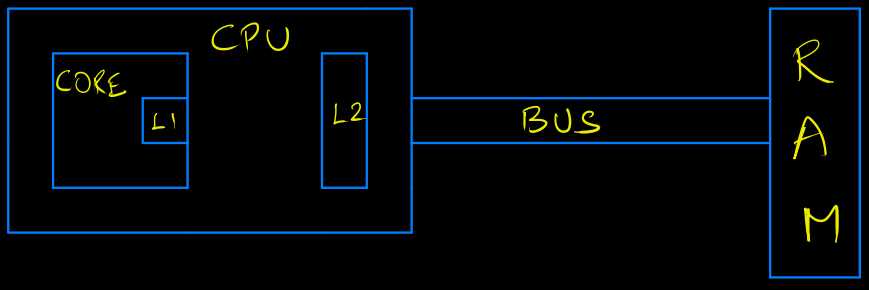
Memory System Performance - Caching

Cache : Collection of memory locations

Resides On-Core, On-chip

What? High speed access (~1ns) (Vs RAM (100 ns))

→ Cannot be large size (Vs RAM)



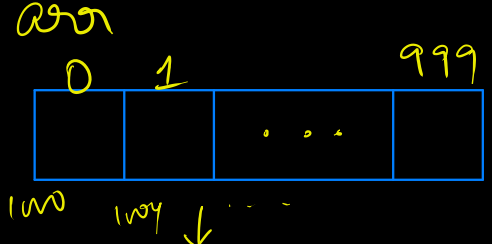
Instruction Cache & Data Cache

Why? → Principle of Locality

→ Sequential Execution 100 101 102

Branch Instructions?

→ Iterative data access



Contiguous Memory locations

```
float arr[1000], Sum = 0.0f;
int i = 0;
// arr values are computed here
for(i = 0; i < 1000; i++)
    Sum += arr[i];
```

⇒ Access of one location is followed by an access of a nearby location ⇒ Spatial locality,
in the nearby future ⇒ Temporal locality

How?

One Cache line = Multiple Memory blocks (e.g. 8 to 16)

Let Memory block = 4 bytes (= bus width)
Let Cache line = 8 blocks
⇒ One cache line = 32 bytes

For single level cache,
When arr[0] is accessed
↳ check if available in Cache

HIT
(Proceed)

MISS
→ Fetch Cache line containing &arr[0] from memory into Cache
→ Execution Stalls
→ But what about the next access?

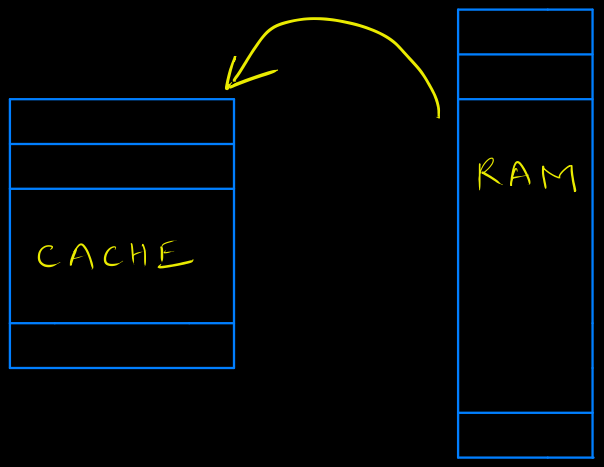
→ Extend the idea to multi-level hierarchy

Issues in Cache Design

I → Writes can cause inconsistency.

- Write-through
- Write-back

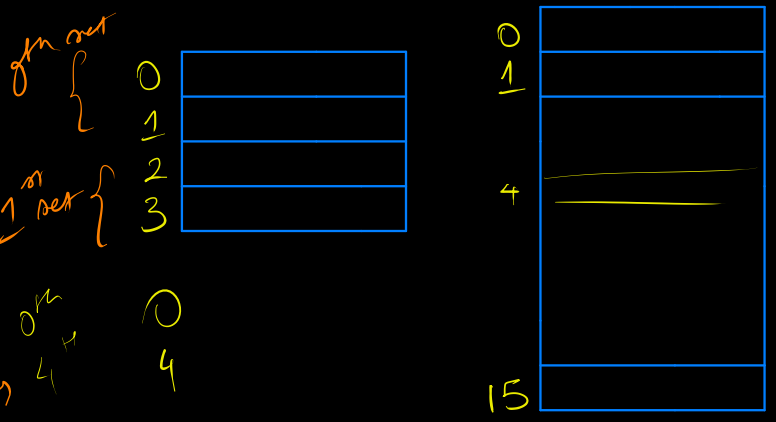
II →



Evict?

Memory blocks to Cache line mapping

Assumption: 1 cache line = 1 Memory block



Direct Mapping:

CL = MB % 4

New line placed at CL

n-way Set Associative Mapping:

NS = No. of CL / n

CL = MB % NS (No. of Sets)

SN

Fully Associative Mapping:

New line can be placed at any location in cache

↓

Cache Replacement Strategy (LRU, LFU, ...)

I. Processor operating at 1 GHz

DRAM latency 100 ns

No. of FMA units = 2

FMA operation completion = 1 cycle

Peak performance = 4 GFLOPS

Cache size = 1 KB

Cache line size = 16 bytes

Cache latency = 1 ns

Data bus speed = 100 MHz (clock = 10 ns)

Data bus width = 1 word (4 Bytes)


```
float a[96], b[96], ans = 0.0f;
int i = 0;
// a & b are set here
for(i = 0; i < 96; i++)
    ans = ans + a[i] * b[i];
```

How many FLOPS can this program achieve?
or what is the peak speed of this algorithm?

CPU accesses $a[0]$ → HIT or MISS ?

↓

Fetches $a[0], .., a[3]$ from RAM to CACHE ⟵ $100\,ns + 10\,ns \times 3 = 130\,ns$

CPU accesses $b[0], .., b[3] \Rightarrow 130\,ns$

$\Rightarrow$ In 260 ns $\Rightarrow$ $a[0], .., a[3]$ & $b[0], .., b[3]$

NO CACHE MISS till $a[4]$ or $b[4]$

2 FMA units : 2 cycles to Complete Computation $= 2\,ns$

(cache access time generally ignored) $N \times (2+1)$

$\Rightarrow$ 8 FLOPs in total 262 ns

[Same pattern Continues for 4, .., 7; ...; 92, .., 95 - 24 times]

Peak Computation rate of the algorithm $= \frac{24 \times 8}{24 \times 262} = \frac{8}{262}$

$= 30.53\,MFLOPS$

% of Peak CPU rating $= \frac{30.53 \times 10^6}{4 \times 10^9} \times 100$

$= 0.76\,\%$ (better than 0.25 %)

II: Processor Speed = 1 GHz (+ 2 FMA in 1 cycle)

DRAM Latency = 100 ns

Cache Size = 32 KB

Cache Latency = 1 ns

Assumption: [Cache line Size = 1 word, Bus width = 1 word]

Multiply 2 matrices A & B, each 32 x 32 (float)

Let C = AB. Will A, B & C fit in cache?

A: $32 \times 32 = 2^{12}$ Bytes = 4 KB

Similarly for B & C. All can fit if there is ideal cache replacement. (NO overwriting)

What is the peak computation rate of the algorithm?

1 word (or float) fetched in 100 ns

Core Compute: `C[i][j] = C[i][j] + A[i][k] * B[k][j];` (FMA)

Let us look into calculation of C[0][0]:

A[0][0]: 100 ns, B[0][0]: 100 ns

After 200 ns, 1 FMA can be completed.

A[0][1]: 100 ns, B[1][0]: 100 ns

After 400 ns, 2 FMA can be completed.

⋮

A[0][31]: 100 ns, B[31][0]: 100 ns

After 6400 ns, 32 FMA can be completed.

Should we consider the same pattern or not?

Let us look into calculation of $C[0][1]$:

Already 0^{th} row of A is in cache!

$\Rightarrow$ After $100\,ns \times 32$ ($B[0][1], \ldots, B[31][1]$), we can compute $C[0][1]$ $\Rightarrow$ 32 FMA can be done.

Similarly for each $C[0][j]$, for $j \leftarrow 0$ to 31

In general, fetch 0^{th} row of A : 3200 ns

fetch 0^{th} col of B : 3200 ns

⋮

fetch 31^{st} col of B : 3200 ns

Total = $3200 + 3200 \times 32$

After that time, we have 0^{th} row of C with 32×32 FMA operations.

Does the pattern change for calculating 1^{st} row of C?

Fetch 1^{st} row of A : 3200 ns

All cols of B are already available in Cache!

$\Rightarrow$ After 3200 ns, 1^{st} row of C can be computed with 32×32 FMA operations.

$\Rightarrow$ In general, for i^{th} row of C, $i \leftarrow 1$ to 31, can each be calculated after 3200 ns each.

$\therefore$ Overall time = $3200 + 3200 \times 32 + 3200 \times 31$

$= 3200 \times 64 = 204800\,ns$

$32 \times 32 \times 100 \times 2$

Total Computations = 32 FMA $\times (32 \times 32)$

$= 32^3$ FMA $= 32^3 \times 2$ FLOPs

Time taken for computation $= \frac{32^3 \times 2 \times 1 \text{ ns}}{4 \text{ (2 FMA per cycle)}} = 16384$ ns

$\therefore$ Fetch Time + Compute time $= 221184$ ns
(cache access time ignored)

$\therefore$ Peak Computation rate of the algorithm $=$

$$\frac{32^3 \times 2}{221184 \times 10^{-9}} = 296.3 \text{ MFLOPS}$$

% of Peak performance $= \frac{296.3 \times 10^6}{4 \times 10^9} \times 100 = 7.4\%$

How would this change if Cache line size = 16 bytes?

Another way to calculate : Cache HIT Ratio

I. 1 DRAM access for every 4 words : CHR $= \frac{3}{4}$

$\Rightarrow$ Average Memory Access time $= \frac{3}{4} \times \underset{(\text{ns})}{1} + \frac{1}{4} \times \underset{(\text{ns})}{100} = 25.75$ ns/word

Dot product : 8 FLOPs after fetching 8 words

$\Rightarrow$ Average Computation per word $= \frac{8}{8} = 1$ FLOPs/word

$\Rightarrow$ Average Computation rate $= \frac{1 \text{ FLOPs/word}}{25.75 \text{ ns/word}} = 38.83$ FLOPS

II. 1 DRAM access for every word : CHR = $\frac{0}{1} = 0$

Average Memory access time = $0 \times \frac{1}{(ns)} + 1 \times \frac{100}{(ns)}$

= 100 ns / word

Matrix Multiply : $2n^3$ FLOPs after fetching $2n^2$ words

Average computation per word = n FLOPs / word = 32 FLOPs / word

∴ Average Computation Rate = $\frac{32}{100}$ = 320 MFLOPS

⇒ Comparing dot product Vs Matrix Multiply,

0.76 % Vs 7.4 %.

(16 byte cache line) (even with 4 byte cache line)

What is increasing an algorithm's Peak Computation Rate?

Data Re-use?