Lec 11 06

Memory System Performance

Topics :-

1) Memory system performance :-
   1) Bandwidth
   2) Latency.
2) Improving Memory latency using cache.
3) Impact of caches.
4) Impact of bandwidth.
5) Approaches for hiding memory latency.

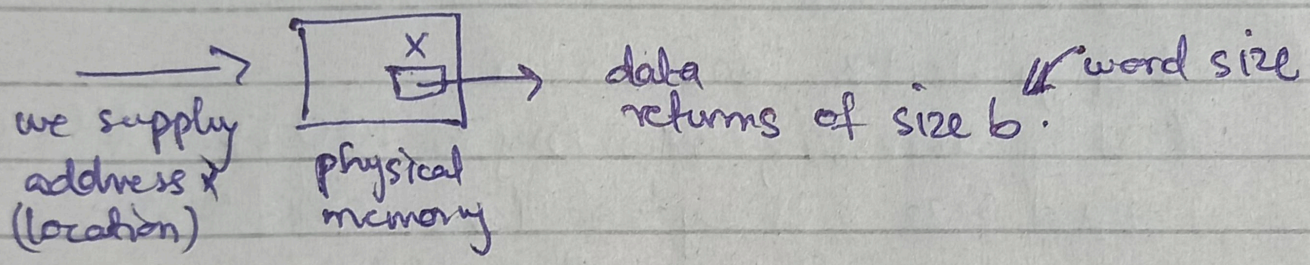
1) Memory System Performance :-

- The performance of a program on a computer depends upon the ability of the memory system to feed data to the processor.
- A memory system takes in a request for a memory word and returns a block of data of size b containing the requested word after l nanoseconds. (l = latency of memory).
- The rate at which data can be pumped from the memory

to the processor determines the bandwidth of the memory.

- latency and bandwidth both play critical roles in determining memory system performance.

EXAMPLE :- EFFECT OF LATENCY ON PERFORMANCE



- latency determines the performance of memory.
- latency is the time it takes from when I give the address and the time it gives me the data.

Example of Matrix Multiply :

$C = A \times B$

64 x 64 matrices.

Each element is 4 bytes.

The simplest code we can write is.

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      Cij += aik * bjk.
```

let see how much time its ~~goo~~ gonna take to perform this code.

let assume a few things:
we have:

1 Giga Hertz processor
Cost of accessing memory = 100 ns or 100 cycles
word size = 4 bytes. (we ask for some data from location it is going to return me a 4 byte value).

Now what are the statements we will execute. First we have to load aik the bkj; then we will mul and add in Cij which we are maintig in a Register

1. Load R1, [R2] ← aik
2. Load R3, R4 ← bjk
3. madd R5, R1, R3 (mul R1 and R3 and adding them to R5)

latency = 100 ns.

How long is it going to take to execute the above code

1st Iteration

line 1 will take 100 ns
line 2 will take 100 ns
line 3 will add some nanosec.

Now we will go back for next iteration. which will consume some nanoseconds but since this will not involve any memory operations, so they are not so expensive so we can ignore them for time being. It is not okay to ignore them but for simplicity we will.

The units to measure performance is Flops. (floating point operations per second)

So we have to calculate how many flops am I getting for this code for single iteration.

So there are 2 operations, mul and add

# of operation = 2.
Time to perform one iteration = 200 ns.

$$FLOPS = \frac{2}{200ns} = \frac{2}{200} \times 10^9 = \frac{2000}{1200} \times 10^6$$

$$= 10 MFLOPS$$

So the processors frequency is 1 Giga Hertz. When you look at 1 GHertz processor you think you are going to perform 1 giga operations every second.

It can actually do that under certain ~~contain~~ conditions, not when you have to go to the memory everytime to fetch data

If your data is only in registers and you are always doing your computation w/ data in registers than you can actually achieve that.

This example highlights the need for effective memory system performance in getting high computation rates.

Improving Memory latency using caches :-

- Cache acts as a low latency high bandwidth storage.

- The data needed by the processor is first fetched into the cache.

- If a piece of data is repeated used, the latency of this memory system can be reduced by cache.

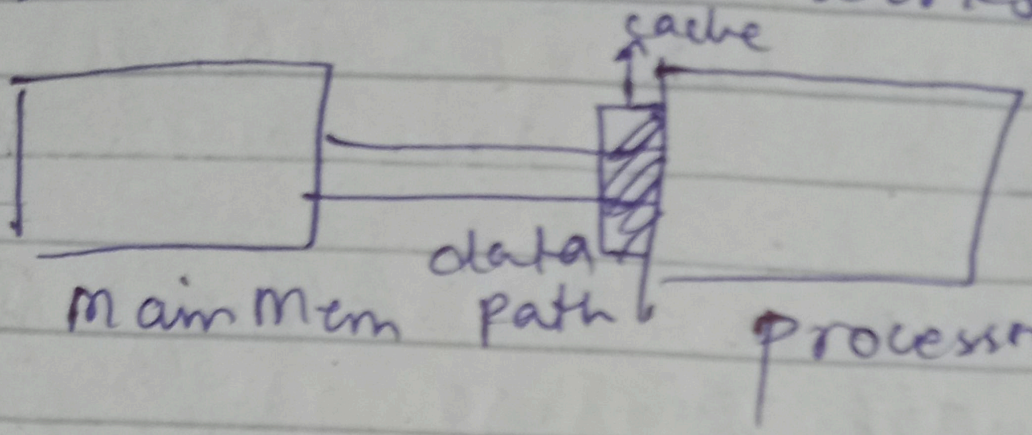
- The fraction of data references satisfied by the cache is called cache hit ratio.

Example -

- The gap b/w the rate at which the processor works and the time it takes to fetch data from the memory

- To solve this issue we use cache which is a smaller but faster memory that sits b/w the processor and main memory.

So, how does a cache works.



- If data is in cache. get from cache else get data from main memory and store in cache.

latency to MM = 100 ns
latency to cache = 1 ns (assume)

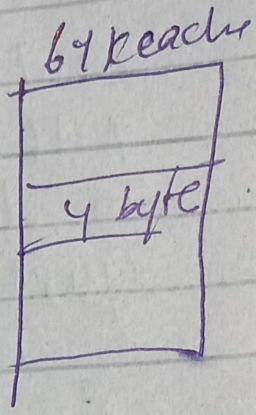
Does cache always give to benefit?
No, it depends on the code if we are accessing data randomly than no, if in a structured way then yes.

lets take the same example of matrices multiplication

MATRIX MULTIPLY :-

$C = A \times B$ 64 x 64 matrix
each element 4 byte

cache size = 64 K



Total size of Mat A (bytes)
= 64 x 64 x 4
= 16384 = 16 K

Size of Mat B = 16 K
" " " C = 16 K.

Total size of all matrices = 48K

So all 48K fits in cache.

Let us assume that the cache gives me a latency of 1 ns or 1 cycle.

latency of cache = 1 ns.
latency of MM = 100 ns

Now what happens is that the first time I access an element it is going to have to go ~~through~~ to the memory and get it which is going to take 100 cycles but every other time

2 access the data it is going to come from cache

Let's just load all the data into the cache first. Although that is not the way to do it but let us assume just for a simple ~~analysis~~ analysis

So we load all Matrices into the cache.

load matrices into cache = 64 × 64 × 2 × 100 ns

= 800 ns.

Now we start executing the code.

load R1, R2
load R3, R4
madd R5, R1, R3

It takes 1 ns.

Suppose it takes 20 cycles to execute 1 iteration.

Total number of iteration-

$n^3 = 64^3 = 256K$

Total time = $256K \times 20ns + 800\mu s$

# of ops = 2×64^3

$$FLOPS = \frac{\#\ of\ ops}{Time} = \frac{2 \times 64^3}{20 \times 64^3 + 800\mu s}$$

$$= \frac{512K}{(256K \times 20)ns + 800\mu s}$$

$$= \frac{512K}{5120\mu s + 800\mu s}$$

$$= \frac{512K}{5920\mu s}$$

$$= 90\ MFlops-$$

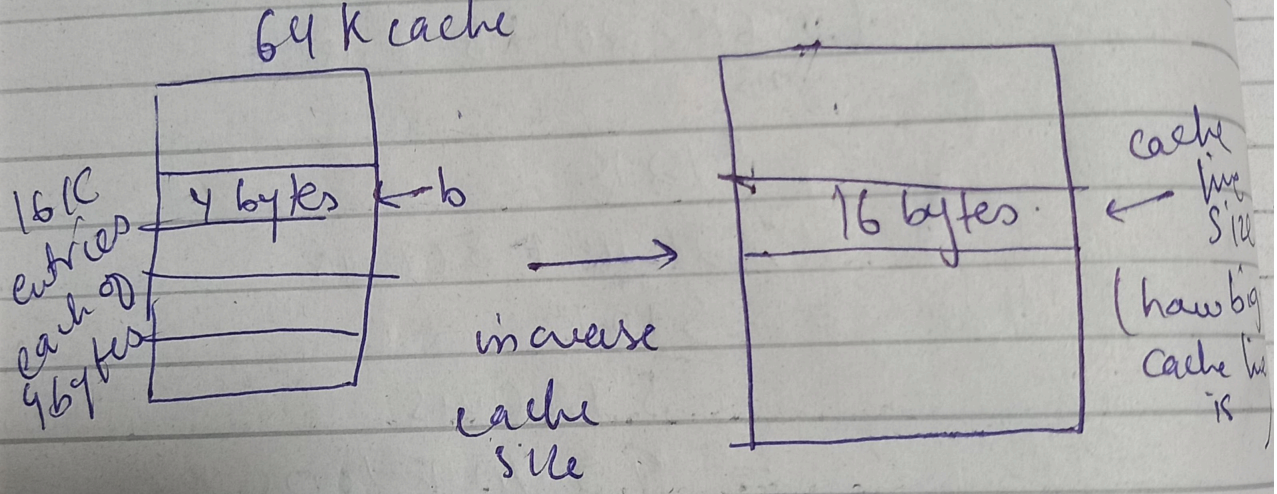
W/o cache = 10MFLOPS
w cache = 90m Flops.

The improvement in performance resulting from the presence of the cache is based on the assumption that there is repealed reference to the same data. This notion of repealed refernce of data item in a small widow is called temporal locality.

Impact of Memory Bandwidth

- Memory Bandwidth is the rate at which data can be moved b/w the processor and memory

- One simple way of increasing the memory bandwidth is to increase the block size.



Example:-

Dot Product of 2 matrices:

repeat
load R_1, [R_2]
load R_3, [R_4]
madd R_5, R_1, R_3

Lets look at the scenario where I have 4 byte cache line size

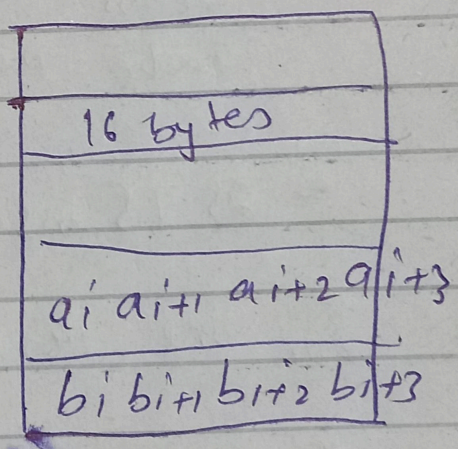
Note: There is no cache reuse because it is dot product not matrix multiply so everytime I get data it is going to have to come from memory.

load R_1, $[R_2]$ → 100ns
load R_3, $[R_4]$ → 100ns
madd R_5, R_1 R_3

$$FLOPS = \frac{2}{200\,ns} = 10\,MFLOPS$$

16 byte cache lines.

load R_1, $[R_2]$ → 100ns
load R_3, $[R_4]$ → 10ns.
⋮



Next Iteration

load R1.]
load R3.] 2ns
⋮
load R_5]
load R_3] 2ns
⋮

Time ≃ 200ns.
operations performed = 8

$$FLOPS = \frac{8}{200\,ns} = 40MF$$

cache-hit ratio: % of memory accesses found in the cache/served by the cache.

Cache hit ratio :- 75 %
(because 1 iteration I had to do from the memory, all remaining from cache)

avg. memory access time =

$$\underbrace{75}_{CHR} \uparrow \underbrace{1n}_{\text{Time it takes to access cache}} + \underbrace{25}_{\text{time it has to go to main mem}} \times 100\ ns.$$

$= 25.75\ ns.$

For one iteration :-

# ops = 2
time = $51.5\ ns \cong 50\ ns.$

$$FLOPS = \frac{2}{50\ ns} = \frac{8}{200} = 40\ MFLOPS$$