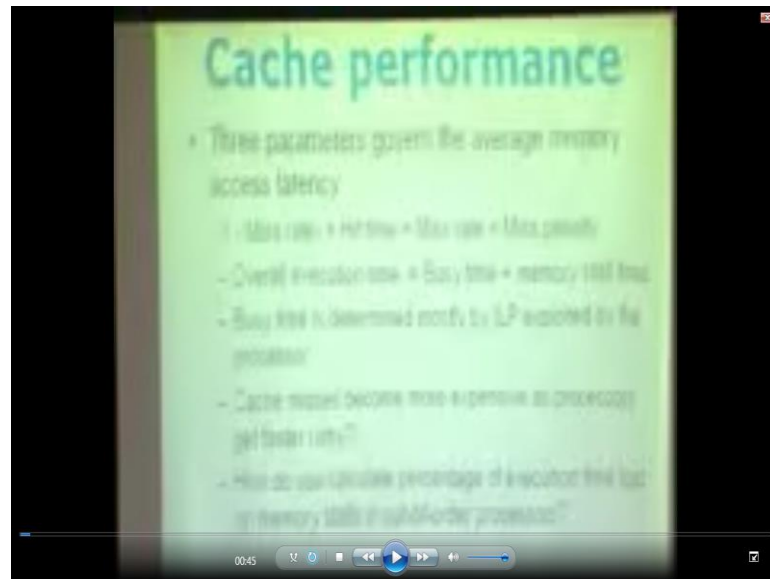


**Computer Architecture**  
**Prof. Mainak Chaudhuri**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kanpur**

**Lecture - 27**  
**Virtual Memory and Caches**

(Refer Slide Time: 00:14)



So, we were discussing about caches. So, today we will try to modify it in the performance of cache and try to address the system components that effect cache performance. So, this is the basic equation that dictates average memory access latency  $1 - \text{miss rate} \times \text{hit time} + \text{miss rate} \times \text{miss penalty}$ . So, it is a basic this much of this much of fraction you hit multiply that by the time taken for each hit. And the remaining 1 accounts for the time and overall execution time can be now framed as busy time plus memory stall time. So, busy time essentially is a time where processor is doing something time is a time when processor is stalled in memory. So, here I also include other stall times in a busy time therefore, the other stores also for example, you may have shortly function. So, busy time is determined mostly closely by I L p exploited by the processor.

So, it is exploited by the processor that of course, depends on how rich the processor is it belongs machineries for extracting I L p how many. So, if it has a full byte if it is a partial byte was other slot slots like this. because of the an. So, there are not part of the memory stall they are all part of this. So, cache misses become more expensive as

processors get faster why is that So, I have a hits to memory module and I have 2 processors; 1 as the 2 giga hertz; one has the 4 giga hertz. And I am saying that for the 4 giga hertz processor the cache is so cache misses become more expensive as processors get faster why is that? So, I have a hits memory module and I have 2 processors 1 answer 2 giga hertz 1 answer 4 giga hertz. And I am saying that for the 4 giga hertz processor, the cache misses are very good memory expensive do you understand more time.

Student: More time I mean percentage of time.

Right exactly. So, if you look at this one if you increase frequency in the processor this is like. So, your portion of memory stall time is looking because this is constant if you get the processor which is going to remain constant this sort of your time so go down gradually. So, in a limit you will be only having memory sort the processor will be, but the limit. Of course, that is not possible really to nullify this completely always you have super fast processor which takes more time to do you know instructions that is what possible. And the question is that when you when you are really talking about this they often becomes so, it often becomes important to measure the memory stall. For example, if you are analyzing the particular program computer architect routinely tries to quantify the execution time into this 2 parts the question how do we. So, the way if I give you a singular term the problem is that. So, when out of order issue processor, there are many things happening concurrently.

So, there may be a memory request outstanding cache has taken a miss which was already gone out from memory in the meantime processor may be doing something else may be actually instructions. So, how did we quantify this particular component that is very important then only you will be going to understand what will execute itself. So, how do you do this? Any idea why this is a question? Alright, because there may be certain memory operations latency of which is partially? So, how do you modify this is. So, I will give you 1 one answer to this although this is not techniques. Also, usually what computer architects do is that they look at the commit stage of the pipeline instructions are leading the pipeline lastly one. Let us suppose that you have a commitments of  $w$  that you can retire  $w$  instructions that we started alright.

So, then what we say is that if we can retire at least 1 instruction inside it that is a busy sign alright. If you cannot retry any instructions cycle that is a stall cycle stall cycles may

come from various reasons. Now, a memory stall cycle is a cycle is a stall cycle when the when the commit stage could not retire any instruction and instruction at the top of the R O B is a memory operation. So, you know that the reason why you could not retire an instruction was because of a memory operation waiting at the top of the R O B which could not be completed for some reason you stall alright. So, that is the memory stall time and everything else is probably the busy line is it clear? So, it is a very simple way of accounting cycles. So, retirement stage and then they account to every cycle as a busy cycle or a stall cycle. Stall cycles you look at the top of the R O B with that instruction is a memory stall memory operation that is a memory stall cycle any question on this?

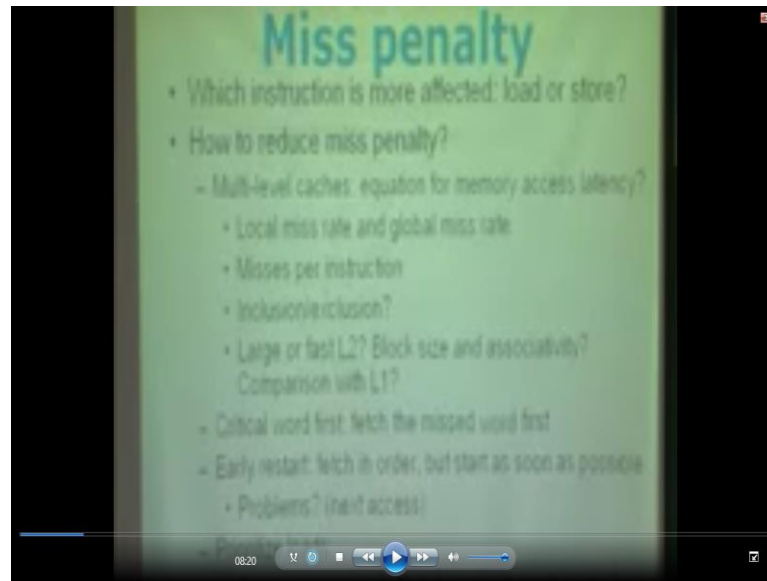
Student: Sir if it is the, if it retires 1 instruction.

Yes

Student: Still it will.

Yes the stall may be, because we cannot commit, because you are saying that suppose in a cycle I retire  $n$  instructions  $n$  is in  $w$ . And I could not retire more because  $x$  plus one instruction is a memory operation yes you could go. So, head and do that also right, but then the, is that to write end up having a very small number here. Because it is very alright that you have a lot of cycles where can consume the full commit bandwidth very alright. So, I meant to so, this is this is again you know I mean. So, say here I am seeing is that if you can do anything could not do could not do anything then it is a stall side. So, yes, you can come up with other techniques also the only point is that you should be able to argue your way out that this miss is correct alright. So, what I do now is take each of the components of this equation right. Essentially there are 3 terms 1 is miss rate 1 is hit time and other one is miss rate and you see how to improve each of them alright. So, as you can see here the latency is initially a increasing function of each of this alright. So, I should be reduce each of this times alright so let us see how to do that.

(Refer Slide Time: 08:17)



So, let us first look at miss penalty so miss penalty is associated by number of cycles that you spent on a cache list waiting for the come from the next level of the memory hierarchy cache. So, the first question is which instruction is more effected loads or stores? So, we have a 2. So, which one is more by miss penalty I will miss a store in the cache when there is a load in the cache which one is memory?

Student: Load

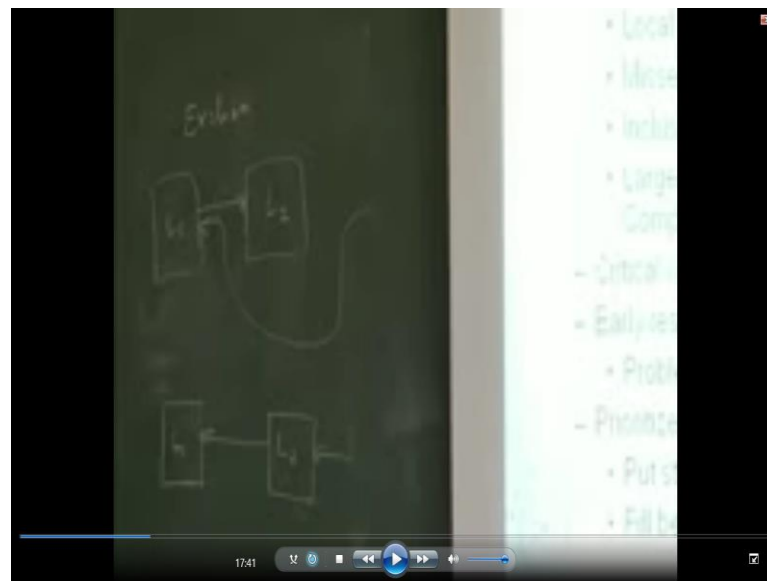
Why?

Student: The load might be bringing in a value for a register.

Right exactly. So, there may be other operations depending on the load right. So, load value being more critical for programming whereas, the value that is stored by a by a store instruction is not important. So, stores actually terminate a chain of that last instruction whereas, loads usually is at the root of a dependence that instructs alright. So, you must execute the root of the dependence as early as possible. So, the loads are most effected by miss predicting. So, how do we reduce miss penalty? Because here are some simple solutions you can have multi-level caches we have already discussed that last time in a levels of caches in hierarchy. And you can also derive an equation that looks very similar to this for a multi-level cache hierarchy alright. So, this is for single level cache hierarchy right you have a miss rate of the cache you have a hit time whenever

miss penalty in your 2 levels of cache if you derive a singular formula right. So, essentially your so this could be the this could be the L 1 L 1 time in L 2 whatever L 1 has this will now have some portion of into the hit in L 2. And the remaining portion will go out and L 2 in this rate multiplied by the miss penalty of the next level alright. Now, when you have a multilevel cache hierarchy how do you compute the misses that is a very important question here.

(Refer Slide Time: 10:46)



So, you have essentially that suppose you have a 2 levels of cache right here L 1 as L 2. Now, what is the miss rate on this hierarchy is it a L 1 miss rate is it L 2 miss rate what is it? So, there are 2 terms that are important; one is called local miss rate that is attached to a particular level of the cache alright. So, if I talk about the L 1 miss rate and L 2 miss rate and other one is called the global miss rate that is if you observe here alright. The whatever misses you get you divide that by if you observe here the total number of accesses into the hierarchy alright that the global miss rate alright. So, misses go out here accesses come in here something happens in between whatever misses you see divided by into the number of accesses is a global miss rate alright. And each level will have a local miss rate that is L 1 miss rate will be basically the count of this divided by the count of this alright L 2 will be this divided by this.

So, and also people use misses for instruction So, if you are if you are looking at the total number of misses here you can divide that by total number of instructions. That is it is

only that you are not dividing it by number of accesses instead you are dividing by the number of all instructions alright. So, this is this is a better representative of performance that is we may have a very very high miss rate. For example, the program let us say have a program has let us say 100 load store instruction right. So, you are observing here to observe there are 100 accesses coming in alright and your answer here you find that all 100 accesses actually miss alright.

So, here we conclude that we have a 100 percent miss rate alright. The question is, is it good or bad? If that is not you can get enough you say whether it is good or bad, but this 1 will actually tell you how much is going to be in fact the performance. Because what they happen is that when you are running a program which has 1 giving an instructions or 100 even if you miss all of them in the cache does not alright. So, this one gives you a very nice matrix for estimating performance in fact of misses alright misses for instruction when you have a multilevel cache hierarchy. You have to decide whether they should be inclusive or exclusive or something somewhere in the middle. So, we talked about inclusion last time let me introduce exclusion a little bit here.

So, inclusion was that you say that the contents of L 1 is a subset of L 2 alright. So, that is what we said that is that is the inclusion alright when you have exclusion what happens is that. So, essentially the contents of L 1 and L 2 are disjoint right. So, whatever you have in L 1 they will not be in L 2 right how do you really guarantee it? So, the way to guarantee one way to guarantee it is that this is how processors implement exclusion whenever the processor access something into L 1 right if it misses in the L 1 then if as usual alright L 2 will be looked up. So, if it increases in L 2 it will go into memory alright on the return path it will not be fit in L 2 it will directly be fit into L 1 alright.

For example] block is evicted from L 1 it will be allocated to L 2 alright. So, you can see that all the time there is an there is the contents here is disjoint from the contents here alright. Now, what will happen is that the processors access something in L 1 this is in L 1 request in L 2 and the block is actually L 2 alright. So, there what you do is you de-allocate the block from L 2 and copy it into L 1. So, that guarantees exclusive what is the, what are the advantages of inclusion versus exclusion? What is good about exclusion compare to inclusion? We will see an advantage of exclusion or inclusion; say again.

Student: More data can be stored in the cache .

More data can be stored in the cache why is that exactly yes. So, you can make full utilization of the cache phase given into the processor. There is no an implication of data that is a major advantage of exclusion comforts more effective capacity right. Any disadvantage of exclusion compare to inclusion so in inclusion so this is exclusion alright. So, when you have inclusion what you do on the fill path you fill into both right on your miss block comes from memory you fill in both L 2 and L 1 alright when the block is evicted from L 1 what do you do? Sorry well it is evicted. So, of course, it is gone so you do anything extra what study you right back to L 2 when the block is dirty you just drop the block what happens here when you evicted from L 1 where L 2 right at that time? So, you see any problem with that, but every eviction from L 1 in this case you have to allocate the block in L 2 in this case on every eviction from L 1 only if it is dirty. You have to say the copy back to L 2 otherwise you can drop the block which one is better or presently matter.

Student: One more search eviction.

One more eviction yes fine. So, that eviction has taken plus already in inclusion at the time of L. So, there is nothing extra going on it does evictions the set how much data is transferred between L 1 and L 2 in this case. Or in this case you say anyone on every eviction the data has to go from here to here it will require a much bigger bandwidth 1 is here only the multi blocks will go from here to here that is it. So, exclusion that is the major problem, the bandwidth requirement between the levels of hierarchy. So, here we talked out it is a larger capacity it will to a bandwidth to alright. So, you decide based on your you know design whether to support inclusion or exclusion we will talk a little bit more about exclusion in a little later. How do L desire this particular problem this bandwidth issue it can be it can be alleviated to some to some extent. So, we have a 2 level hierarchy how do we desire the second level of the cache? Right the L 2 cache. So, it will be a larger so it will be a larger for past we know that these 2 things cannot go large ones will be how would the block size and associativity how do they compare against L 1?

So, let us go one after another should we have a fast L 2 or large L 2 what is the option? We should have a large L 2 right that is a whole part it should be at least larger than L 1

otherwise there is no point in having a multilevel hierarchy right. In fact, if you have an inclusive hierarchy if your L 2 cache is equal to L 1 cache then your L 2 cache will contain exactly the same data as we have anything alright. So, we should be at least bigger than L 2 L 1 cache in inclusive hierarchy however in the exclusive hierarchy. You can have the same size actually the raise of property is such however you cannot be have a bigger one so that you can hold data. How about block size and associativity cache in comparison should be more than L 1 less than L 1 equal to L 1. Why do not we have blocks and why do you organize the caches in terms of blocks? We just have small wards why we have blocks what advantage you will get? What is it called?

Student: Spatial look.

Spatial look at alright so, bigger blocks often use spatial then bring larger in 1 short and hope to use them in your future. So, in can you answer this question how should be a block size can be larger should it be larger? Yes that can be larger can be anything actually I am assuming what should you do? It should be larger you get spatial look at right can be smaller smaller than any 1 block size what is the answer? Yes. No why right at I have a 64 byte L 1 block size and 32 byte L 2 block size what the problem? We are talking about block size yes, right yes it was other. So, why I will have guaranteed number of sets? It will basically not necessary the capacity of a cache is. No why you want hold the here number of sets they can be smaller can you involve misses? So, take the example of L 1 block size of 64 bytes and L 2 box have 32 bytes. How many I do misses? Do I need to solve to page 1 you are on the cache block, how many L 2 misses? Why to solve to this 1 L 1 cache 1?

Student: 2.

2 right Yes, needed 2 accesses. Two accesses to memory right that does not make any sense at all. Actually so, it should be at least as large as a L 1 block size now, do you see anything special about the exclusion when to take something extra from the block size. And I have bigger block size as an exclusive hierarchy L 2 cache block size bigger than L 1 block size. So, we have just computed that the L 2 block size should be at least as large as L 2 block. For exclusive hierarchies they have to be there is no at option actually why they have to be equal in the sense that without introducing more complexes. Yes you can have larger can you explain why we have got that from L 2 hit I de allocate the



block from here and bring it to L. So, I will copy it from L 2 to L 1 what happens if I have a 32 byte L 1 cache block and 64 byte L 2 cache block? Yes I something has to be done with the remaining data right yes. So, either you introduce 2 tags for cache block and maintain status of both the halves which essentially saying that L 2 cache blocks.

So, for exclusive hierarchies the simple solution is to have equal blocks on this alright. So, which is why if you if you go back to the history of a L u processors will actually be bring it exclusive hierarchies you at the block size they are all equal. Whereas, if you go back to the history of Intel processors you will find that after a certain point up to like Pentium and so on. They had to block size and behave the L 1 block size these are the, which you may not discuss in this case. So, the point is that for exclusive hierarchy you cannot have bigger block sizes in the upper levels of the cache. What about associative what do you think L 2 cache associativity L 1 cache associativity what we mean associativity what is the purpose?

Student: access. It proves the access peak.

No wait first let us start at the first question why do you need associative what is the purpose what kind of misses?

Student: Conflict.

Conflict misses, correct exactly. So, should we add more associativity in L 2 cache or less or equal what you think here? Clearly I can do anything right pretty much anything right practically what make sense what you are getting one?

Student: Less than associative

Why?

Student: Because so here giving set full associative cache will be.

That is right yes more associative which become more 100 that is like yes that is true, but what are you competing latency?

Student: Number of collisions will reduce the.

Sorry number of collisions will.

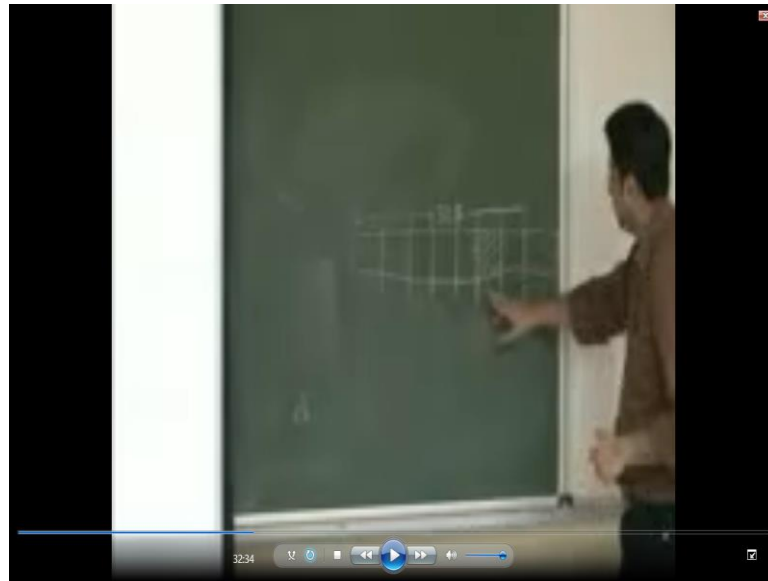
Student: Reduce the ((refer: 28:20))

Well, if they are both conflicting in L 1 might be both we miss because they keep on hashing everyone. So, purpose of L 2 should be that they should not L 2 also right. So, you want to make it more associative right and also qualities that making a cache more associative. Normally slows it down, but the question is what am I competing is what is my next level that is a huge latency right. So, anything you say it is going to be beneficial alright. So, that is your multi-level cache hierarchy other technique for reducing miss penalty is critical word first this pretty much common sense. So, critical word is the word within a cache block on which the miss have actually happen processor want it to access this particular word within the cache block and miss the cache.

So, it actually make sense to send this particular word first and then with the, with the other words of the cache block alright hardly restart. So, this is very similar to critical word first accept that you fetch in order so start from word zero and go in sequence. But you start as soon as you get the critical word alright so that is the, it was going from the entire cache block. So, what is the problem with this the problem is that if the processor is going sequentially and all the cache block will probably suits fall on the next what of the of the cache block. So, you have to take care of this actually alright. So, that is one problem and the last solution for improving penalty miss penalty is prioritize loads. Because you have already that loads are the biggest loads are effected most by miss penalty.

So, essentially what it means is that put stores in write buffers to write through cache and you set the loads first to the next level of the of the cache and you could fill before filling write back cache. So, this is actually says that when the block comes in what we happen is that we have to replace the block to make room for the new block alright. So, you feel that you complete the fill first before sending the write back if the if the block if the evicted block is dirty. So, that that gives you that allows the processor to start early right yes this one yes so you what is the problem? This problem the problem is that so, let us let us take an example right.

(Refer Slide Time: 31:23)

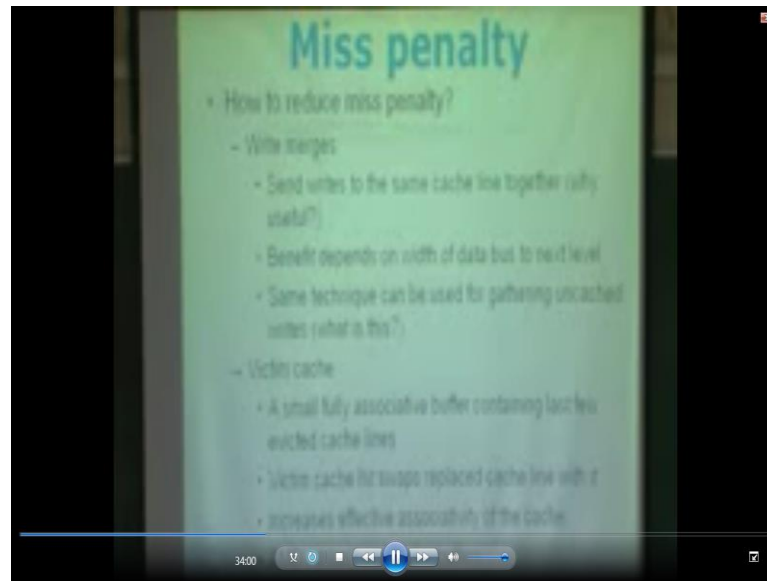


So, this is somewhere I have a thirty byte block. So, it has 8 words 8 each each 4 bytes right. So, let us suppose that etcetera. So, let us suppose that we miss here now we want this word alright. So, what early distracts says is that your fetching order and as soon as you get this word you start the processor. Well, what is the processor is actually accessing this block sequential. So, immediately after this it will probably stall here if it is not that is what the problem is talking about. So, what he is saying is that if it is if the processor is doing a sequential access this may not buy much that is all it is saying you stall very soon. And that problem is there here also this critical word is to say any question on this.

Student: more.

Start. We normally start here and different processors follow different protocols a simple protocol is to ((refer: 32:55)) out alright make site of very strange ordering of words in the fetch path means if the first 1 will be this 1. But it will not be any some other ordering of the words any other question.

(Refer Slide Time: 33:07)

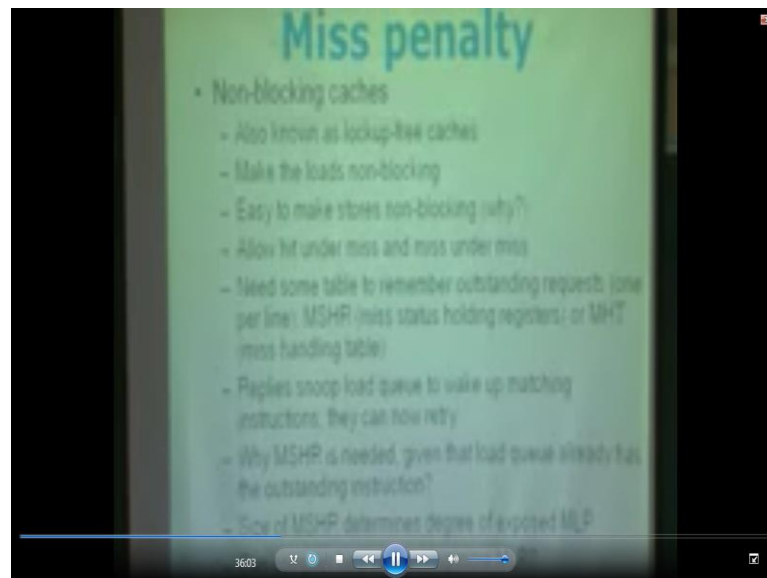


So, continuing this penalty, so, as we said here we wanted to de prioritize the stores alright. So, you put the stores in the right buffer alright for right through cache so that the stores will be, all the values to the next level hierarchy right. So, the buffer the next level of the cache will pick up the values another to do the stores. So, essentially what was saying is that the L 1 cache will not wait to the stores to complete. So, it has other implications which we will not discuss in this class, but this is what it is wrong prioritize the loads the question is what about the write back caches. So, these one actually talks about earning write backs alright

So, what he is saying is that send writes to the same cache line together alright. So, why is it useful? Because it implies your value more effectively instead of sending. So, you know modes of the time you can say the bigger chunk of data to get alright. So, benefits depends on width of the data bus to the next level same technique can be used for gathering uncached writes. So, these are the writes that do not go to cache they go directly to the memory you can attract the same technique. And finally, victim caching it is very common to Intel processors what you do is you put a small fully associative buffer containing last few evicted cache lines. So, essentially what I am saying is that between L 1 and L 2 you put a small buffer which can contain a few cache blocks alright. So, whenever something is evicted from L 1 you put it here alright.

So, the hope is that this block was evicted from L 1 cache, because of some conflict miss. So, it will be it will be reduced very soon from this buffer that is why you put it here. So, the victim cache hit swaps the replaced line with it. So, whenever you hit in the victim cache you will actually swap this line with the line hit with L 1 cache alright. So, it increases the effective associativity of the cache that is that is what you actually achieve AMD athlon has an 8 entry victim buffer pretty. Well, this is the size of the victim buffer 860 such a small buffer usually effective for L 1 caches not for L 2 caches. Because L 2 cache is already having a large associativity it is a small victim cache with it does not mind much. Whereas, L 1 caches are normally small associativity so, putting a victim cache will help any question on this.

(Refer Slide Time: 35:59)



Non-blocking caches were discussed this a little bit. So, essentially the idea was that your cache should be able to handle multiple outstanding misses alright and you talked about this is the in connection with memory level of alright. So, that is also known as block of free caches essentially make a loads non-blocking. So, because what will happen is that the load instruction goes missing the cache inside the memory or with the time the cache very strong which continues to start subsequent requests. So, it is easy to make stores non-blocking because there is nobody waiting for the store so if you just forget about the store. So, it is not an easy for load because there are instructions after it that is waiting. So, you stall those instructions until the load is completed here though the cache may be it will be handle other memory request to the mean.

So, it allows hit hits under misses and misses under misses you need a table to remember outstanding requests 1 per line this is called miss status holding registers or miss handling table. So, this essentially holds the outstanding miss addresses this is addresses that are whenever a reply comes back you just snoop the load queue to wake up the matching instructions they can now retry alright So, there will be request when while they comes back it will look up the load key alright that will be at least 1 sure which is why the miss was set. So, this instruction wakes up and probably the cycle if you reach the cache hit why m s h r is needed given that the load key already has the outstanding instruction I just say that in the table here. But then I am saying that it retries and come back and find out the instruction why do look at the situation where the instruction was not found in the load queue then the reply comes back sorry say again. Why you are right why should it get?

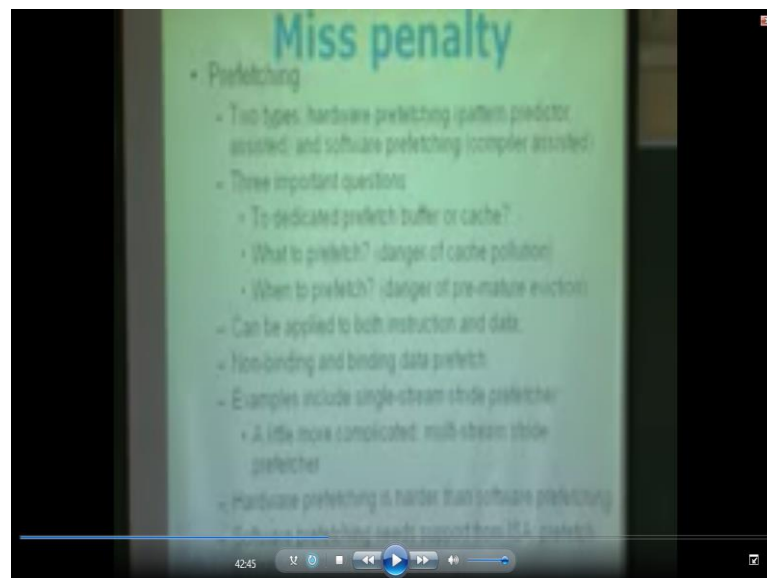
Student: Preceded by a Cisco.

Right alright. So, yes that is that is possible anything else can you have other reason why we get? So, I saying is that this load is in the shadow of Cisco and because the Cisco executes from the pipe is empty which has to be cancelled yes it is possible, but any way. So, assure that you handle Cisco usually the decoder will catch it early that a Cisco can be it would not even issue anything any other reason why the load may get disappeared from the pipeline got it what for which this load is control dependent on branch. This load will probably be removed much before the data comes back right. And for reasons for systems you cannot discard the data now you have to fill the data in the cache because the other external environment has been told that you have the data alright.

So, we cannot discard it anymore that is what that will let consistency. So, we have to fill the data in the cache although you may not need it now alright and that is why this table is needed to make sure that the cache is not receiving an unsolicited response alright. So, this is primary to make sure that this block will actually request. So, when you require it will be matched to get the table alright. So, size of M S H R determines the degree of exposed memory level parallelism. And normally it is small it aid to 16 entries right can compiler help in exposing more memory level parallelism. So, say that essentially memory level parallelism talks about situations where I have a large number of memory request in quiet parallel alright. So, that the certain latencies can be over mapped how can the compiler help any idea?

So, memory level parallelism can be increase device in cluster number of misses to get right number of cache misses together I need to cluster them in 1 point. So, the compiler can actually infer what do they miss in the cache by looking at the source code I can actually reorder instructions. So, we can cluster the load instructions together which are likely to miss in the cache. So, they will all go out together and have a lot of overlapping. So, problems of increasing the size of MHTS essentially the it will slow down your cache controller, because every time a request reply comes back in order to sort to the table to find out an entry alright. So, it slows down the controller it also controls the lot of energy that is why you have to sort bit large whatever I need miss penalty.

(Refer Slide Time: 42:45)



So, the so basic idea is that I want to predict what about the accessing future and bring them into my cache. So, that by the time I will actually access the data it is already in the cache. So, ideally I would have a zero miss penalty. So, there are 2 types of prefetching; one is hardware prefetching these are pattern predictor assisted and software prefetching in the compilers alright. So, before we even go into this particular this implementations there are few common questions. First one is would you like to prefetch into a dedicated prefetch buffer or directly to the cache alright. So, second question is of course, want to prefetch that is what this predict yourself this compiler is trying to figure out and where to fetch. So, let us take a first one what do you think? So, that you have a dedicated prefetch buffer on the cache compute the prefetch block. So, should I directly need to the cache what are the pros and cons?

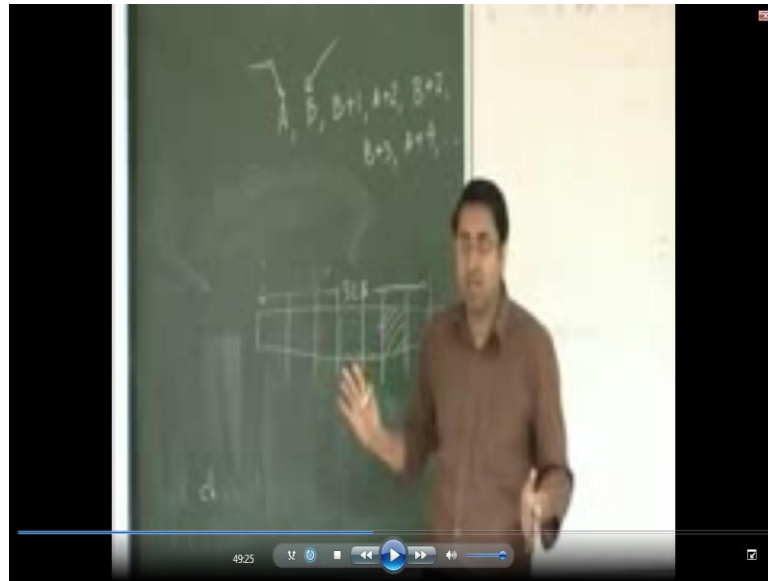
Student: When the con is.

Right. No I should do a parallel look up. Exactly so you have to take care of certain extra things. So, again certain things we cannot discuss in this course as to multiprocessors, but there are. So, whenever you put an extra structure there are certain extra things that you have to do which you which is better avoided. So, today all processors actually put the prefetchers directly into the caches they do not have anything. And as you module out that actually proves special on the accuracy of this particular question you better be correct otherwise you the cache bit time alright. So, these 2 are actually kind of linked these 2 questions that is what to prefetch and when to prefetch you may be 100 percent accurate but if you prefetch too early then you may be still your cache. Because there you may be evicting something out of the cache which is still used which is not good alright.

And if you are prefetching too late, then of course, the whole point of hiding physical, gone. So, this is a very difficult thing to tune when to prefetch exactly and we will find that usually the hardware prefetchers or the compiler to be conservative that is we will try to really hide a about of data. So, we will not try to hide the full latency by and you can applying prefetching to both instruction of data nothing stops you from doing that there are 2 types of prefetchers non-binding and binding data prefetch. So, non-binding prefetchers only prefetch a data block into the cache. Whereas, binding prefetches actually not only put the data in the cache which also copies certain loads into the register alright.



(Refer Slide Time: 46:24)



So, there essentially what it is doing is you might have a load instruction which looks like this with binding prefetch. This load instruction will be converted to prefetch instruction the data will be brought to the cache and the word will be taken out and put in the  $r_2$  alright. So, this instruction would be gone actually completely non-binding prefetches. This instruction will remain there only thing is that you will insert an extra instruction called prefetch which will prefetch for this particular cache. And the hope is that by the time the prefetch is completed the load is still not there alright. So, when the load comes up the data is already in the cache which contain the put in out to alright. So, that is the difference between binding and non-binding prefetchers.

So, binding data prefetch is not done in any processor again, because of things may need to multi processors somebody else may modify the data which may not be reflecting the register value. So, that we cause sort of lot of issues. So, what kind of prefetchers today's processor use? So, one of the most popular one is a is a stream prefetchers where essentially what you do is you divide your data address space you do small chunks alright or streams. And within each stream you try to find out if the address is fall in pattern or not at the typical pattern that you look for is a stride pattern that is whether the addresses that are that are that you are accessing from an arithmetic progression or not alright. So, it takes for longer if that is a case.

So, single stream stride prefetcher does not partition bigger at the specific streams you just have to larger stride. Whereas, multi stream stride prefetcher will actually partition the address space into multiple streams and from each stream which you try to monitor the arithmetic progressions right. Your single stream prefetcher will not be able to cache this one for example, we will get completely confused because this is essentially a mix up 2 until the progressions right. Your multi stream prefetcher which probably separate these 2 streams out starting with a and b, because they will probably belong to different portions of the address space and you learn both the progressions correctly.

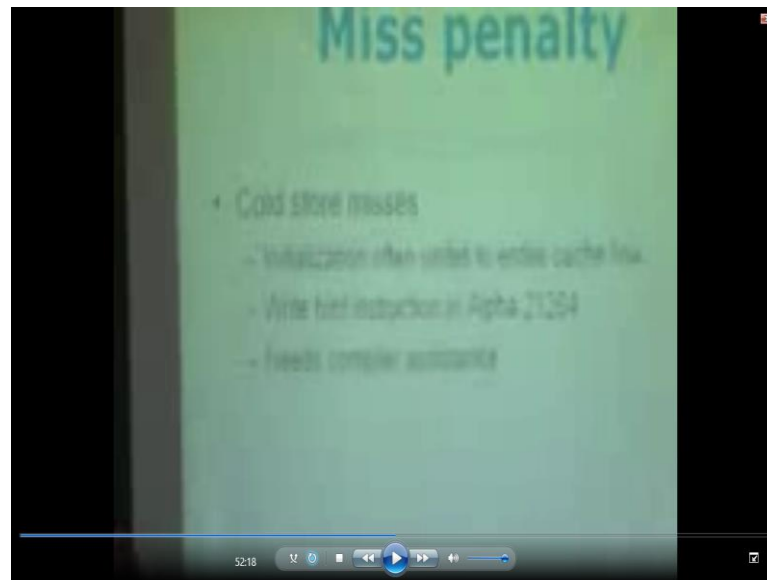
No it does not know no. So, what if your stream sizes such that a and b both fall in the same stream then you not be separate this 1. Yes that is true it does not know anything it just creates streams by assuring the, what can be the size that that today's prefetchers. The page it tries in page and the page is if a and b follow the same page then it will look like a like a single stream prefetcher actually you will not do to separate this. So, this is one of the simpler things that today's processors implement they will other sophisticated prefetchers. If you want to read up on that I can give you papers it is an it is an extremely well researched area with a huge body of literary.

So, hardware prefetching in general is harder than software prefetching, because of the danger of polluting the cache. Because here you do not have any vision of the program you were just seeing a stream of addresses that predict in the future. So, there is a very high chance of making mistakes here whereas, in software prefetch you can see the whole program compare a patterns to the accesses. So, software prefetching it supports from instructions if for example, the prefetch instructions that I shown there which was with non-blocking otherwise there is no point actually. So, this will be able to launch this prefetch and forget about this instruction completely alright. So, the processor can keep quiet.

So, what about prefetches on wrong path I too have a prefetch instruction we started on the path of a miss predicted branch what about these prefetches. Would I cancel them what should I do and what about prefetchers is that in a T L B misses it is possible right prefetching is basically about looking. So, read and running so, read So, that you access a new page for. So, in this case usually you continue the prefetch hoping that the other path will be taken very soon alright this one is a little question. Because T L B misses are costly and you are taking a T L B miss believing the speculation prefetching a

speculation. We predict an address which make wrong or which make correct that based on and believing that we need so much of performance over head may not make much sense actually. So, depending on the desire discard such prefetchers if you want continue in most cases these prefetchers will actually be prefetches the T L B misses.

(Refer Slide Time: 52:10)

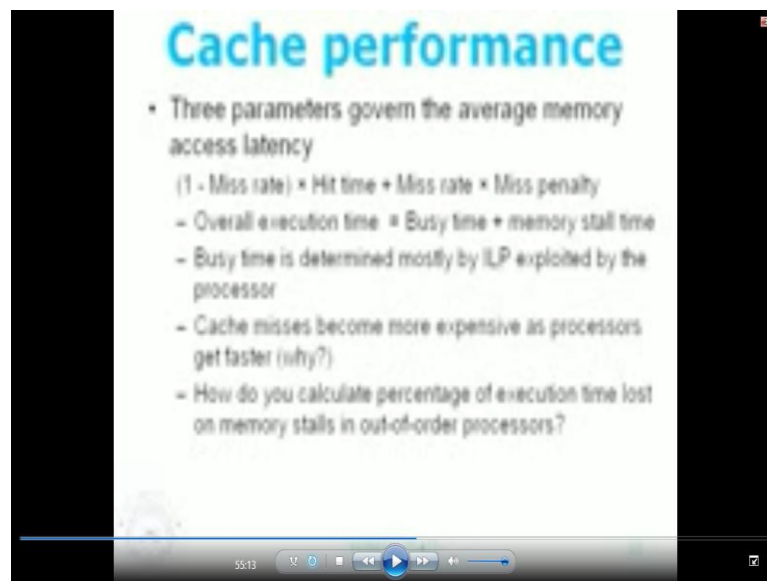


So, once point about miss penalty so often what you do this is very common with that is initialize the large data structure. Let us suppose we have array in a large array you initialize all elements of array to zero program. So, these are often called cold store misses. So, first time you will access it so you cannot avoid these misses it seems right. So, that is in the all stores will be storing everything with the value. So, if you observe what is really happen is happening is that you are writing to entire cache lines. It is not that you are writing to only a few words of the cache line entire cache line ok. So, then the question arises is when I take such a cache miss why should I bring the cache line for memory, because whatever is completely anyway. So, I should not bring it at all right I just allocate a block in my cache. And I just write the new value just say I am able to do it I need to go that yes this is what the pattern is.

So, for this purpose there are write hit instructions they are in 6 and also they are introduced first in alpha 21264. So, what the compiler does is whenever it catches such patterns it will actually generate the write hint instruction meaning that this entire cache block is going to. So, there whenever such a cache miss happens you do not go with the

memory you just allocate a block that is it. So, that that nullifies the entire miss penalty but it needs compiler assistance hardware cannot be. Yes so, usually these factors are called where we have very simple data structure an array. So, yes you can have yes in that case compiler do not do it, but of you are allocating you know that. Yes they will yes you just paid offsets cache may be allowing to allocate some data structure aligned yes otherwise. So, yes to state that what is of this particular in a program yes that is right.

(Refer Slide Time: 55:13)



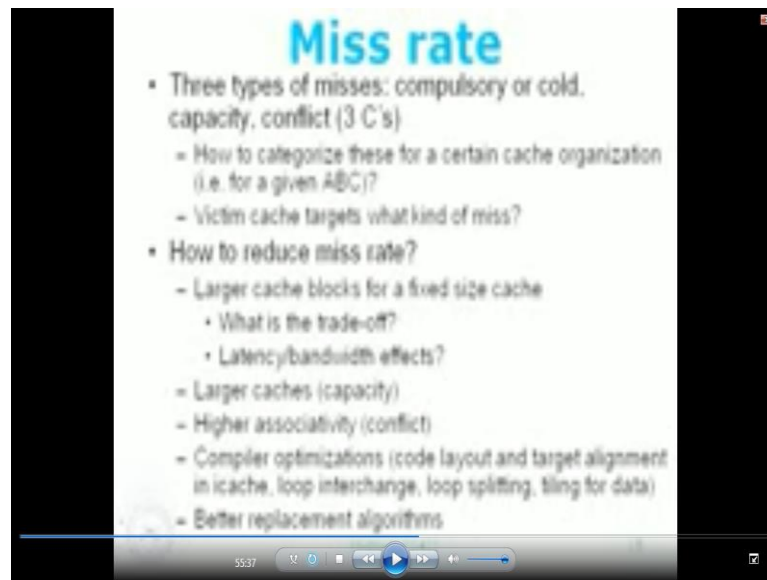
The video player displays a slide titled "Cache performance" in blue text. The slide content includes:

- Three parameters govern the average memory access latency
- $$(1 - \text{Miss rate}) \times \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$
- Overall execution time = Busy time + memory stall time
- Busy time is determined mostly by ILP exploited by the processor
- Cache misses become more expensive as processors get faster (why?)
- How do you calculate percentage of execution time lost on memory stalls in out-of-order processors?

The video player interface at the bottom shows a progress bar at 55:13 and standard playback controls.

So, just to remind you quickly what you are talking about this was the average memory access vacancy equation. And we are trying to look at how to reduce each of the there are 3 parameters in this and what we have already discussed how to reduce miss penalty several solutions for that? So, we will take a miss rate next.

(Refer Slide Time: 55:34)



**Miss rate**

- Three types of misses: compulsory or cold, capacity, conflict (3 C's)
  - How to categorize these for a certain cache organization (i.e. for a given ABC)?
  - Victim cache targets what kind of miss?
- How to reduce miss rate?
  - Larger cache blocks for a fixed size cache
    - What is the trade-off?
    - Latency/bandwidth effects?
  - Larger caches (capacity)
  - Higher associativity (conflict)
  - Compiler optimizations (code layout and target alignment in icache, loop interchange, loop splitting, tiling for data)
  - Better replacement algorithms

So, what is miss rate? It is basically total number of misses divided by total number accesses to a cache that is miss rate. So, before we go there we first needs to understand the reason for this why a particular access missed in the cache. So, for doing that there is a classification of misses between the 3 types of misses which are called compulsory or cold misses capacity misses and conflict misses. So, these are usually known as 3 c's misses alright. So, first question that arises is given the cache organization that is the cache. So, A B C of the cache what is this stands for a is the associativity b is the b is the block size and c is the capacity. So, if I give you the A B C of the cache that completely defined the cache architecture and suppose I give you a particular stream of addresses. And ask you to categorize the cache pieces into these 3 classes, how you do that right is the problem here.

So, I have told you the A B C of the cache and I will give you a string of addresses. So, you fit this string into the cache and some of things will hit some of things will miss. And I ask you to classify the misses into these 2 these 3 categories compulsory capacity and conflict. How to do that? Any suggestion, because once you know what cause the particular miss then you can think about the complicating particular miss. So, let us start with the first one that is the easiest from the code misses how many which misses what is the reputation of somebody? Yes. So, it says that it is an also called compulsory misses which means you cannot avoid so what are these misses exactly. So, whenever we access the cache line for the first compulsory miss which you cannot whatever you do. So, how

do you categorize the misses? How do you find out the compulsory misses given the A B C parameters and a stream addresses.

So, all you have to do is to find out the first time misses. So, how do you do that? What is efficient? so, what are the basic elements? So, you have to remember whenever whatever you have accessed in the past whenever you access something you have to search to the whatever history that maintained you figure out This is the this the first access or you have already seen this cache block right. If you have already seen this cache block it is not a compulsory miss we have not seen this cache block here is a compulsory bits. So, what is the efficient way of doing this? What kind of data structure you use. So, ready to maintain a history which should be efficiently. Well, the requirements you have to tell me data structure to maintain this history yes somebody here. Sorry Set in an abstract let me think what about implementation.

Student: Sorted array.

Sorted array can it be greater than that

Student: Sir binary search.

Binary searching yes. So, but you have to balance it right in L 2 yes anything else which can hashing right. So, what is your hashing? The address 9 right block address. So, we maintain the block address in the hash table and from the hash table I will find out that right. So, in general that that works much better than mine is search, because if it dams alright. Now, so whenever I will get an address I look up the hash table point out if it is already there if it is there then it is not a compulsory miss if it is not there and insert it in the hash table right. So, that takes care of my code is that clear? Presently how I count what about the next one? Let us take up one fifth is an miss how do you find conflict misses?

So, I already separated cold capacity plus conflict I have to separate the conflict pieces from that what is the definition of conflict misses or rather where can I where can I get rid of all conflict misses? Yes exactly when I so given a capacity of block size if I make the cache fully associative then you can say that for this particular capacity I have no conflict misses quietly do that right. So, I know the A B C parameters so all I have to do

is I design the fully associative cache pick the address scheme into this cache [capture] total number of misses.

So, what will be these misses misses code plus capacity right it was conflict miss in the zero I already have told misses So, I have capacity misses then what I do is I actually simulate this actual cache with A B C parameters. See if the save address to right from the total number of misses and this will be cold plus capacity plus conflict. I already have cold a capacity to get conflict misses clear everybody out of how many. So, that that tells you given a particular program I can find out how many misses of each category that program has. We have already talked about victim caches right what kind of misses out of these 3 what kind of misses doing is same if I have a victim cache why?

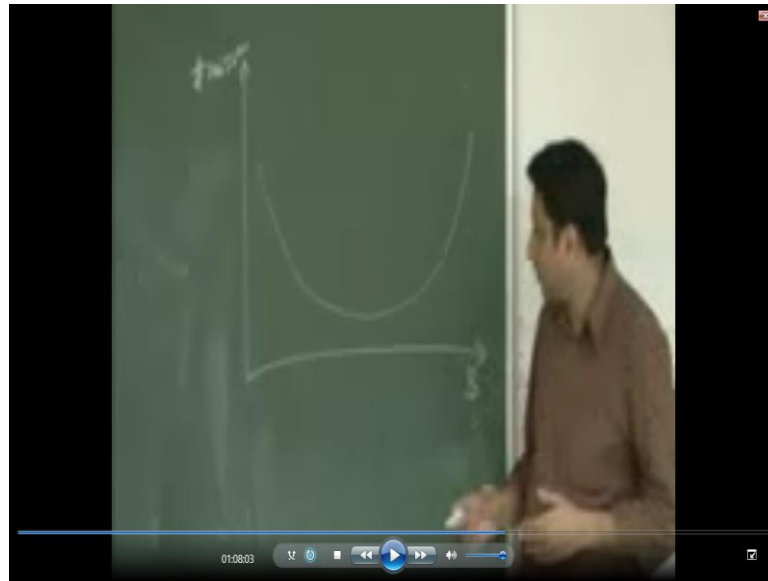
Student: It is fully associative.

So, victim cache effectively this associative so, that is why you save target misses with the victim cache after that.

Student: No

Why that is exactly what steps make right often categorize how to separate capacity conflict? So, in general if you just look at this it may look either way it is very difficult to separate about it. We have to do that in separate now the question is how to reduce miss rate right. So, the first solution is to have larger cache blocks for fixed size cache why does it reduce miss rate. So, essentially what I am saying is that I click my c fix then increase b. So, I can actually do the problem then I can keep my c and a fixed and increase b and reduce the number of sets alright. So, I will keep the capacity unchanged and it says that it reduces number of misses possibly why does it reduce number of bits by increasing the block size? So, if I increase block size what am I doing? I am doing it more data in 1 short right. If the application has good spatial locality they comparing let us say we are comparing a block size of x with the bock size of 2 x. The application has spatial value and break some out of data I would have taken 2 misses now I will this right. The problem is that this nicely does not hold forever, because now if we why can I make the either caching with block right it does not work why is that? So, if you plot the number of misses.

(Refer Slide Time: 01:06:26)



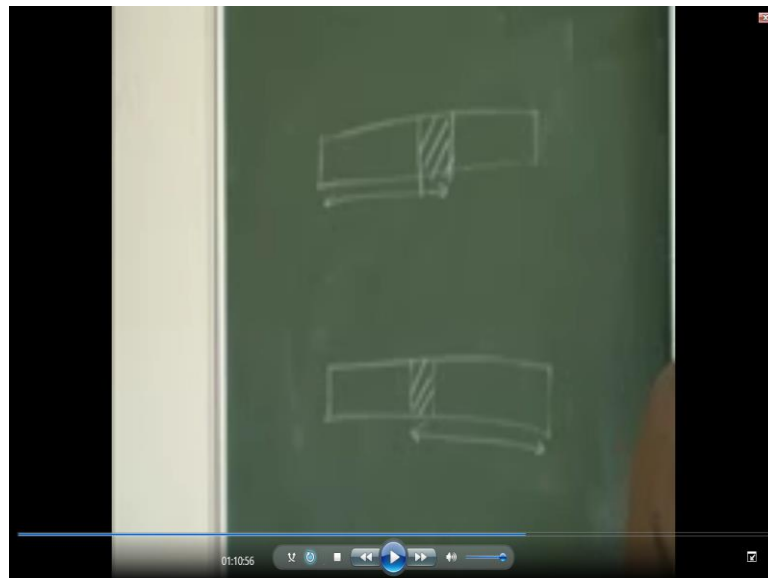
We plot size it will look like this why is that? So, what is happening exactly I have by feet I have a feet associative I have a fixed capacity I am increasing the block size. So, what is rating what is decreasing number of sets is going down? So, which means number of blocks mapping to bus set increases or decreases? Must increase right. So, beyond the certain point the conflict misses go up so much that you have to lose the advantage of having a spatial locality. You started losing that actually number of misses actually start increasing here is a increasing as you as you increase a block size alright. Also not all applications have conflict spatial locality there will be applications that load have very spatial locality. So, in those cases increase in the block size beyond the certain point will actually start because you will be breaking the useless data into the cache.

And plus there is a bandwidth latency of a cross because to bring it more more time that is what. So, we continue more bandwidth also it will take more time to satisfy 1 ways. So, that is the first solution that is increasing b increasing c is now obvious solution to reduce somewhere misses if you have a larger cache after certain point number of misses will go down. But as you increase the capacity the retire will be diminishing alright higher associativity which you increase a does not affect conflict miss and on addition to that there are compiler optimization to reduce number of misses. So, let us take each of these at a time the first one that is code lay out and target alignment instruction cache talks about reducing instruction cache miss. So, let me try to explain what this is?



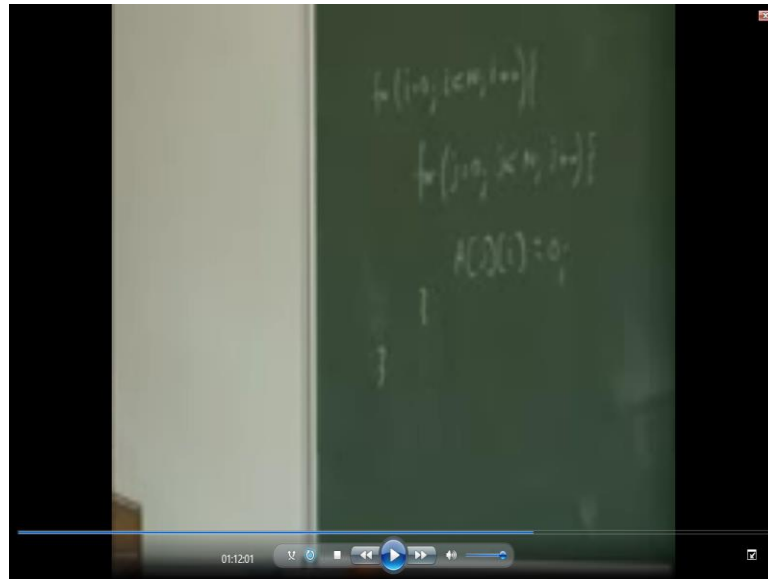
So, code layout essentially talks about reorganizing your code such that most frequently used instructions will be held in the cache alright so, most frequently used instructions contiguous. So, they can be handling the cache in a much easier way target alignment talks about the addresses are following problem. Suppose you have a instruction cache block there is a branch instruction somewhere in a middle that jumps to another instruction cache block somewhere in the middle alright. So, what I have to done you have wasted training part of this cache block and the reading part of this cache block. So, that essentially waste the cache base what target alignment tries to do is it always puts the target of a branch instruction and a head of the cache. So, that will have a complete cache block is likely to be used with the higher probability.

(Refer Slide Time: 01:10:20)



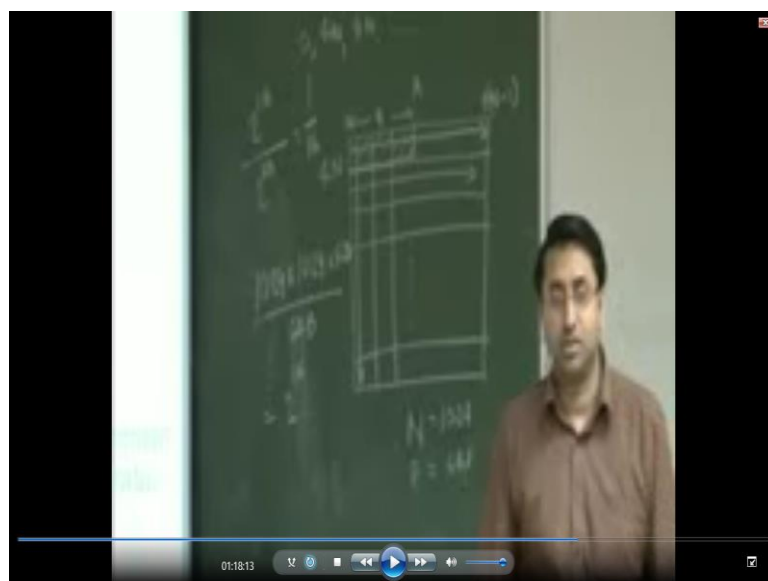
So, this is what instruction cache block this was the branch instruction and the target falls in another cache block somewhere here alright. So, this part of the cache block will be used and this part of the cache block will be used this is wasted this and this alright. So, how can you solve this? Well, the solution is that you can put branch instruction towards the tail of the cache block and you can put branch target towards head of the cache block. So, that will make sure that both the cache blocks are fully utilized alright. So, this is what the compiler can do, but not always it is possible is that clear? Alright.

(Refer Slide Time: 01:11:24)



So, this writes to only instructions data next take an example. So, loop interchange. So, I am initializing the 2 dimensional matrix right do you see a problem with this code once proper defined right. So, it is going to be of extremely so the point is that if I assume that is c code the c compiler allocates address in the row of cache what does mean? That means may fix a we will get addresses contiguous addresses for the first row. And once the first row is done the second row will start that is how the matrix will be laid out in memory alright.

(Refer Slide Time: 01:12:57)



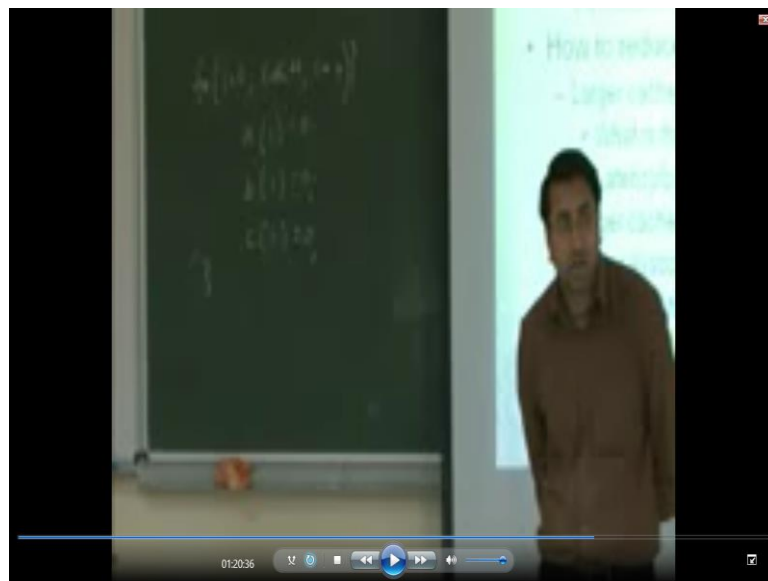
So, now if you look at this one what am I doing? This is my matrix A. So, let us suppose this is address zero alright and if I assume that my matrix is an integer matrix. So, this is going to be what this more times  $n$  minus 1 alright and here we will start from 4  $m$  etcetera alright. So, this will be all contiguous right 4 bytes each. Now, if you look at the code what does it do? It close like this. So, it access in the address zero then 4  $n$  then eight  $n$  and so on, but somehow  $n$  is very large let us say let us look some. So, the way is equal to 1024 let us suppose my cache block size is 64 bytes. So, what did we say about this addresses each cache blocks they are falling will they be falling on the same cache block or different cache blocks these addresses. So, they will be falling on different cache blocks right. So, this is this is 2096 this is 838 1092 whereas, my cache block size is 64 bytes alright. So, they will all go to different cache blocks they will be all cache misses alright the question is when I come back. So, this one is the first element of my cache block the first 1.

So, 1 cache block is like this it will contain eight matrix this. So, I have done with the first problem then I come back to the second problem. So, the question is this cache block scheme is my cache if it is the then I will enjoy a cache hit access in the second column. Now, for this cache block to be in my cache what should be the cache size 1024 rows 1024 cache blocks each cache block is 64 bytes right. So, my if a cache capacity is 64 kilo byte then I should be fine right 6 what am I assuming that by these cache blocks will actually map to different sets or at least different ways of the cache alright. So, you can easily find out that even with the very large cache that is not going to be a, because these cache block address are very far apart they will actually very like to that they will all map to the same set. So, this cache block will get evicted by the time I will come back. So, you do not have pretty much a 100 percent register. So, you can tries out you can go back and mark this program and measure the, it is make  $n$  very large. So, that you do not have you minimize the 1024 is a smaller number make it a very large value right.

So, what this one is talking about is interchange the loops. So, essentially what I will do is I put the  $j$  loop outside and take the  $i$  loop inside. So, that is called loop interchange and now my access pattern is change now, I will go like this the sequential access what will be my cache hit rate. So, I will tell you what will be a cache hit rate. So, there are 1024 times to the total size of 1024 5 1024 times 4 bytes alright divide that by cache block size which is 64 bytes. So, this where the cache lines I have exactly 1 miss for

cache line right that is a code miss. So, how much does it come to 2 to the power 16 right 2 to the power 16 misses out of how many accesses I have 1024 times 1024 access in the matrix right. So, 2 to the power 16 divided by 2 to the power 21 over 16 is going to be my miss rate So, there will be 16 access I will miss 1 right sorry this was 16 elements for caches ok. So, that is loop interchange actually you can try some you can go back home run the 2 misses of code without loop interchange or with loop interchange make n very large we will see a in performance second 1 is loop splitting. So, let us take 1 more example it is most slightly that we find this code also.

(Refer Slide Time: 01:18:58)



So, often we tend to club initialization of matrices together right in a single loop which they have same size. So, what we will think about? This is the code it looks perfectly fine right unfortunately can anybody guess what the problem is? As again suggests this in 3 different loops as opposed to what is the problem it is extremely common. If any do this you guys have problem why is that problem what happens if A B and C are very large? What will happen suppose a size of a is a multiple of you know cache capacity size of p is a multiple of cache capacity size of c is also multiple of cache capacity. We have a 2 way set associative cache what will happen?

Student: All 3 will map to the same set.

All 3 will map to the same set then.

Student: Then a and b will fill that.

Which one will be evicted if I assume regard.

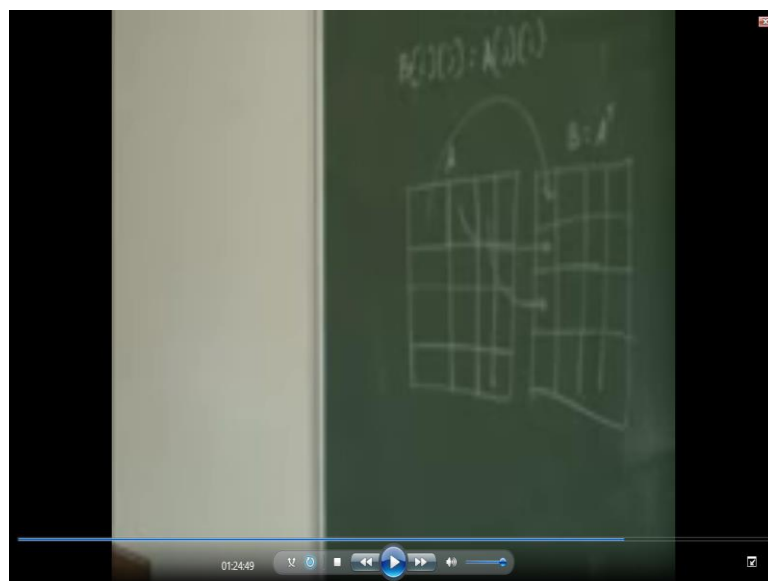
Student: A.

A; so, after c the access of a will come which 1 will evict after c the access of a will come right it is a loop. They may be in b right access of b will come next which one will evict.

Student: C.

C; how do we reset 100 percent right. So, if A B and C are large it is that you do not do this it might seem that you are actually saving space in your program. It might seem that you have a compact code but you are actually getting into trouble code is going to get a very long time alright the loops everybody will be happy. So, that is what the this transformation is loop speed alright it is the very simple example of loop speeding there are many other campaigning examples. There is a 1 common mistake that you can do that is why I wanted to highlight this and the last 1 is tiling for data. So, here what you do is there is a slightly more invoked transformation that compilers applied. So, you must take a very simple example.

(Refer Slide Time: 01:22:50)

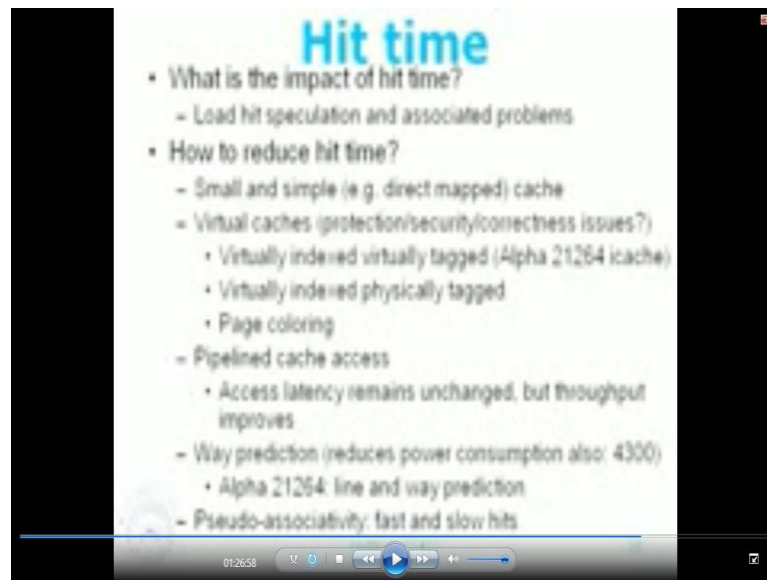


Suppose I want to transpose a matrix alright. So, let us suppose this is my source matrix and I want to transpose this into matrix b. So, typically in a nested for loop I will basically say  $b[j][i] = a[i][j]$  alright. So, b is equal to a transpose so how and when you write the loops whether it is i then j or this j then i one of the matrices would be traversed in the b or a and you already see the problem it can have a very high number of misses. The question how I really solve this problem loop interchange will not solve this problem. So, you essentially what you do is you do a tile transpose you divide the matrices into tiles draw the matrices. These are called tiles sorry and what you do is you make the tiles small enough. So, that those of these tiles can pick in the cache alright without conflicting.

So, then what you do is you transpose this tile into this tile these into this tile these into this tile and so on. And do not be sad once I am done with the tile I never come back to it actually. So, if I can size the tiles so that they fit in the cache I can enjoy maximum hit rate and then throw away the tile and move on to a new tile ok. So, this will give you that hit rate probably the hit rate  $\frac{1}{60}$   $\frac{1}{60}$  in this rate, because you will get completely lose of this. So, this is what loop tile sorry they not this problem time you can you can apply time into other constructs also any question in this. So, of course, the examples that we talked about like the interchange and all by looking at the code is obvious that if I interchange the code the program will see little tile in any cases may not be. So, there is a fairly involved transformation thoroughly behind it which can verify the of interchange began into splitting began into time alright.

So, this was we cannot talk about the and the last option for reducing this hit is to come up with better replacement algorithms. So, this one we have already talked about goal of the replacement algorithm is to minimize the number of misses alright at the simple observation here is that you should not be replacing blocks which are alright. So, and the optimal policy is that actually based on that which essentially says that given up given the number of choices for replacement you should replace the block which is the hardest reuse. So, the gentleman called came up with this algorithm in nineteen sixty six from IBM. So, that is also called the longest majority. So, it looks at the next reuse distance each of the blocks and replaces the 1 which has the largest uses. So, you must have seen this algorithm with your operating systems optimal page replacement policy alright.

(Refer Slide Time: 01:26:58)

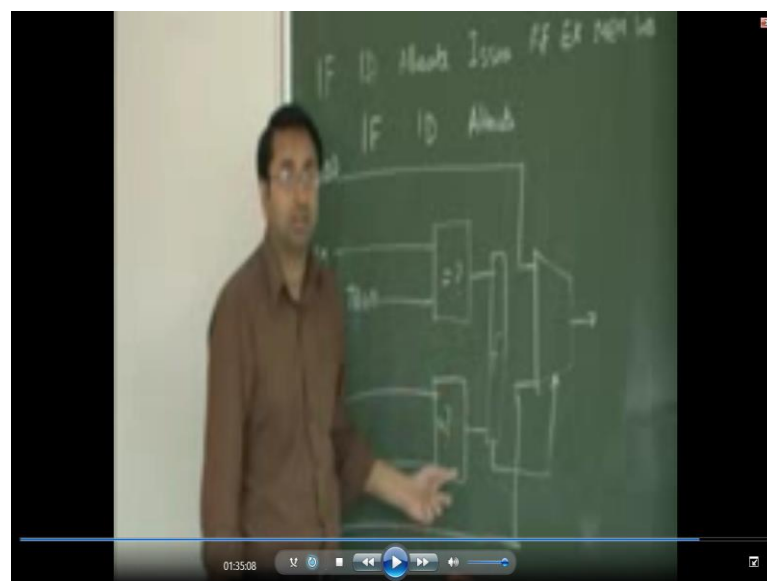


**Hit time**

- What is the impact of hit time?
  - Load hit speculation and associated problems
- How to reduce hit time?
  - Small and simple (e.g. direct mapped) cache
  - Virtual caches (protection/security/correctness issues?)
    - Virtually indexed virtually tagged (Alpha 21264 icache)
    - Virtually indexed physically tagged
    - Page coloring
  - Pipelined cache access
    - Access latency remains unchanged, but throughput improves
  - Way prediction (reduces power consumption also: 4300)
    - Alpha 21264: line and way prediction
  - Pseudo-associativity: fast and slow hits

So, now we are left with the last parameter that is hit time. So, one question is why is this important. So, if you are having one obvious reason why is important this, because of this equation right if I have a larger hit time I will have a larger average accessing alright. So, what is hit time? Hit time is the time taken to look up the cache and get the data out of the cache alright that is the hit time. So, there are other impacts as well which are not so obvious. So, let us take a look at them. So, let us go back to our pipeline that we discussed.

(Refer Slide Time: 01:27:54)



The image shows a man standing in front of a chalkboard. The chalkboard contains a diagram of a processor pipeline with stages labeled IF, ID, Exec, Issn, IF, Ex, Mem, and Wb. There are also labels for 'IF ID Alloc' and 'IF ID Alloc' on the left side. The diagram shows a flow from left to right through various blocks and registers.

So, we have a fetch decode allocate issue register file execute memory write back right. So, this the pipeline that you discussed. So, if you look at the load instruction right it computes the address here. And these are the pipelines tail is I am I am showing it as just 1 stage here is essentially your cache look up stages alright. So, by increasing hit time you would actually introduce more pipe stages here. So, let us first talk about the single stage if it is my hit time is just 1 cycle alright. So, now suppose we have an instruction that we get more instruction. So, let us see what happens to this guy. So, this instruction fetches here it here allocate the question is when does it issue if I issue it here it will not get the results in time right. We have already discussed this we have to bit a load inter lock.

Now, what happens if I increase the hit time further if I have a 2 cycle hit time what happens I cannot issue it clear the dependent I cannot even issue it the cycle. So, that is not so obvious in fact to the hit time it actually starts effecting your pipeline efficiency it introduces more and more inter lock sizes alright. So, todays processor what they do is that they actually speculate that the load is going to hit the cache. So, the problem is that until and unless I have loop the load has looked up the cache. I cannot really say that this instruction is going to get the data in time even after the interlock sorts so clearly that as I said I cannot issue it here. Because even if the load hits in the cache it would not get the data in time because I will issue it here it will make the register file the execution will come here alright. So, I have to put the execution here now suppose that I out put an interlock bubble here alright and issue it here.

And now when I am issuing this instruction I am actually assuming something what is that? I am assuming that the load is actually going to hit in the cache then only this instruction can get the value here. So, it is a speculation under which I am actually issuing this instruction if I did not speculate I do not actually have to wait until this cycle to issue the instruction I would not have issued actually until this cycle. So, today's processor I can routinely use this particular type of speculation and this speculation make sense. Because the cache in is very high in most cases it is actually more than ninety percent in most cases. So, I am going to be correct with the probability more than 0.9 if I do this only ten percent of time are going to be wrong in which case I will have to retry this instruction. And again issue it later when the load this actually deserves and as you



can guess there are predictors which can improve this speculation bit further, because see it is just like a branch predictor.

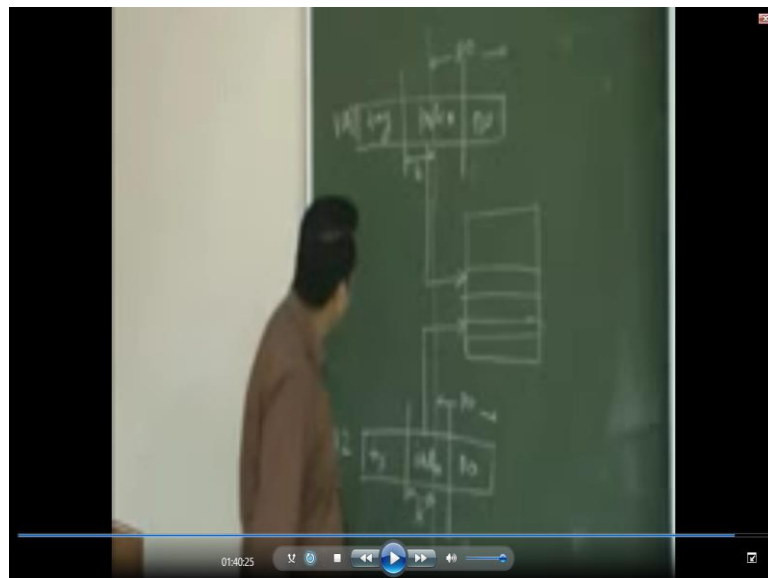
So, indicate this particular point you are asking the following question. If I issue this instruction is it going to get the data in time or not? It is just a binary question right. So, it is just like a branch predictor and you can learn that pattern, because you can ask the that looking at the history of this load tell me whether it is likely to hit or miss in the cache and that is high predictable hit actually. So, that is the that is the load hit speculation and the backs often used to so, to fetch the impact of block hit time alright. So, how to reduce hit time? So obvious solution is that you should have a small and a simple cache like a direct time cache that is very simple introducing associativity normally increases in time. Why is that a 2 way set associative cache is slower than the map cache why where exactly do we lose the time. So, what happens the direct time cache I will just take the address calculate the index and look up the corresponding set. And get the data that is it right what happens in 2 way cache what do I used to do? Yes we have to do that you take the index look out the cache into a tag comparison.

And if you process then you will take the data otherwise not alright what happens in 2 way cache who compares this and after that. So, there is one more thing that you have to do yes exactly there is a multiplexing that is involved here so, this is a comparison. So, this is coming from the tag this is tag zero this is tag 1 we have data zero and data 1 and you begin the multiplexer. And the selection of that is going to a function of these 2 right as you introduce more associativity this multiplexer is going to be in wider. And this is in the critical path you cannot get a data how you have a multiplexer and gradually your latency will increase because of this multiplexer, because these comparisons are in all in parallel. So, they would not really hit into a latency what needs to the latency is this 1 the multiplexer gets wider and wider and you start losing latency. so, that is why smaller simple caches are fast alright the second solution is virtual caches.

So, we talked about this a little bit in connectional theory So, virtually indexed virtually tagged caches have faster than virtually indexed physically tagged cache why is that So, first 1 derived six tag and index both from virtual index and the second 1 derives it is index from virtual address tagged from the physical address So, why are they faster. So, compare to physically indexed physically tagged cache this is faster why is that because you will be look up before you can block the cache alright. And this one does not even

need a T L B the entire thing is in the virtual space alright. So, that is you have to complete that is what 6 4 instruction cache can actually complete alright. However, here so first of all a virtual if you have a virtually indexed virtually tagged cache. Then since your by pass completely you have to be careful about permissions and security issues. Because the T L B when the T L B is looked up it also provides the permission hits over your round to access this T L B now by passing that. So, you have to be careful about that you might want to extend your cache tags with this particular case ok. Now with virtual index physically tagged cache there is some other problem. So, let us try to look at that.

(Refer Slide Time: 01:37:35)



So, what we are doing here is that we have the virtual address block offset index and tag your taking this index and you pick up the cache right that is the virtual index cache alright. Now, what if there are 2 processes that are shared in a particular page let us say alright. And so these 2 pages will have the same physical address, because you are sharing it, but they will be have to differ in virtual addresses 2 different virtual page belonging to different processes that are shared in a physical page alright. So, 1 process access it near 1 another 1 is near 2 so clearly they will access to different cache blocks to be the cache alright. But there are supposed to be pointing to a same physical page because they are sharing.

So, the problem is that 1 process may modify this particular cache block other 1 may read a value from this cache block. So, this is called the synonym problem because these 2 are supposed to be synonyms of each other, but that in varieties all maintained. So, this problem arises only virtual index caches. So, how do you solve this problems? How to make sure that these 2 virtual addresses? Actually have the same index bits. So, essentially the solution is that whenever 2 processes want to share a page their virtual addresses should be hit in such a way by the operating system. Such that their index hits are exactly same then they will map to the same cache block index and that will.

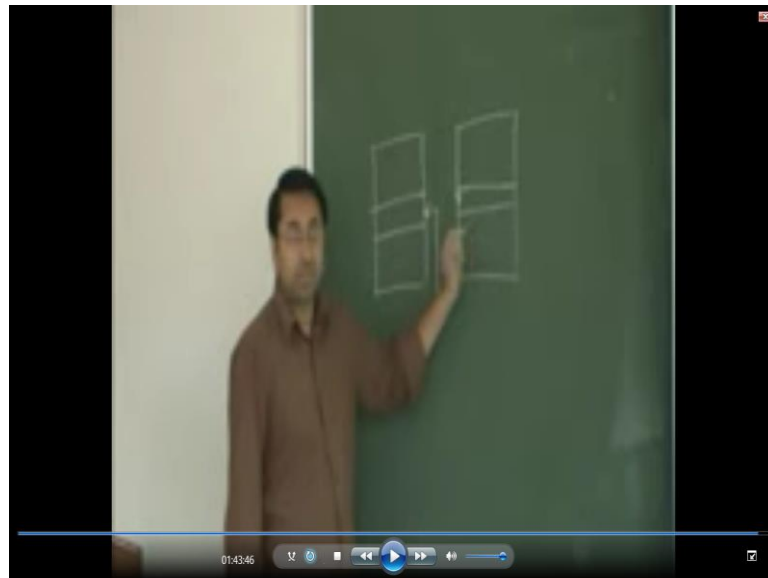
So, how to make that pattern? So, if you look at the way the translation is done. Let us suppose this is my page offset and that has to be identical in both say that the page offset does not change in the translation. So, problem is with these bits way to make sure that these bits are identical here you are done. So, depending on your cache architecture and depending on your virtual memory architecture virtual address space size number of these bits may vary alright. So, let us suppose that there are  $k$  such bits. So, what the operating system does is that it defines the entire version address space into  $2^k$  base right. And whenever 2 processes request 2 virtual pages to be shared the operating system will keep these 2 virtual pages from the same data alright.

So, that will make sure that this  $k$  bits are exactly same and these are called colors this  $2^k$  to the  $k$  colors are assigned to the virtual address space. And the pages are picked in the same colour this particular scheme is called page colour let us observe synonym problems. Third solution for reducing hit time is pipelined cache access where access latency still remains unchanged but the throughput improves pipeline the cache access. So, that you can every cycle you can feed something to the cache although the final data may come out much later that you have 2 number of accesses that you will finish in time another solution is wave prediction. So, this is this 1 used it needs 40 3 100 and alpha 2 2 100 6 4.

So, essentially here the idea is that the cache prime becomes so high that you cannot accommodate it using your cycle time. So, what you do is you make a prediction you ask a predictor to tell you tell me which way should have the data which should have actually looking on the ways alright. So, that is the way prediction 2 1 to 6 4 actually which said that I am not going to compute the index from the address I am going to ask the predictor tell me the line which set will have the data alright. So, you access the data

in the background you actually keep doing the traditional thing and verify. Of course, your prediction will go down sometimes in which case you will actually use the final data came out of the actual access another solution for reducing hit time is pseudo associativity. So, let us try to understand what this is?

(Refer Slide Time: 01:42:45)



Say there are 2 way cache usually what we do? We take the index look up both the ways take the tags from both the ways make 2 comparisons and decide based on that keep it up. So, as we said just now that, because of the multiplexing between the ways it may be slower than direct map cache access. So, what pseudo associativity does is we actually does not do this it has a 2 way cache, but accesses the accesses 1 way first if you hit you are done. You have done a cache look up at the speed of a direct map cache essentially if you miss then you look up the other way and if you hit in this way what you will do is that you swoked these 2 lines. Because every time you look up this way first alright So, this is called pseudo associativity on ((Refer Time: 1:43:53)) it increases pseudo associativity yes. So, here 2 hits now one is a fast hit another one is slow hit.