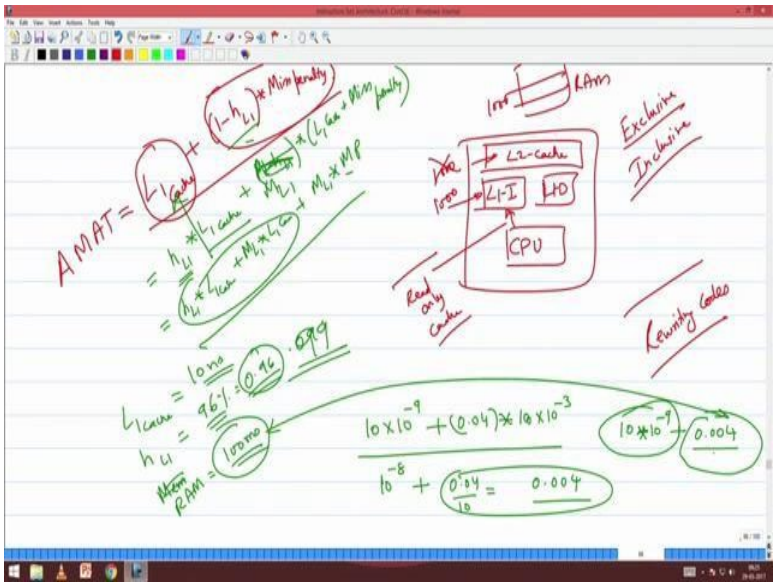
Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 36
Cache, Degree of Multiprogramming

(Refer Slide Time: 00:19)



As of now our understanding is there is a CPU, and there is split cache. So, L 1 I and L 1 D and then there could be a unified cache which we call it as the L 2 cache. So, structural the way why we had a split cache was that there could be a structural hazard and if there is no split cache. And so for that reason we just basically made this as a I and this as a D. And from a security point of view, please also note that this I cache can be made in such a way that from the CPU point of view it can be only a read only cache, yes, they are going to only read the instructions. CPU is not going to unless we have a notion of what we call as rewriting codes. These are concept where when the program is in execution it will decide on what it itself will generate a code and it goes and executes or some interesting problems that we can solve using rewriting codes. Just have a look at it is more of a popular principle of programming languages type of

concept, but just I am giving you a small pointer to that.

So, unless it is a type of a rewriting code right you have only basically going to read some of the instruction cache. So, from the CPU point of view, I can make the instruction

cache read only. Well, there will be updated. So, now this the set. So, let us do some small numerical analysis, let just say that. So, what is AMAT, what is AMAT - average memory access time is what L 1 cache access plus 1 minus hit at L 1 hit ratio at L 1 into miss penalty correct. We derived this, you remember, how did we derive this Rahul?

Student: Probability.

What probability? Hit ratio into L 1 cache time plus miss. So, for all h L 1 is the hit ratio into for all those fraction of those instruction will take L 1 cache time access time. The remaining will take L 1 cache plus miss penalty. Miss penalty is the time required to access anything beyond L 1. Now, this basically gives you h of L 1 into L 1 cache or we can have it as miss ratio M of L 1 plus m of L 1 into L 1 cache plus M of L 1 into miss penalty what are M P. So, this actually becomes L one cache; well this becomes 1 minus h L 1. So, this basically gives us a recurrence equation. So, I can now go L 2 cache L 3 cache, L 4 cache and so on.

Now, let us say L 1 cache time can you guess. So, I will say 10 nanoseconds and hit radio let us say 96 percent or 0.96 even this not good right, in real sense. Now, memory access time memory actually RAM, guess.

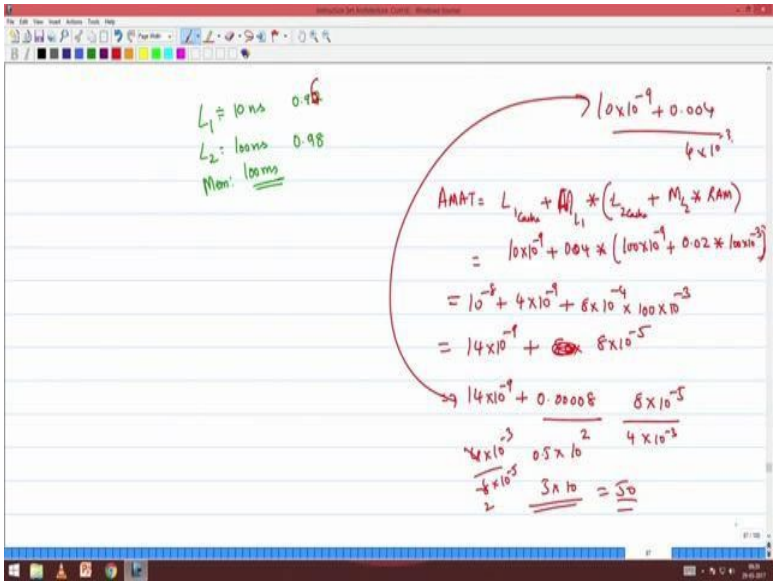
Student: Milliseconds.

Milliseconds [FL] milliseconds say 100 milliseconds. So, the average memory access time would be 10 into 10 let us make everything as 10 into 10 power minus 9 plus 0.04 100 into 10 power minus 3. So, this will be 10 power minus 8 plus 100 is 10 square right by 10, 0.04 by 10, which is something like 0.004 correct, this one around. So, I can still have it as 10 plus 10 into 10 power 9 plus 0.004, so which is dominant here this still dominant you are getting this, even though I have this 100 milliseconds and this ten nanoseconds please note that this still very dominant. Your 0.004 comes in whatever you say 4 milliseconds, your memory access is everything is fresh from memory it is going to be 100 milliseconds per fetch. But since I have cache I have reduced it from 100 milliseconds to some 0.004 that is 4 milliseconds I have got 1 by 25th, speed up this 25 times right by Amdahl’s law, you all remembered Amdahl’s law right. Speed up now becomes I thought I should ask one question Amdahl’s forgot, end some I will ask ok, so 0.004.

So, what does this teach you this teaches as that in spite of having 96 percent, you know cache rate, we are not able to actually you know we are not even closed to the cache access time in terms of average memory access time. So, something like 0.99 sorry or 99 percent is what that could be you know giving us some amount of relief. So, however, good your cache may be right cache has to be very good your replacement strategy plays a very big role. Towards the end of this course last week, I will teach you about you know this cache replacement policies like how we can implement hardware for cache replacement is slightly very advanced and we will basically take it up at that point of time, if we have time we will do that.

But whatever good you are you know cache thing would be right your I have to have a very good hit ratio and in spite of having a very good ratio hit ratio memory still holds between these two components memory still holds the major factor. So, this in it is in the excel sheet this 10 power 10 into minus 9 will never even figure, there will be someone very far off right in Pondicherry. So, we cannot even see that one here, so that is the basic thing and that is why and these are somewhat realistic figures. I am not saying these are the realistic figures, but you can work out you can go and look at Wikipedia and it will give you how that cache speed has increased, how the memory speed has increased, but overall this is something that we could see. Now, the way by which I could reduce this 0.004 still better is to have the L 2 cache and that is the reason why we going for the L 2 cache.

(Refer Slide Time: 08:31)



So, in the context of L 2 cache, what will happen. Now, let us go to the L 1 plus L 2 cache now let us see. So, let us say L 1 is 10 nano seconds and the some 0.97 is hit ratio now L 2 is larger in size and L 2 is a unified cache. What do you mean by unified cache you do not distinguish between instruction and data. So, it is a large cache. So, let us say it is going to say some something like 100 nano seconds and heat; its hit ratio is some 0.98 it will be certainly more than this. Then the of course, memory now let us say this going to be same 100 milliseconds right. Our earlier thing was 10 into 10 power minus 9 plus 0.004 was our earlier thing. I am using the same L 1, same memory, but I am just introducing an L 2 cache.

Now, what will be the equation for this AMAT for this would be L 1 cache plus L 1 miss ratio miss of L 1 into miss penalty would be what the same thing L 2 cache plus miss of L 2 into RAM access time. These are recurrence equation right, these are same recurrence equation instead of miss penalty I say I will access L 2 plus miss of L 2 into whatever RAM. Do you follow how I am getting this equation? I just explained in the previous class also. Just substitute here alone would be 10 nanoseconds 10 into 10 power minus 9 plus miss ratio is 0.03 I just put 0.968 plus 10, I will not touch anything 0.04 into 100 into 10 power minus 9 plus miss of that is 0.02 into 100 into 10 power minus 3 this is the thing.

So, let us see this 10 power minus 8 plus 100 into 0.04 is 4, 4 into 10 power minus 9 plus 0.04 into 0.02 is 4, 4 into 2 - 8 into 10 power minus 4 into 100, 100 into 10 power minus

3. So, this 14 into 10 power minus 9 plus 8 into so this 100 is 10 power minus 10 power 2, so 10 power minus 1 8 into 10 power minus 5. So, you get fourteen into 10 power minus 9 plus 0.00008. So, let us say these two are almost comparable, but what would have happened is this 0.004 actually became some thousand times faster. So, this 4 into

10 power minus 3, and I have something like 8 into 10 power minus 5.

So, I divide this I should divide other way right 4 into 10 power minus 3 by 10. So, this goes 2. So, this goes 10 squared 0.5 into 10 squared are 5 into 10 50 times faster, 50 times faster. So, if it takes imagine right a program taking 50 hours will now take one hour right that is the way you should look at it. What it means by 50 times faster is sorry suppose the unit is 1 hour then did the program take 50 hours, it will take 1 hour. So, observe and if the program is actually memory centric then that is the biggest advantage you get, yeah.

So, what I try to do here is just using some numerical values I just gave you some insight into how cache can basically improve performance, but still memory holds the final world. Out of these two components that you see in the recurrence equation where in one component basically talks about L 1 cache and the another talks about the thing that I miss something in the L 1 cache. Now, between these now you see that this the way we handle this. So, still the memory will have a much higher priority. Now, there is a lot of study gone into architecture books right saying how to improve L 1 cache, how to improve this blah, blah, blah right. I do not find that has an interesting issue at all because finally, memory matters if I have this faster memory right if I have a faster RAM and I have faster interface, this minus Q. So, why should I keep say I will improve. So, how will I improve hit ratio, I will try to keep a big block, larger block, I will try to make my spatial locality and temporal locality more prevalent by having larger blocks, but then that will have a say that will always have a say in when you want to replace how much you need to replace.

So, today that type of an analysis sort of not necessary for an architect, it is my opinion today. So, you go and put as much as cache you want or you can and that is better right. And beyond some point you will not see that cache is going to improve your performance because at some point if you make the if you keep on increasing the cache for smaller programs right for reasonable size program your entire space at any point of time that locality of reference will be there in the cache. And it really does not matter whether you know all the other strategies that we are talking off, all the graphs all the papers that we study these are all useless in the context. So, today I have lot of real estate I do not know what to do with that real estate, you are going seven nanometer, ten nanometer. So, I can put lot of cache try to populate it as much cache, but beyond some point this cache also becomes useless.

And one other thing is that cache is a single block now what would have happened is that I cannot go and cut power to a cache. So, then the people try to look at banking. So, the cache itself will be put as several banks and each bank will have its own power ray. So, if one bank is not utilised because memory consumes more power than the normal gates. So, if I am not using one bank, I can go and basically cut the power to that bank these are as simpler as difficult that is.

So, now let us now start talking about based on this cache hierarchy now I can extend it to L 3, L 4 till l n. So, based on this cache hierarchy how are cache policies classified now already we saw a write back cache with a write through cache right you right, you still remember that write back cache is that I write to L 1, but memory will be different. So, that will be this cache coherency issue. The latest value will be always in the cache not in the memory. Write through cache is when I write something I will also write it there, so there will be no memory coherence, and there are certain classes of problems which are more read only in nature that can be memory intensive, but there are more read in nature which will actually benefit through write through cache, so that is one thing.

And the that the point is that if you have minimum number of write through if you have minimum number of writes whenever I try to replace right I will try to not write not back. So, if you have a write through policy then I can go and throw off any thing at any point of time. So, you save some time when you are trying to replace a cache block. So, this is some understanding.

Now let us look at two types of cache based on this L 1 and L 2 or multi level cache. The two types are basically exclusive cache and inclusive cache. What is this exclusive cache? Whatever I store in L 1 will not be there in L 2. So, if there is a location thousand this the RAM and let me say I am bringing a location thousand here, this thousand will be first fresh to L 2, then it will be fetch to say let it be an instruction it will be fetch to L 1 I when it is fetch to L 1 I this removed. So, a particular location, the data corresponding data or instruction corresponding to a particular location will be in exactly one of the caches in the hierarchy. So that is basically called the this type of a cache organisation is called exclusive cache. By having an exclusive cache, what do you gain?

Student: Hit ratio rate.

What iteration?

Student: Hit ratio.

How do you say hit ratio increase in such exclusive cache? Hit ratio is basically a trace parameter it is not where do you store if I am accessing something again and again it will be in L 1 cache ratio whether that will be contained in L 1 L 2 cache are not is the story. If it is also going to be in L 2 cache then it is a inclusive cache; it is just in the L 1 cache

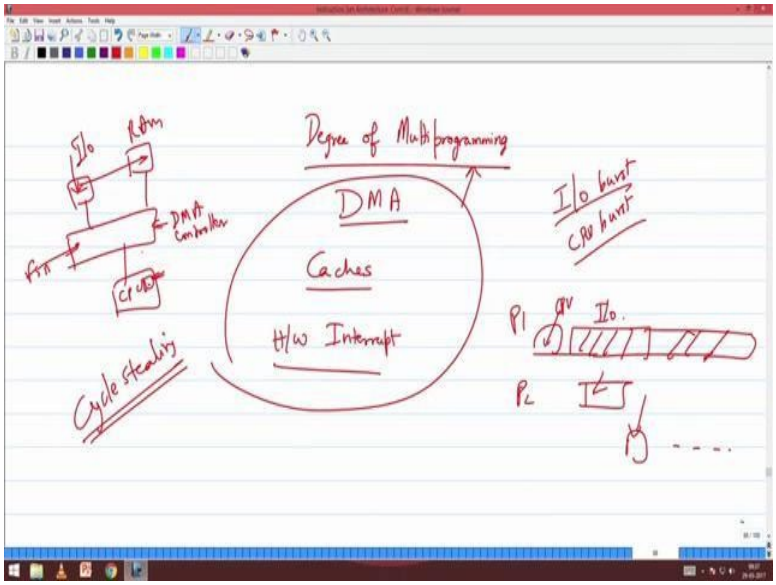
then it is just. So, what are the advantages of exclusive cache, the main thing is that the coherency problem is just restricted to you know the one cache and the memory. In the L 2 cache, the coherency problem can be multi facet. What is there in L 1. So, thousand will be there in L 1 thousand will also be there in L 2 thousand will also be there in memory. Memory will store 100, L 2 will store some 75, and L 1 will store some 125. So, there could be three different values for the same value. So, what will happen is when I am evicting I have to take care that. So, when I evict from L 1 I go to L 2 and then correct it there, and I when I evict from L 2, I go to the memory and then correct it there.

So, the cache coherency problem is now a large one. So, implementation wise right from the implementation point of view, I could have these thousand in multiple memory location and I have to basically handle that at a very stage. So, this one very big problem when we look at you know this inclusive cache. Wherein whatever there is in L 1 will also be there in L 2. So, the content of L 1 is a subset of L 2, content of L k is a subset of L k plus 1, but here the intersection of L 1 and L 2 is 5 is a empty set or L k and anything else is an empty set. So, if just a issue your cache coherency issue is just restricted to one cache. But the bigger problem of having an inclusive cache is that when I want to brings somebody in I do not so in the case of an inclusive cache when I am trying to say let us say L 3, L 2 and L 1 if I want to reach somebody first I will make an entry in L 2 then I will make an L 2 then I will make an entry in L 1. So, three entries have to be made when I am trying to bring, but in its a case of an exclusive cache I have to make only one entry.

But the point is when I am trying to replace somebody in L 1, now I have to see I have take into L 2 and bring somebody from L 2. So, when I want to bring somebody from memory into the L 1 cache, first I have to bring into I can directly come to L 1; from L 1, I have to displace somebody and put that into L 2. And if L 2 is full, I have to displace somebody from L 2 and put it in L 3; and somebody is full in L 3, then I have to move into the memory you are getting this. So, as I keep on mentioning right I think I have mentioned several times computer science is communists science in sense that if you gain something you will lose something. So, these are things. So, so for me a cache coherency issue becomes very easy to handle in the case of a exclusive cache, but cache replacement is equally complicated and even fetching is fast here.

So, in the case of an exclusive cache, when I want to bring a location and that I have empty my cache is empty, it is just one. But in the case of inclusive cache I have to bring it to l three then L 2 then L 1, but the point is when the cache is full then I have to do a big you know big back off when all the caches are full I need to go and do a big back off. So, you design caches in such a way that these caches do not get full very quickly and that is you get the performance right.

(Refer Slide Time: 22:56)



Now, this whatever we have covered now so far is sort of, so one more thing I have to cover before I to go the multi tasking thing. So last time I did talk about three important things. So, one is called what we call as degree of multi programming, DMA caches, interrupt hardware interrupts. These three are basic building blocks to get what we call as the degree of multi programming. What do you mean by degree of multiprogramming, the number of processes that we need to execute at the same point of time, processes that can be ready for execution. We saw issues like I O burst and CPU burst, I think I have covered this, but anyway I will cover it once more. So, so what is a I O, burst what is a CPU burst, we covered this or not, yes. So, tell me what is a I O burst, what is CPU burst.

Student: Like that the times taken for input output and for CPU processing.

So, no it is not time taken. So, what is a IO burst. So, when a program is execution, when a

process is executing, it will go through alternating phases of CPU burst and I O burst;

in CPU burst you will do lot of CPU activity; in I O burst it will do lot of I O activities I O being tremendously slower. We already saw with an example in the case of RAM, I O being extremely slow by the time I will do one I O for example, by the time I press a key it will do something million operation. So, CPU burst will be quite fast.

So, let us take process one when it does I think we covered this, but anyway I will do that again. When it is doing its I O burst, let us do some amount of CPU burst by the time it is doing its I O burst several p 2 CPU burst can be finished p 3 CPU burst can be finished and so on and so forth. So, many processes CPU burst can be finished while one of the processes doing so that is why so when we look at the organisation there is a front side burst, there is a direct memory access controller what we call as DMA controller here is the CPU here is the peripherals I O peripherals and here is the RAM. During the I O burst of a process this RAM and I O can directly act, we have already seen what we call as direct memory access. So, the I O and the RAM can talk to each other. So, the transfer of data can happen between the RAM and I O independent of the CPU correct. And so, but during the CPU it will do what it will execute other processes.

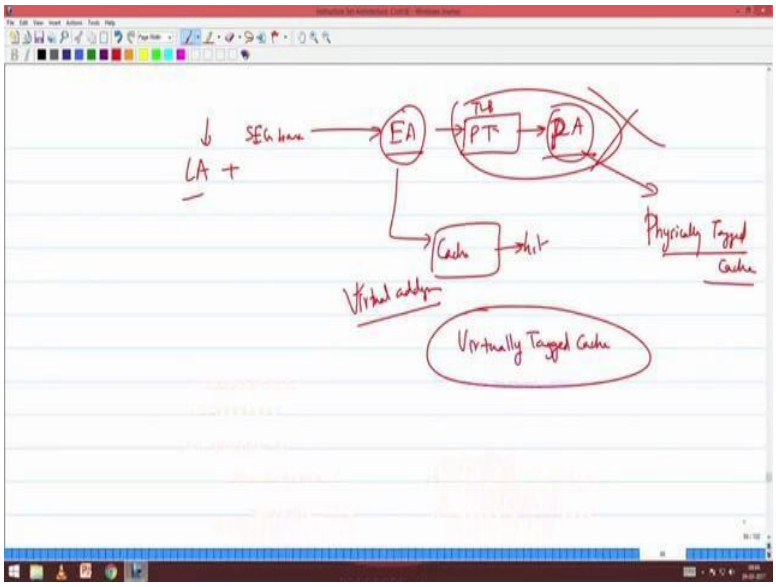
So, the instruction and data for the other processes if it is also there in the RAM then the CPU cannot do anything, CPU has to wait you got it. So, CPU cannot do anything if the I O and the RAM and the if the instructions and data that are required for the next processes are there in the cache, there in the RAM, so that is why cache becomes very important. So, I have a cache. So, when p 1's I O burst is happening the instruction and data for p 2, p 3, p 4, p 5 the subsequent I O CPU burst of these programs will be there in the cache and that can be ensured because of spatial and temporal locality.

So, the CPU can be using that and keep executing the CPU burst of other processes while one process is being the I O burst, are you getting this. So, here this means your CPU and the I O can basically happen together. So, this is actually called as cycle stealing in computer architecture problem. Cycle stealing is when an I O cycle is happening I can steal that cycle and do computation. So, I am hiding the communication overhead by merging it with computation. And this possible because of cache. And also that question that how will I implement the round robin scheduling, we have already seen. What is round robin scheduling I take when I pull you out then put him and there is only one CPU pulling out will be done by the operating system no, it is not done by the operating

system because operating system itself is a software and it ca not execute to pull you out because there is only one CPU.

So, when you are executing such that timer alarm that is set which will pull you out and put the next fellow. So, this one very, very important question that you will be faced you will be asked you know you will be facing any interview committee. So, do not flung there they will crib me right. So, you all understood very clearly. So, this and. So, DMA cache and the hardware interrupt together right work can synchronies work in sync to basically ensure that multiple processes can be executed in a round robin fashion. Cache is all about you know I have an address I am storing some bit or stag bits and some bit as index bits. Some bits are used for mapping and some bits are used for tagging. Now which is the address you will tag.

(Refer Slide Time: 28:43)



So, if I look at so let us go back to the address space. So, there is a compiler which generates some logical address that is gets added with this segment base to get what we call as effective address. And in the presence of paging this goes to a page translation mechanism I get a logical address, sorry I get a physical address. So, a logical address plus segment base gives you effective address that goes to a page translation and I get this physical address. Now, what is the address that I will be using for caching? Can I use the effective address, can should I use the what are advantage. While I am doing that page

translation I can also start looking at the cache, if I use the effective address.

Because the moment I get the effective address and I use effective address for tagging the cache for accessing the cache. So, I can basically use you know the effective address for I will start doing the cache access here itself and if I get a hit [FL], I do not want this. If I get a miss by the time this cache access, this space translation will also be doing it.

So, the space translation will be going through that translation look aside buffer right TLB also. So, this will go through the TLB and then there will be TLB means so much mess can happen there, but while this happening right till a physical address. So, if I use a physical address tag then this becomes sequential first I have to do page translation get the physical address then do a mapping. So, this is what this effective address is basically virtual address. So, I could have a virtually tagged cache or a physically tagged cached.

So, one interesting thing that I get when I use a virtually tagged cache is what I can start accessing the cache the moment I get my logical sorry effective address. I need not wait for the page translation to happen. And if I am getting a cache miss throw it off, but if I get a cache, if I get a cache hit the throw that other part off you can even stop that process. But if I am getting a cache miss by the time I realise cache miss that also be a TLB, and there is a TLB hit then I can start using that then I can start fetching it. So, this a thing.

So, what would happen is so this is one advantage I get if I have a virtually tagged cache versus a physically tagged cache. So, I use a virtual address basically to go and look at this to entirely manage the cache. Now, what is we get complication here having a virtually tagged cache?

Student: We update something in the cache have to write it back to the RAM. So, before that you have to translate it to the physical address.

Yeah. So, if I have a cache miss it is hell for me if I have a cache hit then it is heaven for me correct. If I have a cache miss then I do not know this only a virtual address where in the memory it is going to be there I do not know. So, then I have to wait for the entire translation to happen and then I have to go and map it and go back so that is a big complication when I have a virtually tagged cache. So, can I have this question so we are touching 950. So, can I have a virtually tagged physically mapped cache right, I leave the story here. Can I have a virtually tagged physically mapped cache? You almost did the

class. So, her reasoning is correct wherein if I have a virtual tag I can do things in can parallel because already memory access is crap I am I am into so much issues so monkey

Now, I want to access I want to get rid of that whole thing in whatever best way. So, I will try to do all the best to see that my lot of concurrency happens and this gives me a beautiful concurrency as you see here. But then the complication happens when I want to do a replacement that is a nice thing. So, can I have the best of both where I can do this and that. So, let us can I have a virtually tagged physically mapped cache or vice versa. So, just start thinking about it.