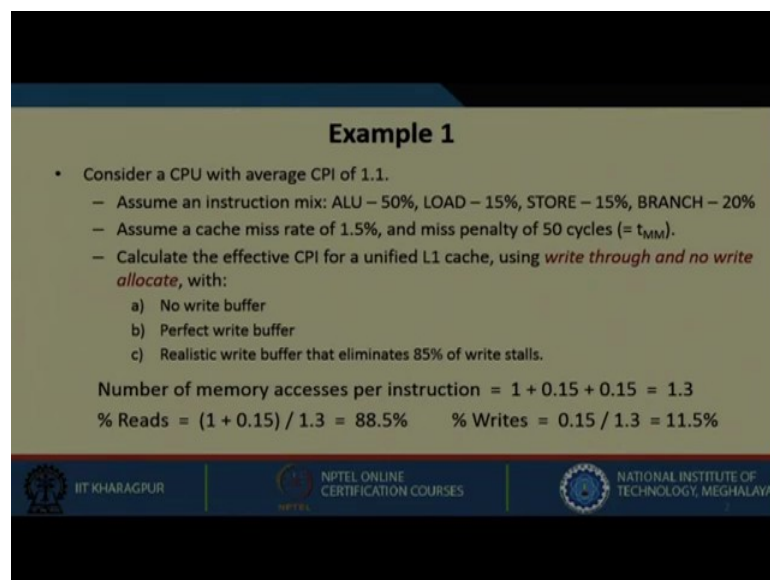


Computer Architecture and Organization
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture - 32
Improving Cache Performance

Welcome to the last lecture on Cache Memory. In this lecture we will be seeing some methods for improving the cache performance, prior to that we will look into two examples.

(Refer Slide Time: 00:31)



Example 1

- Consider a CPU with average CPI of 1.1.
 - Assume an instruction mix: ALU – 50%, LOAD – 15%, STORE – 15%, BRANCH – 20%
 - Assume a cache miss rate of 1.5%, and miss penalty of 50 cycles ($= t_{MM}$).
 - Calculate the effective CPI for a unified L1 cache, using *write through and no write allocate*, with:
 - a) No write buffer
 - b) Perfect write buffer
 - c) Realistic write buffer that eliminates 85% of write stalls.

Number of memory accesses per instruction = $1 + 0.15 + 0.15 = 1.3$
% Reads = $(1 + 0.15) / 1.3 = 88.5\%$ % Writes = $0.15 / 1.3 = 11.5\%$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us consider a CPU with average CPI as 1.1, let us assume that an instruction mix like ALU having 50%, load having 15%, store 15%, branch 20%, and assume that a cache miss rate is 1.5%, and the miss penalty that means, if there is a miss what is the time required to bring it from the next level that is; let us a main memory is 50 cycles. We need to calculate the effective CPI for a unified L1 cache that uses write through and no write allocate.

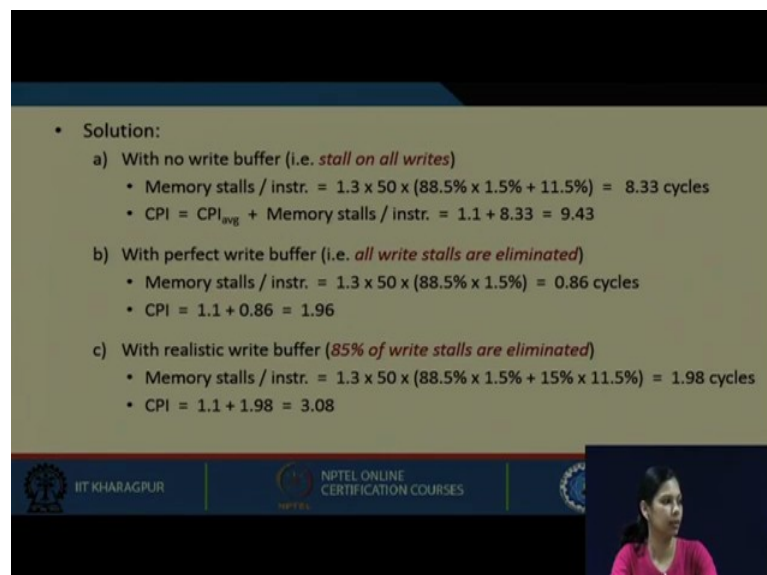
Along with this we will be calculating with no write buffer, next we will be doing with perfect write buffer, and another one we will be doing with realistic write buffer that eliminates 85% of write stalls.

Let us first calculate the number of memory accesses per instruction. When there will be a memory access there will be a memory access for load and store. So, let us say for fetching the instruction we need to have one cycle; we require one cycle for that and then 0.15 for load 15 percent of load instruction, and 0.15 for store where we will require memory operation. So, total is coming to 1.3.

So, for all memory accesses all instruction accesses is 1 and 15 percent for load and 15 percent for store. Now percentage read will be how much? Percentage read will be once we are reading a particular instruction and then again when we are reading when we are loading a word. So, 1 for reading every instruction, and 0.15 for load instruction, divided by total number of memory accesses per instruction which is coming down to 88.5 percent. So, percentage read is 88.5 percent.

Now, coming to percentage writes for the write we will require this 15 percent only, 15 percent divided by 0.13 that is the total number of memory access per instruction. So, now, we have found out percentage read, percentage write and we also know the total number of memory accesses per instruction per instruction what will be the total number of memory accesses.

(Refer Slide Time: 03:54)



• Solution:

- a) With no write buffer (i.e. *stall on all writes*)
 - Memory stalls / instr. = $1.3 \times 50 \times (88.5\% \times 1.5\% + 11.5\%) = 8.33$ cycles
 - CPI = $CPI_{avg} + \text{Memory stalls / instr.} = 1.1 + 8.33 = 9.43$
- b) With perfect write buffer (i.e. *all write stalls are eliminated*)
 - Memory stalls / instr. = $1.3 \times 50 \times (88.5\% \times 1.5\%) = 0.86$ cycles
 - CPI = $1.1 + 0.86 = 1.96$
- c) With realistic write buffer (*85% of write stalls are eliminated*)
 - Memory stalls / instr. = $1.3 \times 50 \times (88.5\% \times 1.5\% + 15\% \times 11.5\%) = 1.98$ cycles
 - CPI = $1.1 + 1.98 = 3.08$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, coming to the solution, in the first case we need to consider where we say that there is no write buffer. So, in this particular case there will be stall on all writes. So,

whenever there is no write buffer. So, we are not writing it to a high-speed hardware rather we are writing it back to the main memory or you can say the lower level memory.

So, in that case how do we consider the number of stalls ... there will be stalls on all writes. So, the memory stalls per instruction will be 1.3 multiplied by 50, into we have percentage read of 88.5 percent, this is percentage write of 11.5 percent and assume that there is a cache miss of 1.5 percent. So, we need to consider that as well. So, if we consider that and what is 50? 50 is our miss penalty. So, for miss penalty there will be 50 cycles. So, when we multiplied 1.3 into 50 into this percentage, we get roughly 8.33 cycles. This is the total number of cycles that we are getting; that means, these many memory stalls per instruction will be there.

So, what will be the CPI? CPI will be average CPI plus we have to consider this memory stalls CPI. So, this memory stalls per instruction is 8.33 we have calculated. So, total CPI will come down when we have no write buffer as 9.43 when we are using write through policy without any no write allocate.

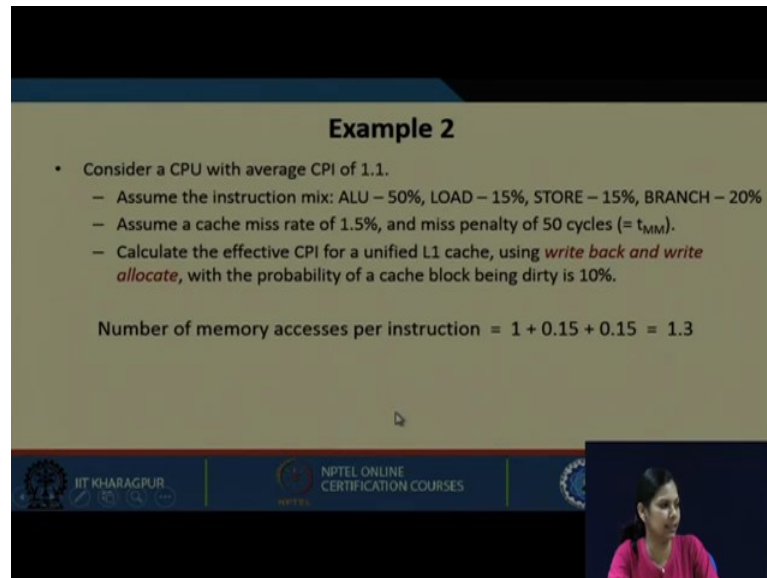
Let us see the next case where we have a perfect write buffer; in this particular case all write stalls are eliminated. So, there will be no write stalls now. So, it will be 1.3 multiplied by miss penalty, multiplied by the reads percentage read and the miss rate we have to multiply these 2 only, and we are not considering anything for the write in the previous case we have considered this 11.5, but here we are not considering that.

So, for this it is coming to 0.86 cycles. So, the CPI when we have a perfect write buffer we have 1.1 plus 0.86 which is coming to 1.96. So, we can see that there is a huge difference between when we have no write buffer and when we have some write buffer. Now let us consider a realistic write buffer. So obviously, perfect write buffer is difficult to have because there might be some writing going on and at the same time some read operation is also required.

So, in such cases if there are 85 percent of the write stalls are eliminated; that means, 15 percent will still be there, but 85 percent there will be no write stall. So, in that case how do we find out 1.3 into miss penalty, multiplied by this was the miss plus a read percentage for the read percentage it will be 1.5 and there will be 15 percent for the write.

So, let us just see this is 11.5 percent and for this 11.5 percent this will be 1.5 this is not 15 this should be 1.5. So, which will come down to if you solve this particular equation it will come down to 1.98 cycles. So, it is now coming down to CPI 1.1 plus 1.980, which is roughly equal to 3.08 which is to some extent a realistic value because we will definitely have some even if we have write buffer there will be some stalls.

(Refer Slide Time: 08:46)



Example 2

- Consider a CPU with average CPI of 1.1.
 - Assume the instruction mix: ALU – 50%, LOAD – 15%, STORE – 15%, BRANCH – 20%
 - Assume a cache miss rate of 1.5%, and miss penalty of 50 cycles ($= t_{MM}$).
 - Calculate the effective CPI for a unified L1 cache, using *write back and write allocate*, with the probability of a cache block being dirty is 10%.

Number of memory accesses per instruction = $1 + 0.15 + 0.15 = 1.3$

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a woman in the bottom right corner.

Let us consider another example where we have similar kind of values, but now we have calculate the effective CPI for a unified L1 cache using write back and write allocate; that means, we will not be writing it through to the main memory always, rather we will be writing it into the cache memory with write allocate; that means, we must have a space in our cache memory to write that particular thing.

With the probability of cache block being dirty is 10%, and we assume that 10% of the time the cache block will be dirty and if the cache block is dirty we know we knew what we need to do it in that particular case, we have to write back the data into the main memory and then we have to again read or write whatever we need to do the next.

(Refer Slide Time: 09:58)

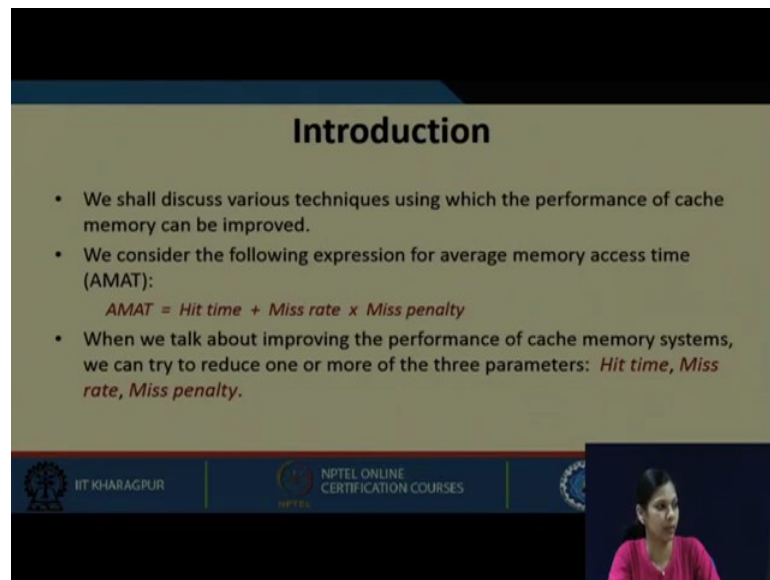
- Solution:
 - Memory accesses per instruction = 1.3
 - Stalls / access = $(1 - H_{L1}) \cdot (t_{MM} \times \% \text{ clean} + 2t_{MM} \times \% \text{ dirty})$
 $= 1.5\% \times (50 \times 90\% + 100 \times 10\%) = 0.825 \text{ cycles}$
 - Memory stalls / instr. = $1.3 \times 0.825 = 1.07 \text{ cycles}$
 - Thus, effective CPI = $1.1 + 1.07 = 2.17$

So, the number of memory accesses per instruction will be 1 plus 0.15 plus 0.15 that is 1.3. Now let us see the memory accesses per instruction which is 1.3, stalls per accesses will be one minus hit ratio of L1 x t_{MM} x percentage time it is clean, because when it is clean then we have to only write it once. If it is dirty then we have to do it twice we have already seen that in our previous lecture. The calculation is shown, which comes to 0.825 cycles.

Now we see the memory stalls per instruction. We will just multiply the total memory accesses per instruction and the memory stalls per instruction, and we get 1.07 cycles. Thus what will be the effective CPI 1.1 + 1.07 which is coming to 2.17 which is even less than the realistic write buffer if we use in our previous case when we are using write through policy.

So, these are the 2 examples we have discussed for writing into the cache using write through and write back policies.

(Refer Slide Time: 11:46)



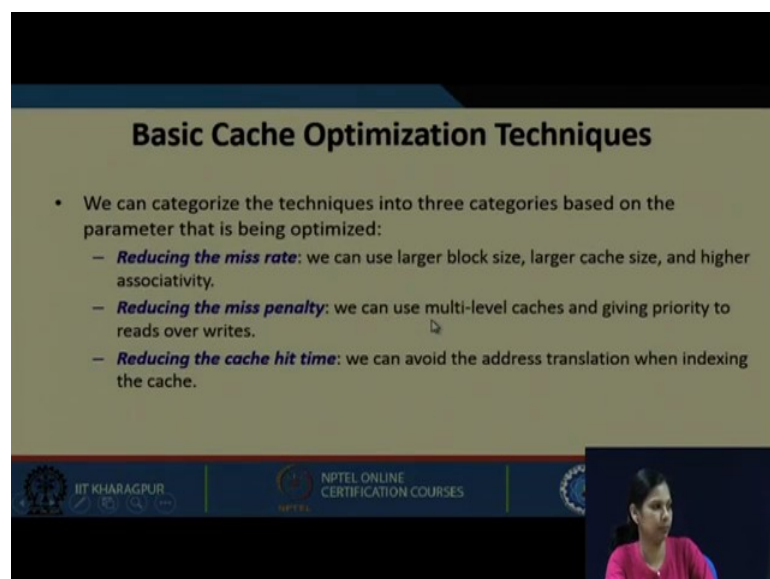
Introduction

- We shall discuss various techniques using which the performance of cache memory can be improved.
- We consider the following expression for average memory access time (AMAT):
$$AMAT = Hit\ time + Miss\ rate \times Miss\ penalty$$
- When we talk about improving the performance of cache memory systems, we can try to reduce one or more of the three parameters: *Hit time, Miss rate, Miss penalty.*

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, coming to the methods that we will be looking into for cache improvement, how we can further improve the cache. Let us consider this expression of average memory access time AMAT, which is the hit time (basically hit ratio multiplied by the access time of the cache) plus miss rate into miss penalty. So, when we talk about improving the performance of this cache memory system, ultimately we need to reduce AMAT. So, either we can reduce hit time, or reduce miss rate or miss penalty.

(Refer Slide Time: 12:48)



Basic Cache Optimization Techniques

- We can categorize the techniques into three categories based on the parameter that is being optimized:
 - **Reducing the miss rate:** we can use larger block size, larger cache size, and higher associativity.
 - **Reducing the miss penalty:** we can use multi-level caches and giving priority to reads over writes.
 - **Reducing the cache hit time:** we can avoid the address translation when indexing the cache.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

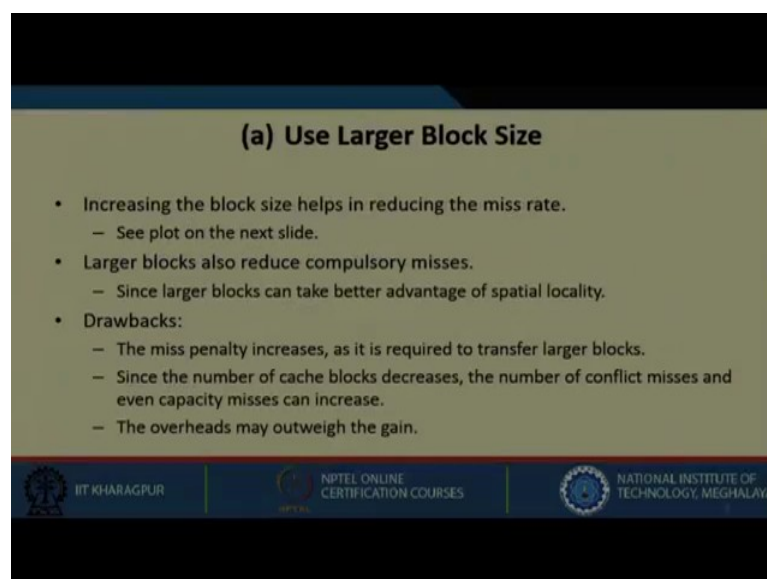
So, we need to see how we can take care to reduce all these parameters. In basic cache optimization techniques we categorize the techniques into three groups based on the parameters. How we can reduce the miss rate? Miss rate can be reduced by using larger block size. If you use a large block size then if my program is pretty large and you have a large block, the whole program is brought to some block of the cache and you can access it nicely. So, larger block size in turn reduces the miss rate.

Now how do we reduce miss penalty? By miss penalty we mean that if you have multi level cache we bring the data from main memory into the upper level of memory, and then from upper level of memory it goes to cache. The miss penalty can be reduced by having multi level caches.

Now, how do we reduce the cache hit time? When we are hitting cache and we are searching for the TAG, we are matching something. From the address part we have already seen we have certain parts for different kinds of mapping technique; if you are using set associative mapping technique you have a TAG you have a SET you have a WORD. So, you need to match that TAG with the number of blocks that you are having. When you are matching this TAG the CPU generates a logical address, the logical address gets translated into physical address and then that matching takes place.

Instead once we know the logical address, from the logical address if we can extract the TAG and then can directly start this matching, the hit time can be reduced.

(Refer Slide Time: 15:07)



(a) Use Larger Block Size

- Increasing the block size helps in reducing the miss rate.
 - See plot on the next slide.
- Larger blocks also reduce compulsory misses.
 - Since larger blocks can take better advantage of spatial locality.
- Drawbacks:
 - The miss penalty increases, as it is required to transfer larger blocks.
 - Since the number of cache blocks decreases, the number of conflict misses and even capacity misses can increase.
 - The overheads may outweigh the gain.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

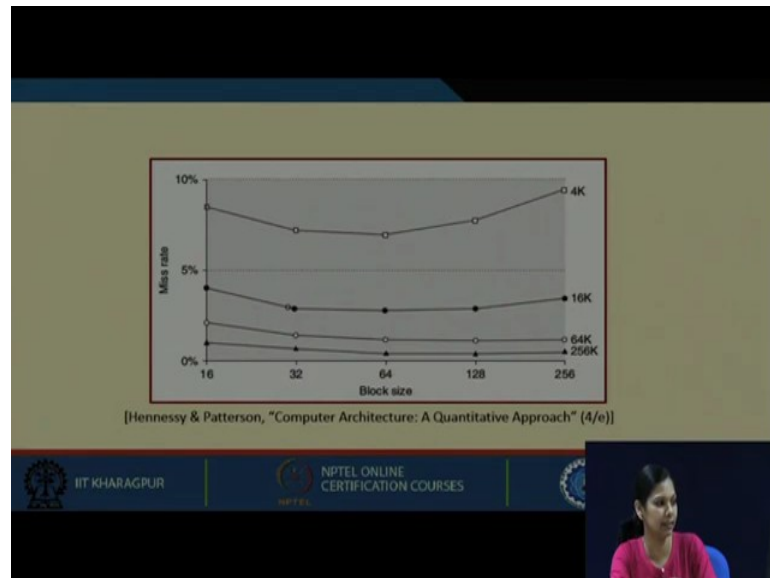
Let us see each one of the points in some detail. When we use larger block size what happens? Increasing the block size helps in reducing the miss rate; obviously, if you have a larger block size the entire program you can bring to the same block and then you can access in a faster fashion using that.

We have various kinds of misses that also we have seen in the previous lecture. Larger block size will also reduce the compulsory misses. Every time you bring a data there is a miss, but if you want to bring a data you bring a larger block where all the data are now brought in. And now you will be just hitting the cache to get the subsequent one, the time will be spend in bringing the entire data from the lower level of the memory to higher level of the memory, but then you can access it very easily, but what will be the drawback. The miss penalty increases as it is required to transfer a large block.

So, for bringing a large block, there will be more number of words in that particular block. You have to bring the all the words into your cache. In that particular case you have to wait until the entire block is brought in. So, the miss penalty will increase, but in turn the hit time will increase. Since the number of cache blocks decreases the number of conflict misses and even capacity misses can increase. So, conflict miss will happen we know for direct mapping and for set associative mapping, where different blocks of main memory are mapped to a same block of cache memory.

In that case this conflict will be more because you have smaller number of blocks because the block size you have increased. If the cache size is limited and the block size is more, smaller number of blocks will be there. So, if you want to bring two blocks together then it will be a problem. So, it can be seen that if you have a larger block size the overhead that we are getting might outweigh the gain, the gain which we were expecting you might not get, but the overhead becomes much more.

(Refer Slide Time: 18:11)



Let us see this particular diagram. This is the miss rate, this is the block size and this is the cache size. When the cache size is small the miss rate decreases, but after certain point the miss rate again increases, whereas if you have a larger block size you can see that the miss rate decreases and there is an increase, but not so much here.

(Refer Slide Time: 19:03)

- Selection of block size:
 - The optimal selection of the block size depends on both the latency and the bandwidth of the lower-level memory.
 - High latency and high bandwidth
 - Encourages large block size since the cache gets many more bytes for a miss for a nominal increase in miss penalty.
 - Low latency and low bandwidth
 - Encourages smaller block sizes since more time is required to transfer larger blocks.
 - Larger number of smaller blocks may also reduce conflict misses.

We have seen that if you increase the block size the miss rate will decrease. In such case, other overheads will also be there; conflict misses and capacity misses will also be there,

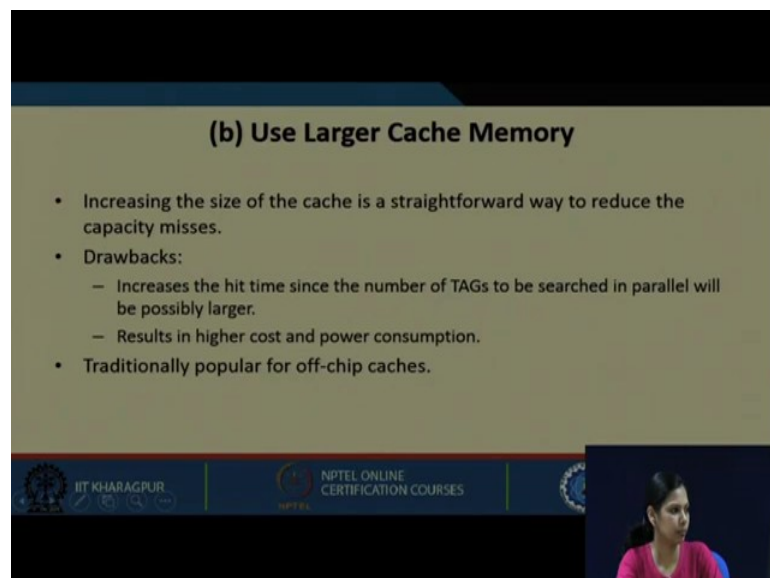
then how do we select the block size? The optimal selection of block size depends on both the latency and the bandwidth of the lower level memory.

If we have high latency and high bandwidth this encourages large block size, since the cache gets many more bytes for a miss for a nominal increase in miss penalty. So, with nominal increase in miss penalty we get more bytes from a miss. If you have low latency and low bandwidth this encourages smaller block size since more time is required to transfer large blocks of course, more time will be required to transfer larger blocks.

Larger number of smaller blocks may also reduce conflict misses if you have more number of blocks, then this conflict miss can get reduced.

In such situation what can happen is that if you have large number of smaller blocks, you will have various opportunities of mapping different blocks. So, the conflict misses can even reduce.

(Refer Slide Time: 21:11)



(b) Use Larger Cache Memory

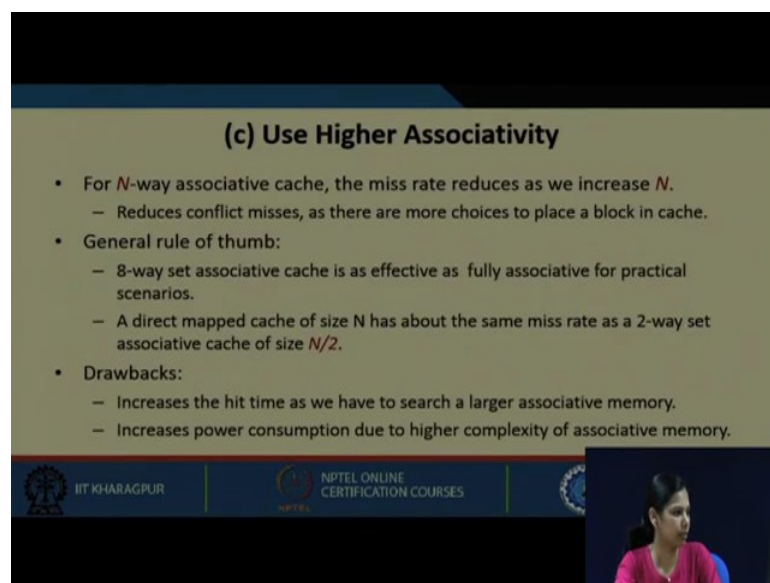
- Increasing the size of the cache is a straightforward way to reduce the capacity misses.
- Drawbacks:
 - Increases the hit time since the number of TAGs to be searched in parallel will be possibly larger.
 - Results in higher cost and power consumption.
- Traditionally popular for off-chip caches.

The slide is part of a video lecture. At the bottom, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. A small inset video shows a woman in a pink shirt speaking.

Next is use larger cache memory. So, if you have larger cache memory, then it will be easy. So, let us see how it can improve. Increasing the size of the cache is straightforward. It is a straight way straightforward way to reduce capacity misses, but what is the drawback then? Increases the hit time since the number of tags to be searched in parallel will be possibly large.

So, one of the parameter is increasing. So, we are decreasing the misses the miss rate is decreasing, but the search time for the tag in the cache memory will increase, that is, the hit time result in higher cost and power consumption of course. We know that cache memory is built using static RAM that requires roughly 6 transistors for one bit. So, it is much more costlier compared to data memory. If you are increasing the cache memory you are increasing the cost as well. So, all these are related. So, which one you will reduce to how much such that you will get a better gain in turn needs to be analyzed and understood.

(Refer Slide Time: 22:38)



The slide is titled "(c) Use Higher Associativity" and contains the following text:

- For N -way associative cache, the miss rate reduces as we increase N .
 - Reduces conflict misses, as there are more choices to place a block in cache.
- General rule of thumb:
 - 8-way set associative cache is as effective as fully associative for practical scenarios.
 - A direct mapped cache of size N has about the same miss rate as a 2-way set associative cache of size $N/2$.
- Drawbacks:
 - Increases the hit time as we have to search a larger associative memory.
 - Increases power consumption due to higher complexity of associative memory.

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a woman in the bottom right corner.

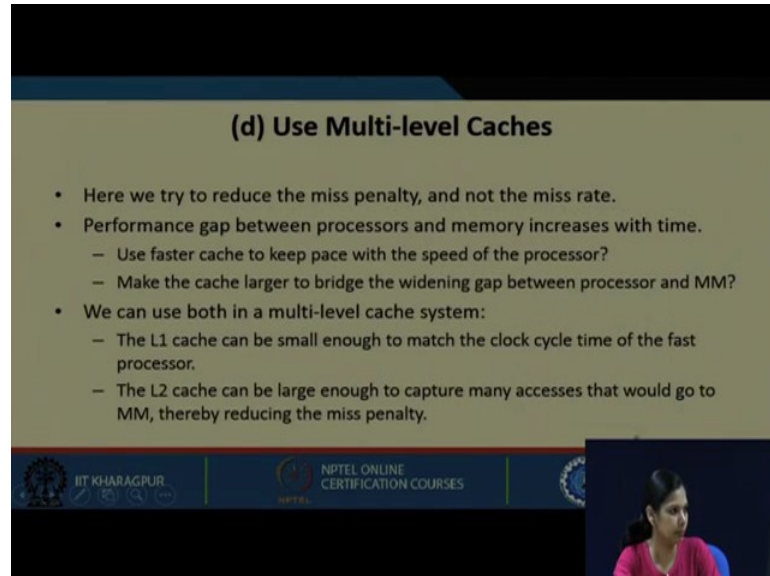
We can use higher associativity. So, for a N -way associative cache the miss rate reduces as we increase N . This reduces the conflict misses as there are more choices to place a block in a cache. As the size of this N increases what we can do is that more number of blocks from main memory can be placed in same set in that particular cache.

In that way the conflict misses will definitely reduce. So, there is a general rule of thumb: 8-way set associative cache is as effective as fully associative for practical scenario. So, we need not have to pay the cost for a fully associative cache.

So, direct map cache of size N has about the same miss rate as a 2-way set associative cache of size $N/2$. So, if we have a $N/2$ size 2-way set associative cache, it will be same as direct mapping. The direct mapping will be easy to implement as well. So, what is the

drawback of this? Increases the hit time and as we have to search a larger associative memory.

(Refer Slide Time: 25:05)



(d) Use Multi-level Caches

- Here we try to reduce the miss penalty, and not the miss rate.
- Performance gap between processors and memory increases with time.
 - Use faster cache to keep pace with the speed of the processor?
 - Make the cache larger to bridge the widening gap between processor and MM?
- We can use both in a multi-level cache system:
 - The L1 cache can be small enough to match the clock cycle time of the fast processor.
 - The L2 cache can be large enough to capture many accesses that would go to MM, thereby reducing the miss penalty.

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom, and a small video inset of a presenter in the bottom right corner.

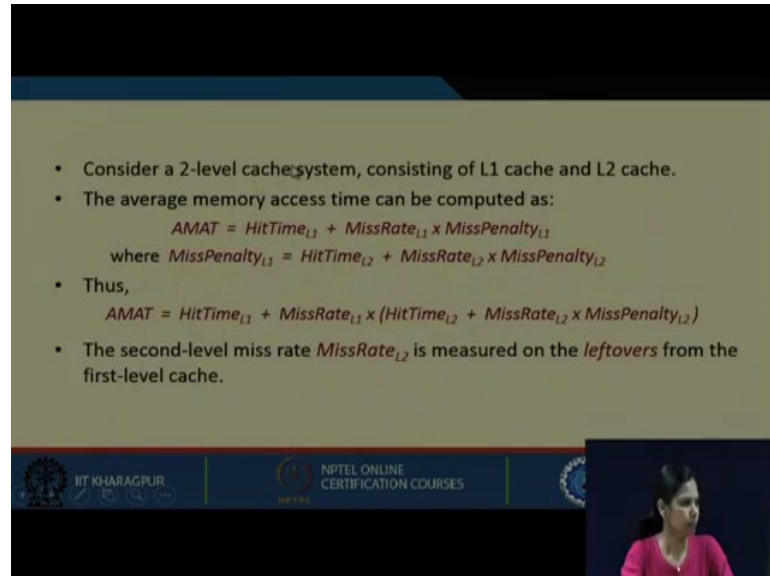
This is another way use of multi level caches; here we try to reduce the miss penalty and not the miss rate. So, we are not reducing the miss rate rather we are trying to reduce the miss penalty, that is the time to bring the data from your main memory to cache memory and then to processor.

The performance gap between processor and memory increases with time that we already know, use faster cache to bridge the widening gap between processor and main memory. Multiple number of levels of cache will be helpful because we are bringing the data from main memory and we are keeping in those multi level caches, and whenever it is required by the processor and if it is not present in L1 cache, it is looking into L2 and L3 to find out the data and in most cases they are getting it from L1 and L2 and they do not have to go to main memory to access the data. The L1 cache can be small enough to match the clock cycle time of the fast processor, the L2 cache can be large enough to capture many accesses that would go to main memory thereby reducing the miss penalty.

So, what we are trying to say here is that L1 cache will be small enough and it will be much faster, and L2 cache will be little larger such that most of the data from main memory will be there in L2 cache, and whenever it is not present in L1 cache I am just

getting it from L2 cache and not from main memory. So, we are getting some advantage out here.

(Refer Slide Time: 27:44)



• Consider a 2-level cache system, consisting of L1 cache and L2 cache.

• The average memory access time can be computed as:

$$AMAT = HitTime_{L1} + MissRate_{L1} \times MissPenalty_{L1}$$

where $MissPenalty_{L1} = HitTime_{L2} + MissRate_{L2} \times MissPenalty_{L2}$

• Thus,

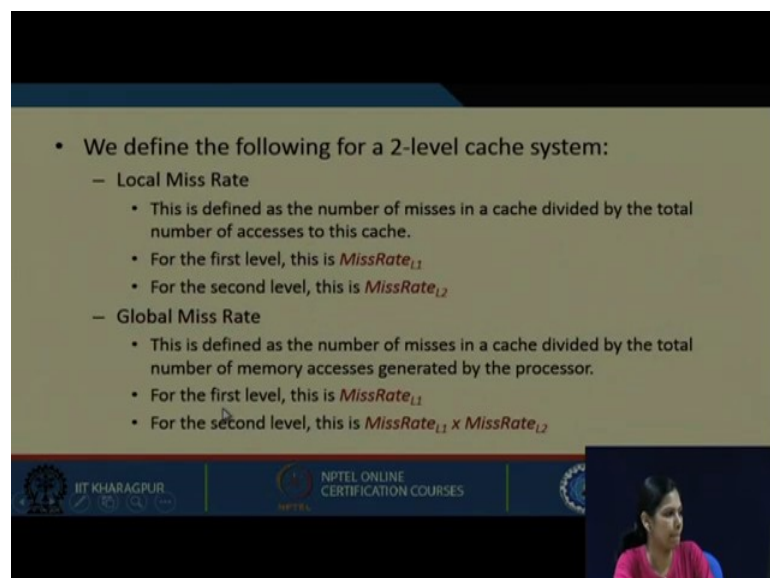
$$AMAT = HitTime_{L1} + MissRate_{L1} \times (HitTime_{L2} + MissRate_{L2} \times MissPenalty_{L2})$$

• The second-level miss rate $MissRate_{L2}$ is measured on the *leftovers* from the first-level cache.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, let us consider a 2-level cache system consists of L1 cache and L2 cache, the average memory access time can be computed as shown.

(Refer Slide Time: 28:35)



• We define the following for a 2-level cache system:

- Local Miss Rate
 - This is defined as the number of misses in a cache divided by the total number of accesses to this cache.
 - For the first level, this is $MissRate_{L1}$
 - For the second level, this is $MissRate_{L2}$
- Global Miss Rate
 - This is defined as the number of misses in a cache divided by the total number of memory accesses generated by the processor.
 - For the first level, this is $MissRate_{L1}$
 - For the second level, this is $MissRate_{L1} \times MissRate_{L2}$

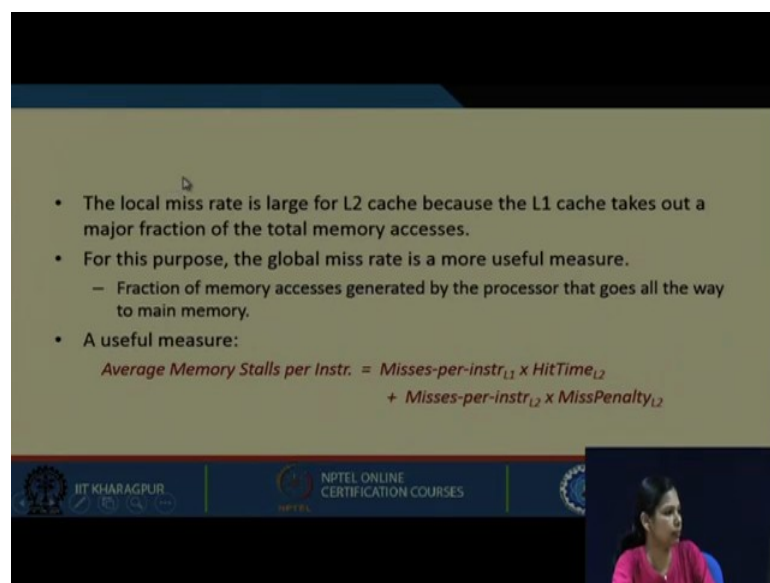
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We define the following for the 2-level cache system here. One is call local miss rate; this is defined as the number of misses in a cache divided by total number of accesses to this cache. So, for the first level this is miss rate of L1, and for the second level this is

miss rate of L2. Now what do you mean by global miss rate? This is defined as the number of misses in the cache divided by total number of memory accesses generated by the processor.

So, we are not separately taking miss for L1 and miss for L2; rather we are taking in terms of the total. This is in terms of total number of memory accesses generated by the processor. For the first level this is miss rate of L1, but for the second level it will be miss rate of L1 x miss rate of L2.

(Refer Slide Time: 29:51)



The slide contains the following text:

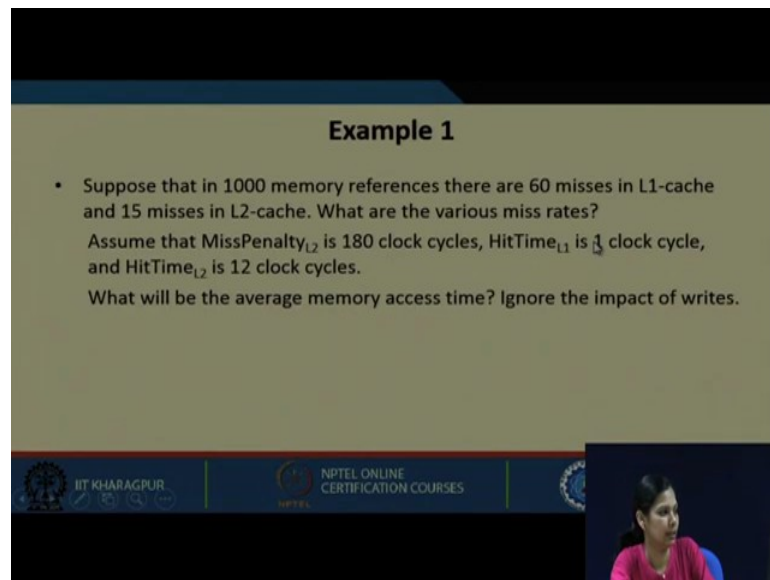
- The local miss rate is large for L2 cache because the L1 cache takes out a major fraction of the total memory accesses.
- For this purpose, the global miss rate is a more useful measure.
 - Fraction of memory accesses generated by the processor that goes all the way to main memory.
- A useful measure:
$$\text{Average Memory Stalls per Instr.} = \text{Misses-per-instr}_{L1} \times \text{HitTime}_{L2} + \text{Misses-per-instr}_{L2} \times \text{MissPenalty}_{L2}$$

The slide footer includes the logos of IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a presenter.

The local miss rate is large for L2 cache because the L1 cache takes out a major fraction of the total memory access, because we are separately calculating for this purpose the global miss rate is more useful measure. So, we generally use this global miss rate, fraction of memory access is generated by the processor that goes all the way to the main memory.

A useful measure that can be used is average memory stalls for instruction, defined as shown.

(Refer Slide Time: 30:50)



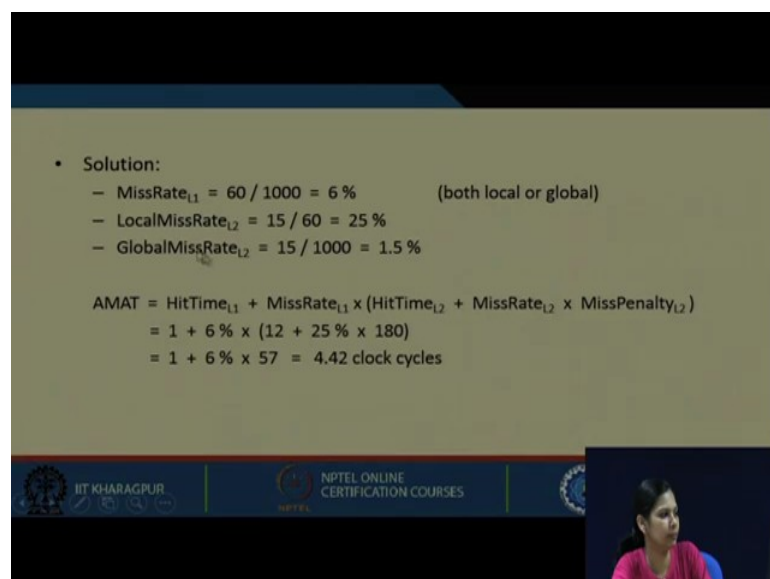
Example 1

- Suppose that in 1000 memory references there are 60 misses in L1-cache and 15 misses in L2-cache. What are the various miss rates?
Assume that MissPenalty_{L2} is 180 clock cycles, HitTime_{L1} is 1 clock cycle, and HitTime_{L2} is 12 clock cycles.
What will be the average memory access time? Ignore the impact of writes.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see this example. Suppose that 1000 memory references are there and there are 60 misses in L1 cache and 15 misses in L2 cache, what are the various miss rates. So, total is 1000 out of which 60 misses are there in L1 and 15 misses are there in L2. Assume that miss penalty is 180 clock cycles, hit time of L1 is 1 clock cycle, and hit time of L2 is 12 clock cycle, what will be the average memory access time ignore the impact of writes?

(Refer Slide Time: 31:34)



Solution:

- $\text{MissRate}_{L1} = 60 / 1000 = 6\%$ (both local or global)
- $\text{LocalMissRate}_{L2} = 15 / 60 = 25\%$
- $\text{GlobalMissRate}_{L2} = 15 / 1000 = 1.5\%$

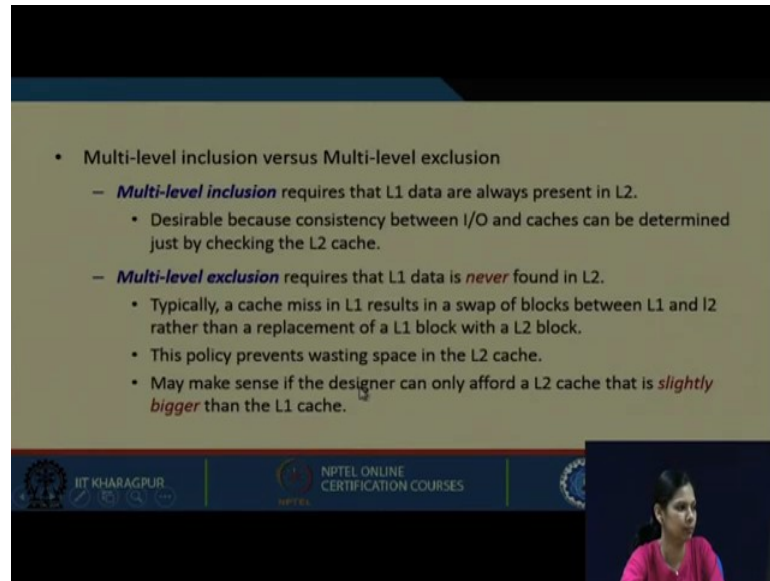
$$\begin{aligned} \text{AMAT} &= \text{HitTime}_{L1} + \text{MissRate}_{L1} \times (\text{HitTime}_{L2} + \text{MissRate}_{L2} \times \text{MissPenalty}_{L2}) \\ &= 1 + 6\% \times (12 + 25\% \times 180) \\ &= 1 + 6\% \times 57 = 4.42 \text{ clock cycles} \end{aligned}$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The miss rate of L1 will be $60 / 1000$, because total we are considering that is 6% both for local or global.

Miss rate of L2 means how many misses total in L1 is 60; out of 60 there are 15 misses. So, $15 / 60 = 25\%$, and global miss rate for L2 will be $15 / 1000 = 1.5\%$.

The calculation of AMAT is shown (Refer Slide Time: 32:41)



The slide contains the following text:

- Multi-level inclusion versus Multi-level exclusion
 - **Multi-level inclusion** requires that L1 data are always present in L2.
 - Desirable because consistency between I/O and caches can be determined just by checking the L2 cache.
 - **Multi-level exclusion** requires that L1 data is *never* found in L2.
 - Typically, a cache miss in L1 results in a swap of blocks between L1 and L2 rather than a replacement of a L1 block with a L2 block.
 - This policy prevents wasting space in the L2 cache.
 - May make sense if the designer can only afford a L2 cache that is *slightly bigger* than the L1 cache.

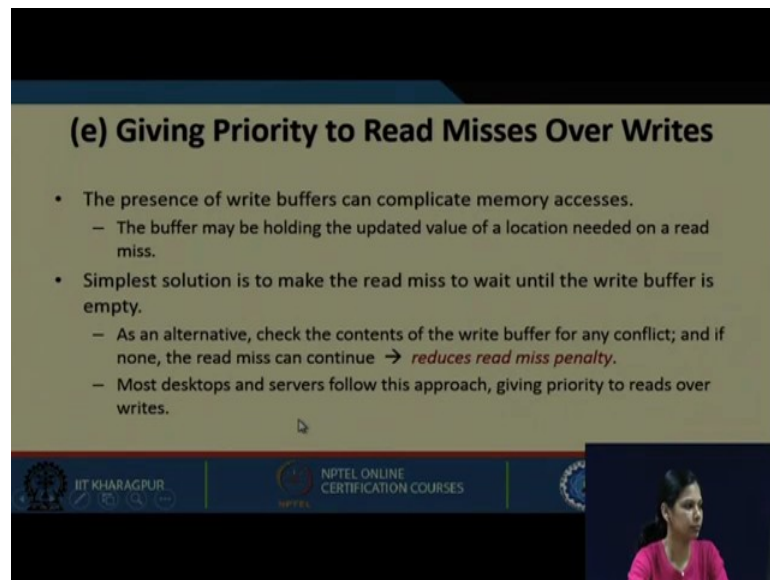
The slide footer includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL, along with a small video inset of a woman in a pink shirt.

Let us see another thing; multi level inclusion versus multi level exclusion. Multi level inclusion requires that L1 data are always present in L2 cache, inclusion means it is included. So, in L1 we have certain data and L2 cache is the next level cache where we are saying that whatever data is present in L1 is also there in L2, that is called multi level inclusion. This is desirable because for the consistency between IO and caches can be determined just by checking the L2 cache.

The IO can just check the L2 cache and find out the data. What is multi level exclusion? This requires that L1 data is never found in L2; that means, L1 data are certain data which is there in L1, and L2 data are certain data that is present in L2, but there is no common data in between. That means, whatever is there in L1 is not there in L2. Typically a cache miss in L1 result in a swap of block between L1 and L2, rather than replacement of L1 block with a L2 block.

So, there is no replacement rather you are bringing a block from L2 to L1. This policy prevents wasting space in L2 cache.

(Refer Slide Time: 34:50)



(e) Giving Priority to Read Misses Over Writes

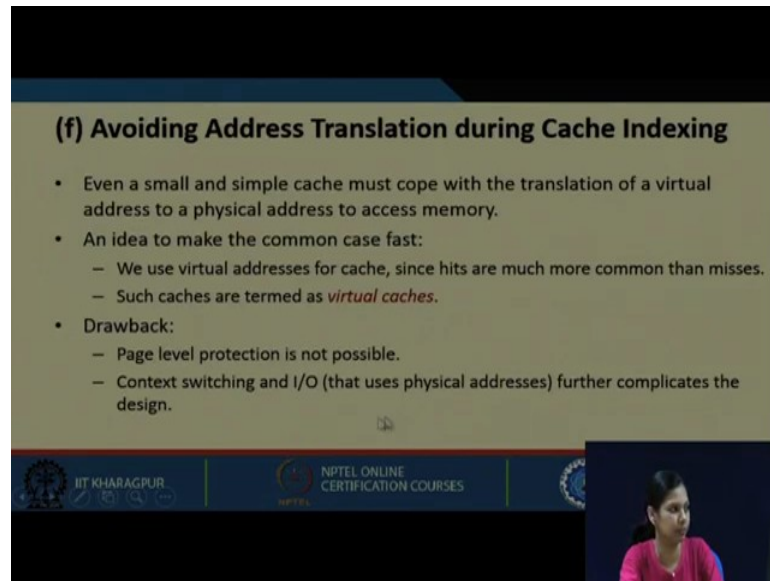
- The presence of write buffers can complicate memory accesses.
 - The buffer may be holding the updated value of a location needed on a read miss.
- Simplest solution is to make the read miss to wait until the write buffer is empty.
 - As an alternative, check the contents of the write buffer for any conflict; and if none, the read miss can continue → *reduces read miss penalty.*
 - Most desktops and servers follow this approach, giving priority to reads over writes.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now, giving priority to read misses over writes. We know that reads are more frequent, the presence of write buffers can complicate the memory accesses, the buffers may be holding the updated value of the location needed on a read miss. So, whenever there is a read miss the buffer may be holding an updated data. The simplest solution that is there is to make the read miss to wait until the write buffer is empty.

As an alternative what can be done is check the content of the write buffer for any conflict, if there is any conflict we have to check the write buffer and if none the read miss can continue.

(Refer Slide Time: 36:05)



(f) Avoiding Address Translation during Cache Indexing

- Even a small and simple cache must cope with the translation of a virtual address to a physical address to access memory.
- An idea to make the common case fast:
 - We use virtual addresses for cache, since hits are much more common than misses.
 - Such caches are termed as *virtual caches*.
- Drawback:
 - Page level protection is not possible.
 - Context switching and I/O (that uses physical addresses) further complicates the design.

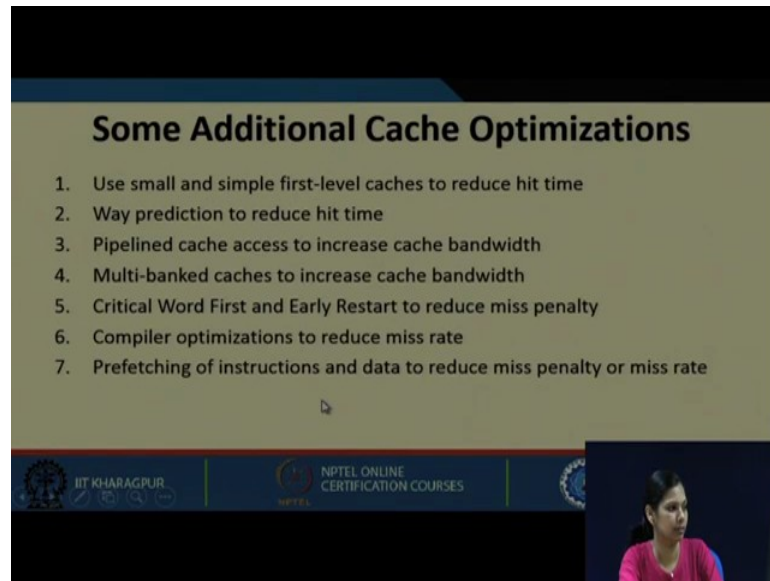
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Another is avoiding address translation during cache indexing. As I said that whenever CPU is generating a logical address, it is first translated into physical address and then we extract the TAG bits that are matched. Finally, we see that whether the data or instruction is present in cache memory or not. Even a small and simple cache must cope with the translation of virtual address to physical address to access memory. So, this has to be done.

As in an idea to make the common case first, what can be done we use virtual addresses for cache since hits are much more common than misses. Such caches are termed as virtual caches. We will have some virtual caches in place to make the common case fast, but what is the drawback? We are saying that we will not be having a translation rather we will be extracting the address from the logical address only.

Page level protection is not possible, this is a drawback and context switch and IO that uses physical address further complicates the design, because this IO and the context switching; that means, switching between two processes is required often. So, that process becomes more complicated.

(Refer Slide Time: 37:43)



Some Additional Cache Optimizations

1. Use small and simple first-level caches to reduce hit time
2. Way prediction to reduce hit time
3. Pipelined cache access to increase cache bandwidth
4. Multi-banked caches to increase cache bandwidth
5. Critical Word First and Early Restart to reduce miss penalty
6. Compiler optimizations to reduce miss rate
7. Prefetching of instructions and data to reduce miss penalty or miss rate

The slide is part of an NPTEL presentation from IIT Kharagpur. It features a dark blue header with the title 'Some Additional Cache Optimizations' in white. Below the title is a list of seven optimization techniques. At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. A small inset video of a presenter is visible in the bottom right corner of the slide frame.

These are some additional cache optimizations like to use small and simple first level caches to reduce hit time, way prediction to reduce the hit time, pipeline cache access to increase cache bandwidth, multi-banked caches to increase the cache bandwidth, critical word first and early restart to reduce the miss penalty, compiler optimization to reduce miss rate and prefetching of instruction, and data to reduce miss penalty or miss rate. So, these are some more cache optimization through which we can reduce hit time, miss penalty and miss rate.

There are also some more methods through which miss rate, hit time, and miss penalty can be further reduced.

We are at the end of lecture 32 and we are done with memory system. So, in this week what we have seen are various kinds of techniques through which we can reduce the memory access time, such that it can bridge the speed gap between the processor and the memory.

Thank you.