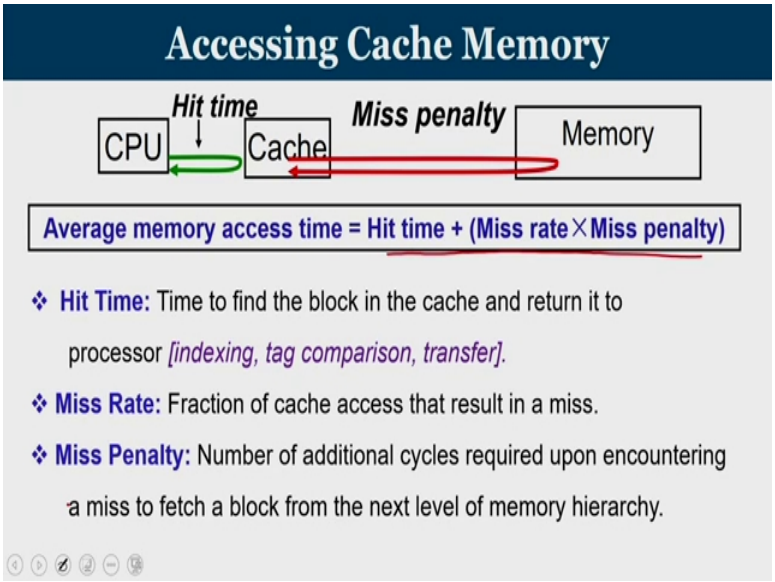
Advanced Computer Architecture
Dr. John Jose
Department of Computer Science & Engineering
Indian Institute of Technology Guwahati

Lecture No. – 15
Optimization Techniques in Cache Memory

Hello everybody, welcome to lecture number 15. We are continuing our discussions on cache memories. In today's lecture, we are planning to find out the techniques or mechanisms by which we can improve the performance of cache memory access and the lecture is titled as optimization techniques in cache memory.

(Refer Slide Time: 00:58)



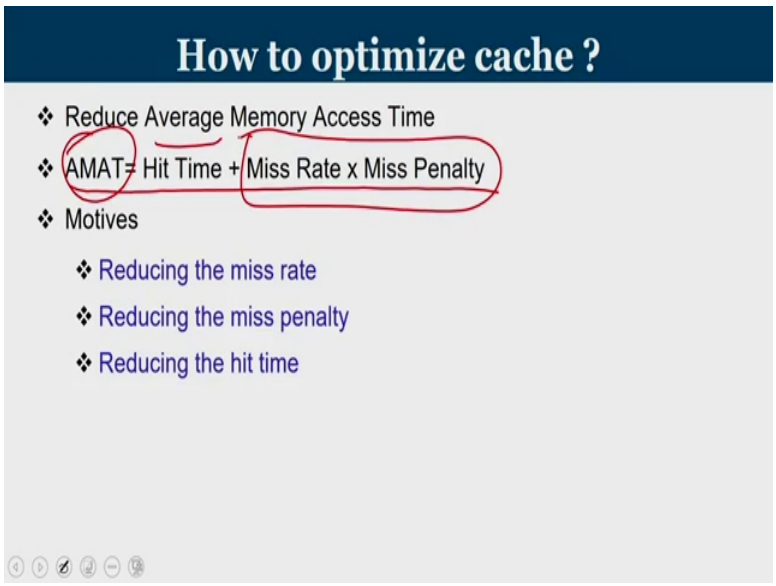
So, we know that accessing in cache memory, the CPU first tries to access the cache and that is what is known as hit time. So, CPU is going to refer to cache, it is known as hit time, and if it is a miss, than the extra time needed all the way going into main memory and bringing is called miss penalty. Look into average memory access time is defined as**,** so in every access, whether it is a hit or a miss, this green component is there always.

So, this green component is there in all the cases whether it is a hit or miss and this red component is there only when there is a miss. So, if it is a hit, it is purely this circle. If it is a miss, it takes half of the green, go all the way to main memory, come back, and then complete the remaining portion of the green. So, average memory access time is defined as hit time, the time to access the cache and return the corresponding word, that is called hit time, plus miss rate into miss penalty.

Average mem access time= Hit time + Miss Rate* Miss Penalty

Now, we will try to see what is hit time, time to find the block in the cache and return it to processor. It involves time to index, tag comparison, and transfer of data. Miss rate is a fraction of cache access that has resulted in a miss, and miss penalty is the number of additional cycles required upon encountering a miss to fetch a block from the next level of memory hierarchy.

(Refer Slide Time: 02:26)



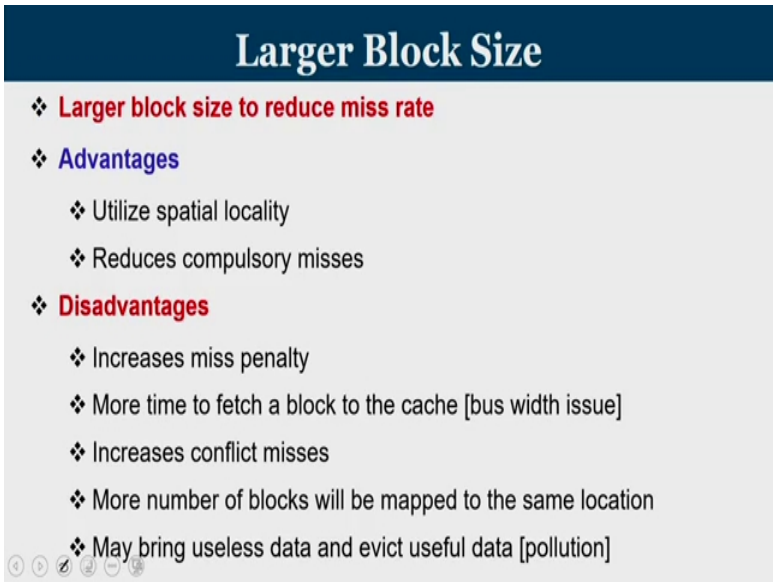
Now, to reduce average memory access time, you know, this is the equation given by AMAT given by hit time plus miss rate into miss penalty. Now, how can you reduce average memory access time, because you always want memory access to take less amount of time. To reduce average memory access time, either reduce hit time or you reduce miss rate or you can reduce miss penalty.

So, these are the three ways by which we are going to work. But this is a slightly complex equation. If you are bringing up some techniques which will reduce miss penalty, then automatically miss rate will go high, or if you do something on hit time, then sometimes miss rate and miss penalty will get impacted. So, these three terminologies are very closely knitted. We cannot play around with only one technique. All are equally getting affected.

So, in this lecture, our whole idea is how to reduce average memory access time and the technique that we use is very simple. Come up with techniques that can reduce hit time, come up with techniques that can reduce miss rate and techniques that can reduce miss penalty. But we have to carefully do it in such a way that impacting one, trying to tweak one

of the parameter will be counterproductive in some other aspects. So, altogether, we have to get average memory access time less.

(Refer Slide Time: 03:45)



First let us see very simple techniques. The first technique is called larger block size to reduce miss rate. So, when I am going to increase my block size, let us say, previously my block size was 16 words. So, for every miss I go and bring 16 words, that will reduce compulsory miss for the next 15 words.

If the block size would have been 32 words for every miss I am going to bring larger 32 words. That is the way it is working. So, when you increase the block size, you are trying to increase the spatial locality that will reduce compulsory misses. But what are the disadvantage, it is increasing miss penalty. How miss penalty is increased?

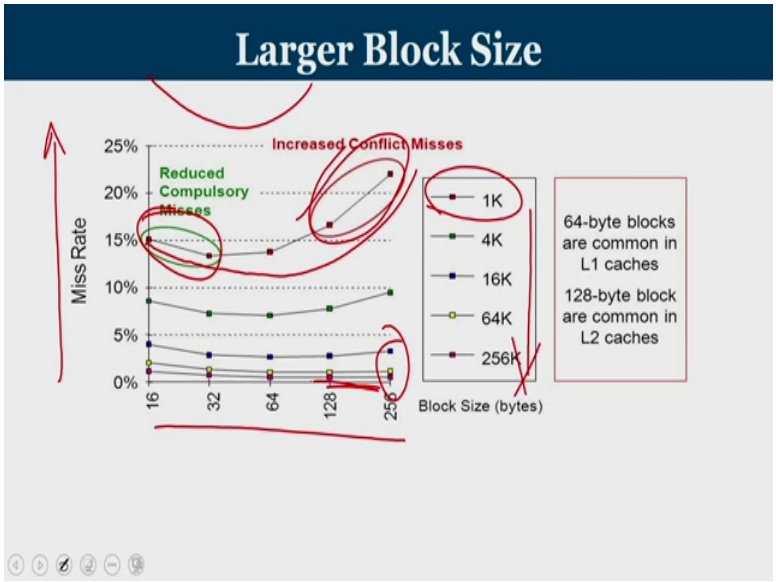
Previously, how to bring only 16 words, I can bring only one word at a time, so it will take 16 consequently bus cycles to bring 16 words. Now, when the block size is doubled to 32 words it takes more time to fill up a cache block. So, my miss penalty will increase. You require more time to fetch a block to the cache. It is basically a bus width issue.

And for a given cache size, if you increase the block size, then the number of set will come down, because total cache size is **(()) (4:50)**, total cache size is nothing but number of sets into the block size. If you multiply that in the case of a direct mapped cache. So, one parameter, if you increase block size, automatically set number will come down which can

increase your conflict miss. So, increasing block size is a good idea but there are two aspects to it.

When you increase block size compulsory miss will reduce but conflict miss will go high, beyond the point conflict miss will be very high than the saving that you get in compulsory miss. That is a takeaway. So, when more number of blocks will be mapped to same location conflict miss is going to increase, and sometimes when you increase block size to very large quantity it can bring useless data and evict useful data and that is what is known as pollution.

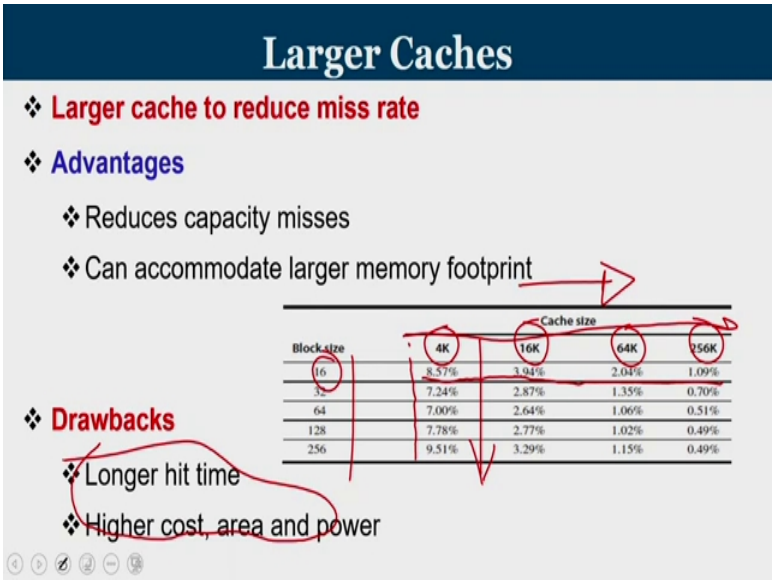
(Refer Slide Time: 05:40)



So, look at this graph. It is a data collected for some of the application. On the x-axis we have block size in terms of bytes, 16 bytes, 32 bytes, 64 bytes, 128 bytes, and on the y-axis you have the miss rate ranging from 0% all the way to 25%. Let us say I am considering a 1K cache, that what we can see, the misses come down and then it goes high. So, you will be getting reduced compulsory miss as the block size increases and conflict miss is going to be very high when you go for larger block size.

The same trend, it dips down then goes up, that has been shown in all different size of cache. We have compared with different size of cache and then try to vary the block size and see how the miss rate is performing. The trend is more or less similar. Now, whatever big block size you take, still you have some category of misses that is coming and that is a basic conflict miss that you get.

(Refer Slide Time: 06:35)



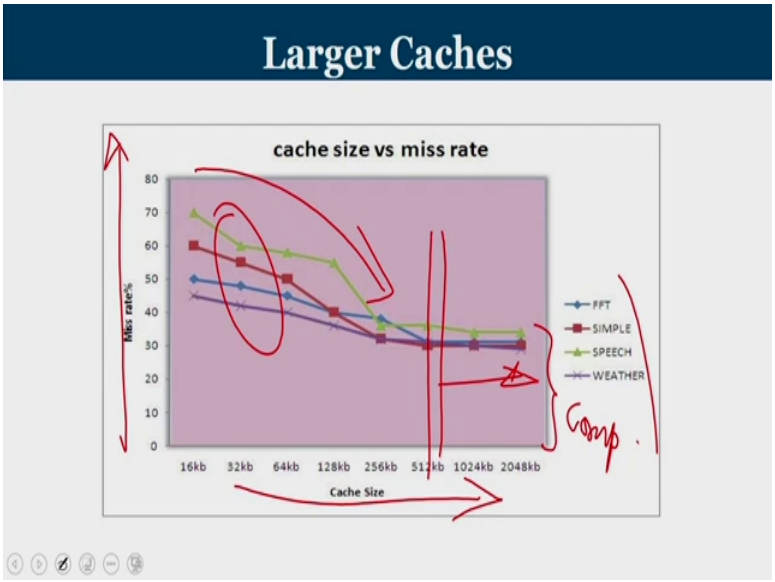
So, first the technique was employing larger block size. The next technique to reduce miss rate is go for larger cache. So, when you go for larger cache you can reduce capacity miss because we can accommodate more memory footprint at the drawbacks of longer hit time, higher cost, area, and power.

So, this is the table that will show us for one of the representative application. So, on row we can see the different cache size, 4K cache, 16K cache, 64K cache, and 256K cache have been used. Let us say we fix up one block size. For an application the miss rate goes from 8 to 3 to 2 to 1. So, as we move towards the right side, as we employ larger cache, miss rate is coming down.

For a given cache if you go down, that is actually block size variation, the misses come down, 8.5, 7.27, then it goes high 7.78 and 9.51. So, we have seen the properties. So, larger cache will help you in some way at the expense of longer hit time. Why it is longer hit time? The bigger the cache, the more number of set index it has, it will take more time to decode and reach the corresponding set.

Of course, naturally it is going to take more of floor area, higher cost, area, and power.

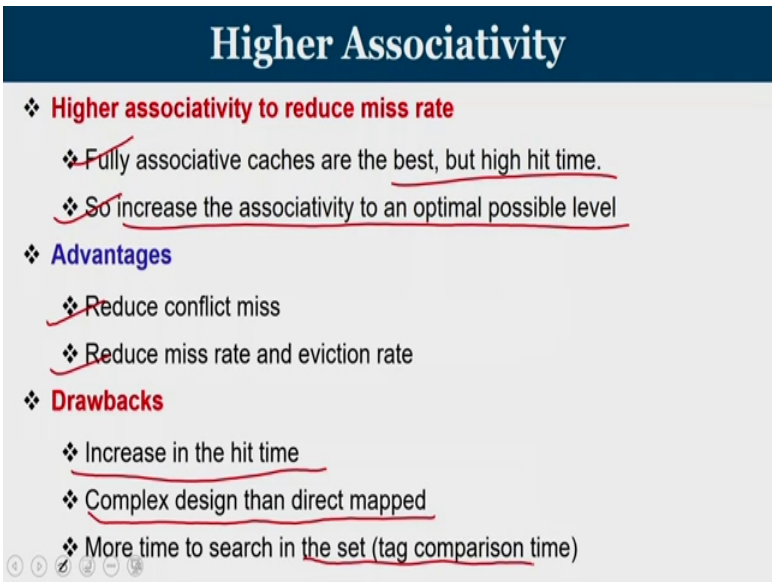
Refer Slide Time: 07:54



So, for a few applications like FFT, simple program, speech processing program, weather prediction program, we are trying to see can go for larger cache size, yes. And on the y axis we have miss rates. We have seen that all the applications when you increase cache size you can see that there is a dip in the miss rate.

But beyond the point your miss rate is not coming down even if you increase the cache size. That is because every cache or every application, whatever be the cache size, it will surely have compulsory misses. So, compulsory miss will be there even in an infinite cache.

(Refer Slide Time: 08:36)



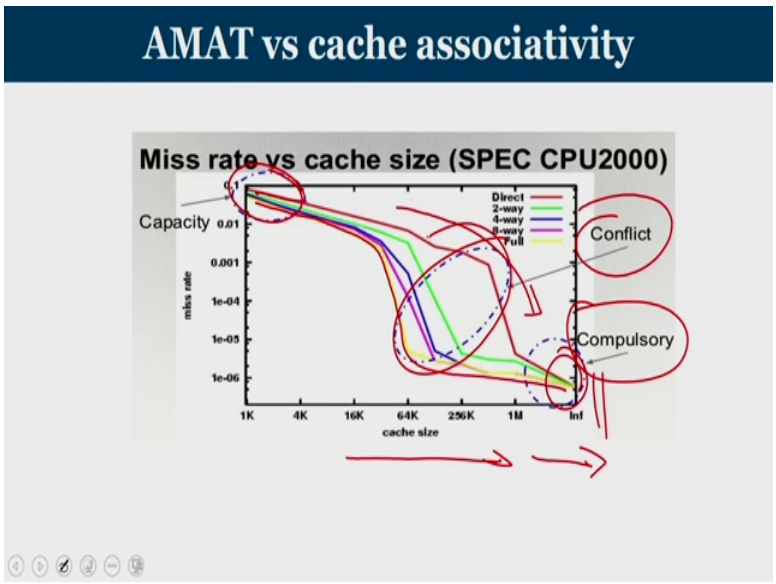
Now, the third approach to reduce miss rate is going for higher associativity. Fully associative caches are the best, but it takes more hit time because you have to search in all the ways and then find out where there is a hit happen, and then, how to take out the appropriate

data. It is good, but then it will take more time. So, the best possible way is do not stick on to direct mapped cache which will have lots of miss rates due to conflict miss, increase the associativity to optimal possible level.

What are the advantages? It will reduce conflict miss, it will reduce miss rate and eviction rate. But disadvantage is when you go from direct mapped cache to 2-way associative or 4-way associative cache, because of the extra multiplexer circuit, it will increase the hit time because in direct mapped cache you go and then directly find out what is the set and then extract the data if it is a hit.

But if it is a 2-way associative, you have to parallelly perform tag search in both of them and then where the hit occurred only from that place we can extract the data. So, that will produce slightly higher hit time and it has a slightly complex design than direct-mapped and more tag comparator circuits are required.

(Refer Slide Time: 09:54)



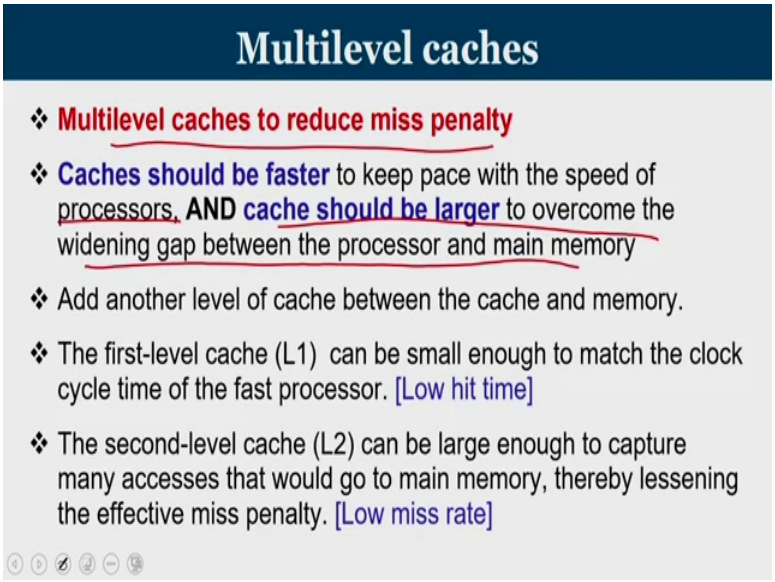
So, average memory access time, if you look into, this red one indicates a direct mapped cache. So, as the cache size increases, the misses will come down, and the yellow one is the same cache if it is fully associative, then the misses are very very less. Whatever be the associativity even if you increase to infinite cache still you have a lot of misses and they are the compulsory miss.

So, capacity miss is predominant if it is a small cache. Whatever large cache it is, still there will be some kind of a miss which is called compulsory miss. Based upon associativity the

conflict miss section varies. So, direct mapped cache has more conflict miss whereas fully associative cache has less conflict miss and 2-way, 4-way, and 8-way all other set associative caches have a conflict miss which is lower than direct mapped cache but higher than fully associative cache.

We have learned three techniques which are reducing the miss rate. First one is increasing the block size, second one is increasing the cache size, and third one is increasing the associativity. Now, let us continue with another approach. Can we reduce miss penalty? Yes. That is what we are going to see.

(Refer Slide Time: 11:19)



Multilevel cache is to reduce miss penalty. So, what should be the purpose of cache? Caches should be faster to keep pace with the speed of processor because cache is going to interact with the pipeline. So, hit time of the cache should be equal to the processor speed and cache should be larger to overcome the widening gap between processor and main memory.

So, what you can do is, rather than a cache and a main memory, have one more level of cache which is bigger than your first level cache and which is faster than your main memory. The first level of cache should be small enough to match the clock cycle time of the fast processor low hit time.

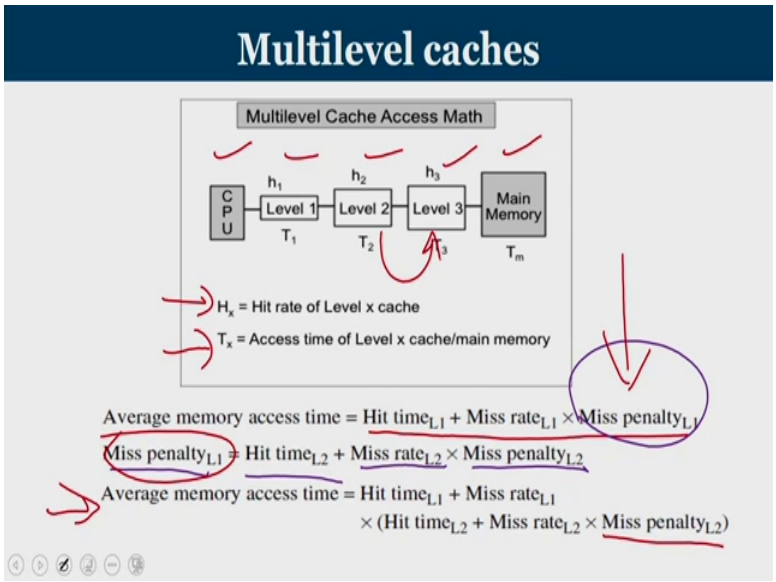
The second level cache should be large enough to capture many accesses that that would go to main memory, thereby lessening the effective miss penalty. So, if it is main memory only,

if it is cache and main memory only, any miss go all the way to main memory and bring multiple words of data and then store.

Since main memory is of chip it takes more time. But if you employ one more level of cache, the extra time that you needed all the way to bringing in something from main memory can be reduced much by taking something from the cache alone. It is the next level of cache. So, I can have L2 cache, L3 cache like that. Each cache is having more capacity, but they will take more hit time because the number of sets is more.

So, the idea of multilevel cache is L1 cache should be small and simple enough to have lower hit time; L2 cache should be large enough such that many of the misses from L1 should be accommodated in L2 as a hit, and it can effectively reduce miss penalty directly from main memory. Because if L2 is there, the miss penalty of L1 cache should be much smaller. If you are going to take something from L2, then L1 is directly interacting with your main memory.

(Refer Slide Time: 13:10)



This is the logical diagram of a multilevel cache access path. We have CPU, L1, L2, L3, and main memory, different levels of caches have been shown. Hx indicates the hit rate of level x and Tx indicates access time of level x. Let us say it is cash or main memory. Now how is memory access time defined as? Hit time of L1 plus miss rate of L1 into miss penalty of L1. Now, this miss penalty of L1 is defined again.

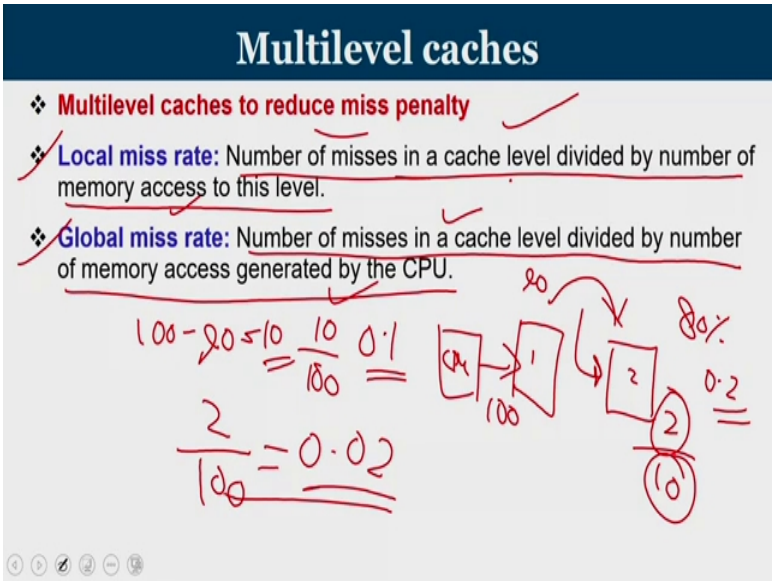
Avg mem access time= Hit time(L1) + Miss rate (L1) * miss penalty (L2)

Miss penalty of L1 is hit time of L2 plus miss rate of L2 into miss penalty of L2.

Miss penalty(L1)= Hit time (L2) + Miss rate (L2) * Miss penalty(L2)

Now, you substitute this miss penalty of L1 here. That is what you get in the last time. Average memory access time is hit time of L1 plus miss rate of L1 into hit time L2 plus miss rate L2 into miss penalty of L2. Now, miss penalty of L2 can be defined as hit time of L3 plus miss rate of L3 into miss penalty of L3. Like that it can be applicable in a nested circuit.

(Refer Slide Time: 14:24)



Now, these multilevel caches are going to reduce your miss penalty. Now, there are two things as far as miss rate is concerned. So far, our concept of miss rate is number of misses divided by number of memory access. The local miss rate, it is the number of misses in a cache level divided by number of memory access to this level. This is our normal miss rate.

Local miss rate= no. of misses in cache level/ no of memory access to this level

Now, global miss rate is number of misses in a cache level divided by number of memory access generated by CPU. So, consider the case that you have your L1 cache and your L2 cache. Now, this is CPU. Let us say CPU is going to make 100 requests to L1 out of which 90 of them is a hit. So, what is local miss rate? So, 90 of them is hit, so 100 – 90, that is 10 is the number of misses.

10 divided by 100, that is called local miss rate, number of misses in a cache divided by number of memory access. Out of 100 memory access 10 of them resulted in a miss, so local miss rate is 0.1. Now only this 10 will go to L2. Let us imagine in L2 80% of them are going

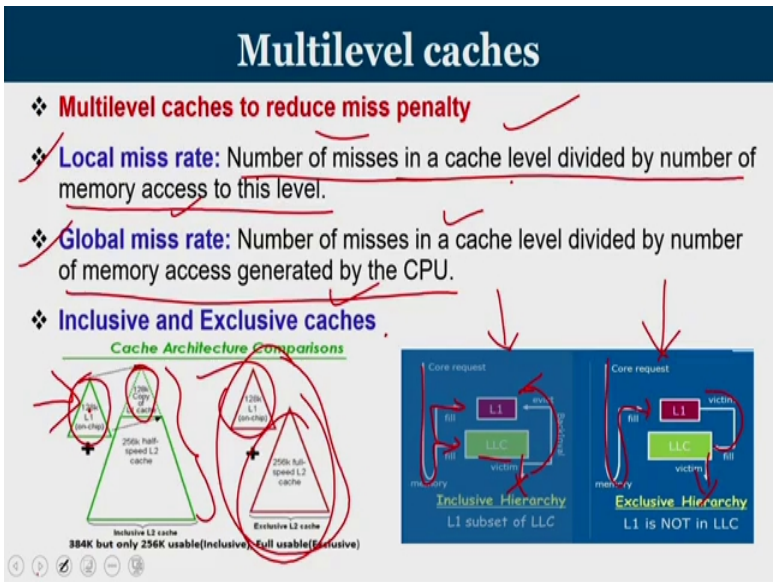
to be hit. So, I gave 10 requests out of which 80% is hit, that means only 2 is my miss. So, my local miss rate is 0.2.

Number of memory access is 10 out of which number of misses is 2, so 2 by 10, 0.2. So, 0.1 is the local miss rate of L1, 0.2 is the local miss rate of L2. Now, what is global miss rate? Number of misses in cache level divided by number of memory access generated by the CPU. Now number of misses I got is 2. What is the number of memory access that I have?

Global miss rate= no. of misses in cache level/ no of mem access generated by CPU

The number of memory access made is total 10 only in L2, but number of memory access generated by the CPU, CPU has generated 100. So, the local miss rate is 0.02. So, as far as L1 cache is concerned local miss rate and global miss rate are same, but for L2 cache, local miss rate and global miss rate are different. Now, when you have multiple levels of cache we have the context of whether they are inclusive cache or exclusive cache.

(Refer Slide Time: 17:00)



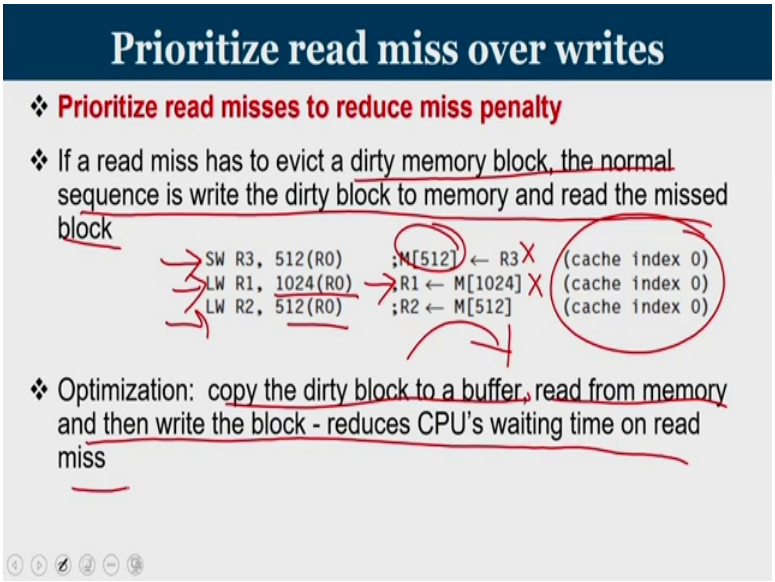
When your L1 cache is a subset of L2, then whatever is there in L1 surely it will be there in L2. That is called inclusive cache. When your L1 and L2 are mutually exclusive, so the contents in L1 will never be there in L2, then they are called exclusive cache. So, the working of inclusive cache and exclusive cache is slightly different.

So, think of a case where you have 128 KB of L1 cache and 256 KB of L2 cache, if it is inclusive hierarchy total on chip or total size on cache is not 256 + 128, because this 128 is

actually a copy that is there in L1 or L1 is actually copying from L2, whereas in the case of exclusive cache, the contents of L1 and contents of L2 are separate.

So, this diagram shows whenever there is a core request that comes in an inclusive hierarchy it goes all the way to memory, fill the last level cache which is called LLC, and then you fill from L1, and whenever there is eviction that has been needed, that has been thrown out, and then you are going to supply. Whereas, in this case you are first going to fill up in L1, that is going to evict, into L2, that that is the way how things are being processed out.

(Refer Slide Time: 18:25)



Now, there is the next miss, that is going to be prioritizing read miss over writes. So, if a read miss has to evict a dirty memory block, then the normal sequence is to write the dirty memory block and read the miss blocks. So, think of a case that you are going to perform a store operation on memory.

So, you are going to write something on a memory location and then you are going to load something from some other memory location, and load a new value from the same memory location. Let us imagine all of them are index to cache 0. So, I have to perform faster writes? So, imagine that it is a dirty block you are going to evict something. So, this is mapping into the same thing.

So, I encounter a read miss. So, here there is a write, then these are read miss. So, what we are trying to see is the optimization issue, how to copy the dirty block to a buffer and read from memory and then write the block, this will reduce the CPU's waiting time. So, you are

going to perform a write, and if it is a write through cache, you have to first write to cache and main memory together.

Now, think of a case that you have a dirty block, that is something that is written, now there is a read miss that comes. This read miss is going to evict out this block. Since it is dirty, what we saw is the eviction process will be over only if the dirty data is written to main memory. It will take a lot of time. In the meantime, the read miss is waiting.

So, the optimization is whenever you have a write operation and a read operation to be done, prioritize read over the write. So, this dirty write I will put up the value in the buffer, perform the read so that the block is being updated, you got a new block to read, processor will continue in the background, you complete the writing. That is the whole idea of prioritizing read misses over write.

So, we have learned five cache optimization techniques today to reduce average memory access time, larger blocks which will reduce miss rate, larger caches will reduce miss rate, and higher associativity also will reduce miss rate. Multilevel caches will reduce miss penalty and prioritizing read miss overwrites also will reduce miss penalty. There are a few more others. So, whatever we have learned is basic optimization. There are a few more cache optimizations that we will learn in the next lecture. Thank you.