GBs	About 50–100 ns
SSD	100s of GB	About 100 000 ns

Average memory access time (AMAT) = hit time + miss rate × miss penalty. With a 1 ns hit time, a 5% miss rate and a 100 ns penalty: $1 + 0.05 \times 100 = 6 \text{ ns}$. Here the miss term (5 ns) dwarfs the hit time (1 ns), so reducing misses is the best way to improve performance.

Why code style matters. Arrays are stored row by row, so the loop below walks along memory and uses every byte of each cache line. Swap the two loops (column by column) and every access jumps a whole row ahead, wasting most of each line: often several times slower for identical logic.

```
FOR Row <- 1 TO 1000
  FOR Col <- 1 TO 1000
    Total <- Total + Grid[Row, Col] // consecutive memory: cache friendly
  NEXT Col
NEXT Row
```

Von Neumann vs Harvard

The von Neumann design shares one memory (and one set of buses) for instructions and data, so the CPU cannot fetch an instruction and read data at the same moment: the **von Neumann bottleneck**. A **Harvard** machine has separate memories and buses for each, allowing both at once; it is common in microcontrollers and DSPs.

Modern PCs use a hybrid (sometimes called *modified Harvard*): separate **L1 instruction and data caches** feeding a single shared main memory. Keeping code and data distinct also supports security rules such as marking data pages as non-executable.

Amdahl's law: why 64 cores is not 64× faster

If a fraction **p** of a program can run in parallel and the rest must run in sequence, the speed-up on **s** processors is:

$$\text{speed-up} = 1 / ((1 - p) + p / s)$$

With $p = 0.8$ on 4 cores: $1 / (0.2 + 0.2) = \mathbf{2.5\times}$. With *infinite* cores: $1 / 0.2 = \mathbf{5\times}$. The sequential 20% sets a hard ceiling, which is why designers fight to shrink the serial part of a program.

Challenge questions

1. An L1 cache has a 1 ns hit time and a 10% miss rate; a miss costs 50 ns. Find the AMAT. A larger cache cuts the miss rate to 2%. What is the new AMAT?
2. A program is 90% parallelisable. What speed-up does Amdahl's law predict on 8 cores? What is the maximum possible speed-up?
3. Explain why summing a 2D array column by column can be much slower than row by row, even though both loops do the same number of additions.

Worked answers

Only look once you have had a proper go at each question.

1. Before: $1 + 0.10 \times 50 = 6 \text{ ns}$.

After: $1 + 0.02 \times 50 = 2 \text{ ns}$, three times faster.

2. $1 / (0.1 + 0.9/8) = 1 / 0.2125 \approx 4.7\times$. With unlimited cores the limit is $1 / 0.1 = 10\times$.

3. Memory holds each row contiguously. Reading along a row uses all 64 bytes of each cache line (spatial locality), so most accesses are cache hits. Reading down a column jumps a full row between accesses, so each access lands in a different line and most of the loaded line is never used. The miss rate (and therefore the AMAT) is much higher.