

Multi-level Caches

- Primary cache (Level-1) attached to processor (small but fast)
- Level-2 cache services misses from primary cache
 - larger, slower but faster than main memory
- Main memory services Level-2 cache
- Some high end machines use Level-3 cache

Multi-level caches; now, with respect to single level caches, we also we have said before that we have multiple cache hierarchies. Now we will talk about them here. The primary cache or level one cache in multi-level caches is attached to the processor; it is small but fast. Added to that we have a level 2 cache which services misses from the primary cache, it is typically larger in size, but also slower, but slower than the primary cache; however, being much faster than the main memory. So, the main memory then services L2 cache.

So, what hierarchy do I have? I have the processor, then from the processor I have a small primary cache, typically these have these are separate data and instruction caches, then a basically I have a combined much bigger in size L2 cache, this will be these are L1 these both are L1 the L2 cache, and this is then attached to the main memory; which is much bigger. Now, in more in a more high end machines we also have 3 levels of cache. Typically, L1 or L2 are on chip, sometimes L3 cache is off chip in typically L3 caches are off chip.

(Refer Slide Time: 29:08)

Multi-level Caches - Example

- **Given:**

- CPU base CPI = 1 (*all references hit primary cache*), clock rate = 4GHz
- Miss rate/instruction = 2%
- Main memory access time = 100ns

- **With primary cache**

- Miss penalty =
- Effective CPI =

Now, we will take an example on the use of multi-level caches. Particularly, we will see how multi-level caches are able to reduce miss penalties. Let us consider a CPU with a base CPI of one when all references hit the primary cache. So, cycles per instruction is one cycle cycles per instruction is 1. So, one cycle instruction execution when all references hit the primary cache, the clock rate is 4 gigahertz. Miss rate per instruction is 2 percent. And so, 2 percent of all the instructions miss the primary cache.

And I have the main memory access time so, time to go access the main memory and get data is 100 nanoseconds. And first we will consider the case when I have just one primary cache. So, what is the length of the clock cycle? I have a clock rate of 4 gigahertz. So, the length of a clock cycle is $1 / 4 \text{ gigahertz} = 0.25 \text{ nanoseconds}$. So, what is the miss penalty? So, the miss penalty will be given by miss for the penalty that for one for one miss is 100 nanoseconds.

(Refer Slide Time: 30:35)

Multi-level Caches - Example

- **Given:**

- CPU base CPI = 1, clock rate = 4GHz
- Miss rate/instruction = 2%
- Main memory access time = 100ns

- **With primary cache**

- Miss penalty = $100\text{ns}/0.25\text{ns} = 400$ cycles
- Effective CPI =

So, in terms of clock cycles, it will be $100 / 0.25$ nanoseconds; which is equal to 400 clock cycles. So, 100 nanoseconds to access main memory is 0.25 nanoseconds for one cycle. So, 100 nanoseconds divided by 0.25 equal to 400 cycles. So, when just see that when I hit the primary cache, if I can execute one instruction in one cycle. Now, if I have to go to the main memory, then I incur 400 cycles of penalty. The good thing is that, the good thing is that only 2 percent of these instructions miss the cache right. So, what will be the effective CPI?

The effective CPI will be 1 when I have a cache hit plus so, I tried to I went to the cache, I saw that there is a cache miss. This will happen in only 2 percent of the cases or 0.02. So, for all these 0.02 or 2 percent cases I will incur 400 cycles of miss penalty. So, what will be the effective CPI? $1 + 0.02 \times 400 = 9$ cycles. Now let us assume that with this along with this cache we have added a L2 cache ok. The L2 cache has an access time of 5 nanoseconds ok.

To get to the L2 cache I require to get to the L2 cache and get the data I require 5 nanoseconds. Now, the global miss rate to main memory is 0.5 percent. So, the number of the percentage of cases for which I miss the primary cache as well as the secondary cache means the L2 cache is only 0.5 percent. Now the miss penalty with the L2 cache is given by how many how much? So, 5 nano in the miss penalty with the L2 cache hit miss penalty with L2.

(Refer Slide Time: 33:02)

Multi-level Caches - Example

- **Add L-2 cache**
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- **Primary miss with L-2 hit**
 - Penalty =
- **Primary miss with L-2 miss**
 - Extra penalty =
- **CPI =**
- **Performance ratio =**

Computer Organization and Architecture

83

So, primary cache has missed, and in the L2 cache I have a hit. So, what will be the penalty? The penalty will be 5 nanoseconds divided by 0.25 nanoseconds. So, 5 nanoseconds / 0.25 nanoseconds = 20 cycles. So, therefore, primary miss with L2 hit. So, when I have primary miss with L2 hit, I will have it will be 20 cycles.

(Refer Slide Time: 33:30)

Multi-level Caches - Example

- **Add L-2 cache**
 - Access time = 5ns
 - Global miss rate to main memory = 0.5%
- **Primary miss with L-2 hit**
 - Penalty = $5\text{ns}/0.25\text{ns} = 20$ cycles
- **Primary miss with L-2 miss**
 - Extra penalty = 420 cycles
- **CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$**
- **Performance ratio = $9/3.4 = 2.6$**
 - With L-2 cache, processor faster by 2.6 times

Computer Organization and Architecture

85

When I have a primary miss with L2 miss what will be the extra penalty? The extra penalty will be 420 cycles. Why? Because 400 cycles will be to get to the main memory and 20 cycles to get to the L2 cache; so, the total penalty when there is a primary miss with L2 miss is 420 cycles. Now what will be the effective CPI? The effective CPI will be 1 when there is a hit in the primary cache, plus 0.02 into 20 because 2 percent of my instructions miss the primary cache. So, 2 percent of 20 2 percent into 20 this many times I will have. And L2 cache and L2 cache access, out of these so, 2 percent of the times I will go to the L2 cache ok. Because I have a primary miss, 2 percent of the times I will have a primary miss, I will go to the L2 cache.

For that I will have a overhead of this. And now out of this, accesses to the L2 cache 0.5 percent will result in misses on the L2 cache. And for each of those times I will have to go to the main memory. So, 0.5 percent or 0.005 into 400 cycles will be the effective cost. So, the total effective CPI becomes 1 plus 2 percent of the times I missed the primary and go to the L2, and 0.005 times I miss the L2 and go to the main memory. So, the effective CPI will be 3.4. Now, the performance ratio without and with the secondary cache will therefore be 2.6. So, with the L2 cache, the processor is faster by 2.6 times. So, when I add the L2 cache my effective CPI reduces from 9 to 3.4 and therefore, my processor is 2.6 times faster.

(Refer Slide Time: 35:53)

Multi-level Cache Design Issues

- **Primary cache**
 - Focus on minimal hit time
- **L-2 cache**
 - Focus on low miss rate to avoid main memory access
 - Hit time has overall less impact
- **Result**
 - L-1 cache is usually smaller than a single-level optimally designed cache
 - L-1 block size is smaller than L-2 block size

Now, multi-level cache design issues. So, the focus of the primary cache is to minimize the hit time. Because I expect that each time I go for executing an instruction I will go to the cache and I have to fetch. So, I will try to minimize the amount of time I require to access the primary cache and get data. So, what is the focus of the L2 cache? The focus is on low miss rate; I want to avoid going to the main memory access. Why? Because we saw that only one cycle was for hit into the primary cache. Around 5 times was the hit with for a with respect to the primary cache with around 5 times was the cost of going to the L2 cache. And was 100 times was the cost of going to the main memory when I when I if I have to go to the main memory.

So, hit time has overall less impact for the L2 cache, because why because, I go to the L2 cache when only when I have a miss on the primary cache. The result is that L1 cache is usually smaller, because I want to access it fast. And to get that I will have a smaller than a single level optimal optimally designed cache. So, if I have a single level optimally designed cache, I don't have multiple caches with respect to that my L1 cache will typically be much smaller. Why because, I have the support of a big L2 cache. I want the L1 cache to be small and make the access times much lower much quicker.

For the L1 cache block size will also be smaller compared to the L2 block size. Why this is also, that if the block sizes are larger my transfer times for the block becomes higher. So, we lower the block size lower at the transfer time. So, I will prefer a lower block size for the L1 cache. Now, also because my L1 cache size is smaller so, block size will also be smaller in turn.

(Refer Slide Time: 38:03)

Advanced CPUs

- **Out-of-order CPU can execute instructions during cache miss**
 - Dependent instructions wait in reservation station
 - Independent instructions continue execution
- **Effect of miss depends on program data flow**
 - Hard to analyze
 - Simulate the whole system

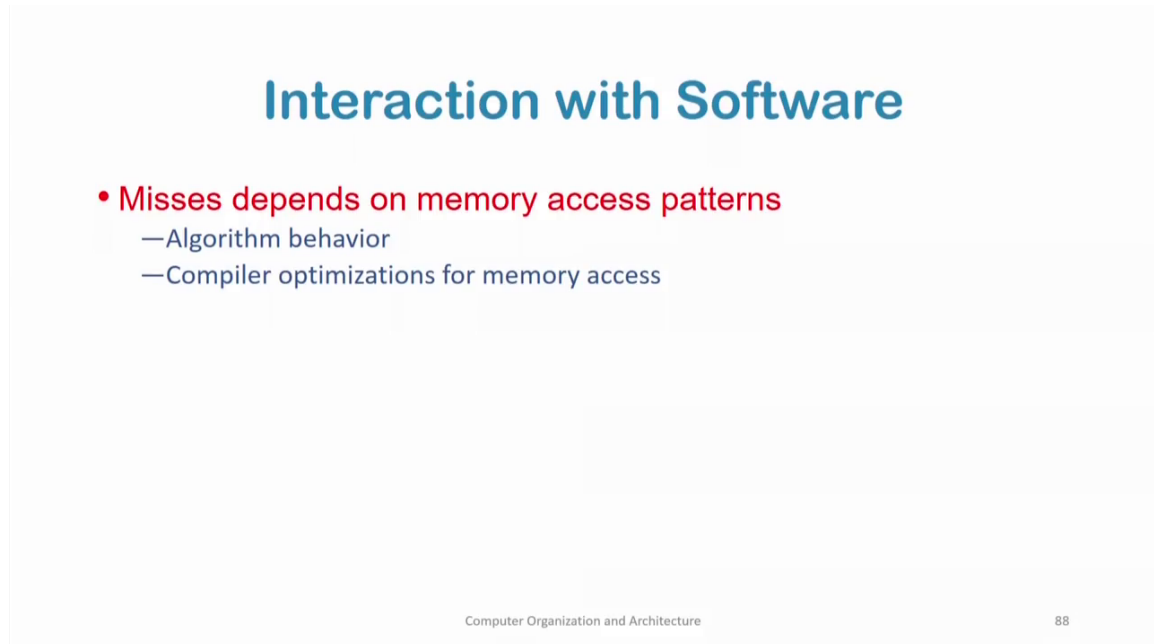
For more advanced CPUs; for example, out of order CPUs currently whatever we were studying where what in order CPU. So, whatever instructions are whatever machine instructions are placed to it they execute in order sequentially. So, in out of order CPUs the in the CPU has the ability to find independent instructions and execute out of order. So, what happens? Dependent instructions wait in reservation. So, I have an instruction subsequent to that I have other instructions which depend on my current instruction.

But after that, I also have a few instructions with my which are independent of my current instruction. So, those independent instructions can continue, and I will have reduction in cache misses. So, in cache misses if I have to wait for cache misses my dependent instructions will wait. So, an out of order CPU can execute instructions during cache miss how? Dependent instructions will wait on the reservation station, and independent instructions will continue their execution. Effect of miss depends on program data flow as well. Now how the program data means what instructions are accessed after what? So, each program has control and data control flow graph. Now depending on that the data depending on the control the data will flow that the flow of data through the program is going to vary.

Depending on that the sequence of instructions being executed is going to vary. And cache misses in that case becomes harder to analyze. And the way to do that is to simulate the whole system for more time constraint, more performance centric systems where the demand on performance predictability on performance becomes very critical,

we need to simulate the whole system sometimes to get a measure of what will be the penalty of misses corresponding to the execution of a program. Misses also depend on the access memory access patterns, the algorithm behavior here so, and the compiler optimizations for memory access.

(Refer Slide Time: 40:26)



The slide features a light blue background with a white rectangular area in the center. At the top of this area, the title "Interaction with Software" is written in a bold, blue, sans-serif font. Below the title, there is a bulleted list. The first bullet point is "Misses depends on memory access patterns" in red. It has two sub-bullets: "—Algorithm behavior" and "—Compiler optimizations for memory access", both in blue. At the bottom of the white area, the text "Computer Organization and Architecture" is on the left and "88" is on the right, both in a small, grey font.

Interaction with Software

- Misses depends on memory access patterns
 - Algorithm behavior
 - Compiler optimizations for memory access

Computer Organization and Architecture 88

So, today's compilers do different types of optimizations to improve to improve cache hit rates, and reduce the number of misses. We will take an example to show how compiler optimizations can be used to improve cache hit rates.

(Refer Slide Time: 40:50)

Problem 1

- Consider a 4-way set-associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. Determine the number of bits in the *tag*, *index* and *offset* fields.



Now, we will take 2 problems, the first of which is as follows. Consider a 4 way set associative cache consisting of 128 lines with a line size of 64 words, the CPU generates 20-bit address of a word in main memory. Determine the number of bits in the tag index and offset fields. So, I have a 4 way set associative cache consisting of 128 lines with a line size of 64 words.

The CPU with so, with the line size of 64 words; so, first thing that comes to our mind is that, if the line size is 64 words, the block size in memory is also 64 words. That means, the number of if I have a word addressable memory, the number of words the number of bits required to access a word becomes 2^6 , because $64 = 2^6$.

(Refer Slide Time: 41:57)

Problem 1

- Consider a 4-way set-associative cache consisting of 128 lines with a line size of 64 words. The CPU generates a 20-bit address of a word in main memory. Determine the number of bits in the *tag*, *index* and *offset* fields.
- Line size of cache = block size of main memory = 64 words = 2^6 words
- So, no. of offset bits = 6



So, line size of cache = block size of main memory = 64 words or 2^6 words. So, the number of offset bits in a in the main memory address becomes 6 we get the first result for the offset. Then so the total number of bits for the tag and index fields within the 20-bit address. So, I have a memory address of 20 bits, I have already said that 6 bits are used for my offset bits. So, the number of bits used for my tag plus index field will be $20 - 6 = 14$ bits.

Now, the number of sets in cache now will become cache size divided by the number of ways. The number of sets in cache is cache size by number of ways. So, we said that the that my cache contains 128 lines. So, catch size is 128, I have a 4 way set associative cache so, number of ways is 4. So, the number of sets in cache is $128 / 4$ or $2^7 / 2^2 = 2^5$. Or therefore, I have 32 cache, 32 line 32 sets in the cache I have 32 sets in the cache. So, the number of bits required to index the cache will be given by 5. So, I have 5 bits in my index field of the main memory. So, the number of tag bits is given by $20 - 6 + 5 = 9$, 6 bits for my offset 5 bits for my index so, $20 - 6 + 5$ so, 9 bits in the tag field.

(Refer Slide Time: 43:44)

Problem 2

- A CPU has a 32 KB direct-mapped cache with 128-byte block size. Suppose A is a two dimensional array of size 512 x 512 with elements that occupy 8-bytes each. Consider the following two C code segments, P1 and P2,

```
P1 : for (i=0;i<512;i++) {
        for (j=0;j<512;j++) {
            x+=A[i][j];
        }
    }
```

```
P2 : for (i=0;i<512;i++) {
        for (j=0;j<512;j++) {
            x+=A[j][i];
        }
    }
```

- P1 and P2 are executed independently with the same initial state, namely, the array A is not in the cache and i, j, x are in registers. Let the number of cache misses experienced by P1 be $M1$ and that for P2 be $M2$.
- Find the values of $M1$ and $M2$

Computer Organization and Architecture

94

Now, we take another example. A CPU has a 32-bit direct mapped cache with 128-byte line with 128-byte block size. So, block size is 128 byte so, line size is also 128 byte. Suppose, A is a 2 dimensional array of size 512×512 with elements that occupy 8 bytes each. So, consider the following 2 code segments P1 and P2. So, going before going to the code segment just let me reiterate, I have a 32 bit 32 kilobyte direct mapped cache. So, 32 kilobyte direct mapped cache will mean. So, what will be the size of the cache? 32×2^{10} .

So, 2^5 into 2^{10} ; so, 2^{15} bytes is the size of the cache. And I have a block size of 128 bytes. So, 2^7 is my block size. And I have a 2 dimensional array of size 512×512 of 8 bytes each. So, what will be the total size of my array? $2^9 \times 2^9 \times 2^3$; $9 + 9 + 3$ so, 2^{21} bytes is the size of my data. So, the size of the array is 2^{21} bytes. Now we come to these 2 code fragments consider the following 2 code segments P1 and P2. So, in P1 what do I have? I have 2 nested arrays.

So, the first goes from $i = 0$ to $i = 512$. The second goes from $j = 0$ to $j < 512$ $j++$. And I access $x = 2$ so; I just sum up all the elements in the array here. So, $x = x + A[i][j]$. In P2 I have the same i and the same j , the only difference being that I do $x = x + A[j][i]$; that means, that this for this array I access the array row wise one row at a time ok.

So, $i = 0$, for $i = 0$ I first go for with all j 's. Then $i = 1$ I go with all j 's from 0 to 512. So, I access one row at a time. In this case, I go column wise, why? Because I do $x = x +$

$A[j][i]$. So, first I access $A[0][0]$ then I access then I access a first I access $A[1][0]$, sorry, $A[0][0]$ then I access $A[1][0]$, then I access $A[2][0]$ finally, I access $A[512][0]$, and then I go back to $A[0][1]$, $A[1][1]$, $A[2][1]$, $A[3][1]$ up to $A[512][1]$. Then I go to $A[0][2]$, $A[1][2]$ $A[3][2]$ up to $A[512][2]$, then I go to $A[0][3]$, $A[1][3]$ up to $A[512][3]$ and so on.

So, I access this array column wise. Now P1 and P2 are executed independently with the same initial state. Namely, the array A is not in the cache and i, j, x are in registers. Let the number of cache misses x cache misses experienced by P1 be $M1$ and that for P2 be $M2$. So, find the values of so I want to find the cache misses for P1 and P2, when I execute P1 and when I execute P2.

(Refer Slide Time: 48:19)

Problem 2

- 32 KB direct-mapped cache with 128-byte block size; $A = 512 \times 512$, 8 B each
- P1 accesses array A row-wise
- Block size = 128 B
- No. of data elements in each block = $128 / 8 = 16$

Computer Organization and Architecture

95

So, how do I find that out? So, I have A 32 kb direct mapped cache with 128-byte block size, the size of the array is 512×512 , 8 bytes each. P1 accesses the array row wise as we just discussed. So, block size is 128 bytes. So, the number of data elements in each block becomes $128 / 8 = 16$.

So, my one block or one line contains 128 bytes. And I said that each data element is 8 bytes. So, the number of distinct data elements in each block is given by $128 / 8 = 16$. Then the number of blocks required to store all the data will be given by; so, what is the total number of elements I have? 512×512 ok.

So, the number of blocks and I said that so, these are how many elements? The number of elements are 512×512 . And in each block I can store 16 elements. So, the number of blocks that will be required to store all my data is given by 512×512 by 16; that is, 2^9 into $2^9 / 2^4$. So, $2^{18} / 2^4$ which is 2^{14} and $2^{14} = 16384$ blocks. Now I am accessing for P1, for P1 I am accessing the elements row wise. So, what happens? When I accessed $A[0][0]$, I incurred a miss, and I brought that block of 16, I brought the block of 16 bytes to the sorry, 16 elements to the main memory. I brought in the block of 16 elements.

Now this 16 elements what does it contain? $A[0][0]$ $[0][1]$, $[0][2]$ $[0][3]$ up to $[0][15]$ ok; now if we see if you if you just go and look back at the at the code for P1, we see that I first access $A[0][0]$, then I acts as 01, I acts as 02, I acts as 03 up to 0 15. So, there will be a miss for 00, but then for from 01 to 0 15 there will be no misses because, all these data has been brought into cache during the first miss on $A[0][0]$. So, in the number of misses will be equal to the number of blocks as accessed. So, all blocks are required to be accessed at least once, and there will be one miss per block. So, therefore, the number of misses will be 16384 now; however, P2 accesses the array column wise. As I said I access $A[0][0]$ I incur a miss, then I have what I have brought into cache is $A[0][1]$ to $A[0][15]$.

But what do I access next? I access $A[j][i]$; that means, I am accessing $A[1][0]$ next. $A[1][0]$ is again not in cache and therefore, there will be again another miss in the cache. So, basically if we see all the accesses to the cache will result in misses. So, M2 will be will be the number of misses will be all the accesses, all the accesses will result in misses and it will be given by this value; which is which is 512×512 ; that means, $2^9 \times 2^9$ or 2^{18} . So, this is equal to 2^{18} . This will be the total number of misses. So, M1 is 2^{14} and M2 is 2^{18} .

With this we come to the end of this lecture.