Computer Architecture

Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

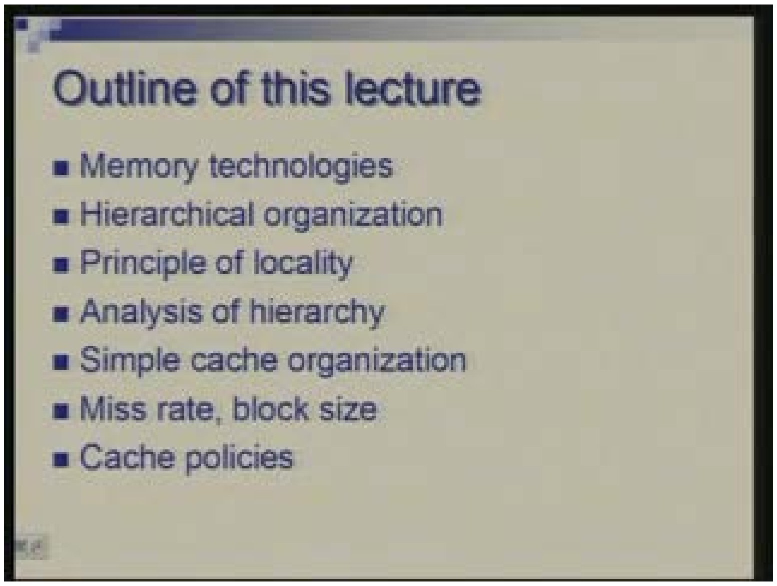
Lecture - 28

Memory Hierarchy: Basic Idea

We have been discussing design of processor which is the main component of complete computer system. Our discussion in the recent lectures have been focused on increasing the performance and what we have been attempting to do is to execute as many instruction as possible in as few cycles as possible. So all this puts great demand on memory because we need to fetch in simple pipeline case one instruction per cycle and if we go for ILP then we have to go for multiple instructions to fetch per cycle so that is the requirement of instruction fetching which is put on the memory; at the same time you also have requirement of data transfer for instruction which accesses memory like load and store.

So far we have assumed that memory is flat that means you have simply an array of words; you give an address you can get the word or you can give an address and data you can write the word. But with the given memory technology if memory is organized in this particular manner it cannot meet the demands of performance which is placed on it by the processor. So we need to have a more complex structure which is called memory hierarchy and that is what we will start discussing today.

(Refer Slide Time: 00:02:25)



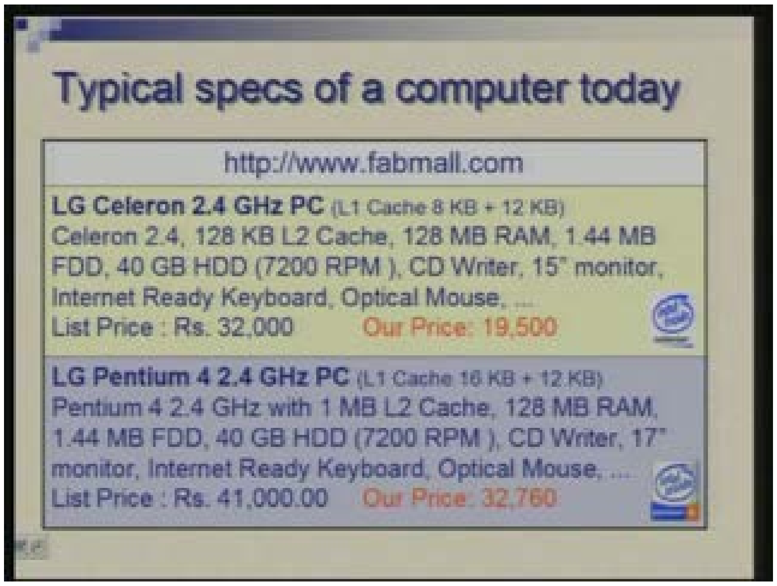
Outline of this lecture

- Memory technologies
- Hierarchical organization
- Principle of locality
- Analysis of hierarchy
- Simple cache organization
- Miss rate, block size
- Cache policies

So we will quickly look at what memory technologies are available. We do not go into deep of this issue but just to notice a point that there is variation in cost and speed and sizes; and we will look at memory hierarchy as an attempt to put these different technologies together in the same framework. This works on the principle of locality that is same thing you can access again and again; what you access once gets reused. We will look at impact of this approach on the performance and we will look at a specific case of hierarchy or specific level of hierarchy called cache organization.

For a simple cache organization we will see how performance parameter called miss rate varies with some architectural parameter such as block size and we look at some cache policies which are followed in different systems.

(Refer Slide Time: 00:03:34)



Typical specs of a computer today

http://www.fabmall.com

LG Celeron 2.4 GHz PC (L1 Cache 8 KB + 12 KB)
Celeron 2.4, 128 KB L2 Cache, 128 MB RAM, 1.44 MB FDD, 40 GB HDD (7200 RPM), CD Writer, 15" monitor, Internet Ready Keyboard, Optical Mouse, ...
List Price : Rs. 32,000 Our Price: 19,500

LG Pentium 4 2.4 GHz PC (L1 Cache 16 KB + 12 KB)
Pentium 4 2.4 GHz with 1 MB L2 Cache, 128 MB RAM, 1.44 MB FDD, 40 GB HDD (7200 RPM), CD Writer, 17" monitor, Internet Ready Keyboard, Optical Mouse, ...
List Price : Rs. 41,000.00 Our Price: 32,760

So let us look at a typical add which you see today; for computer I have picked this up from an online computer store. You see a typical configuration which is specified here and many of the items you note here actually refer to storage of information. For example, you have cache; you are talking of L2 cache, RAM, FDD with a floppy disk drive, hard disk drive HDD, CD so all these are technologies of media for storing information.

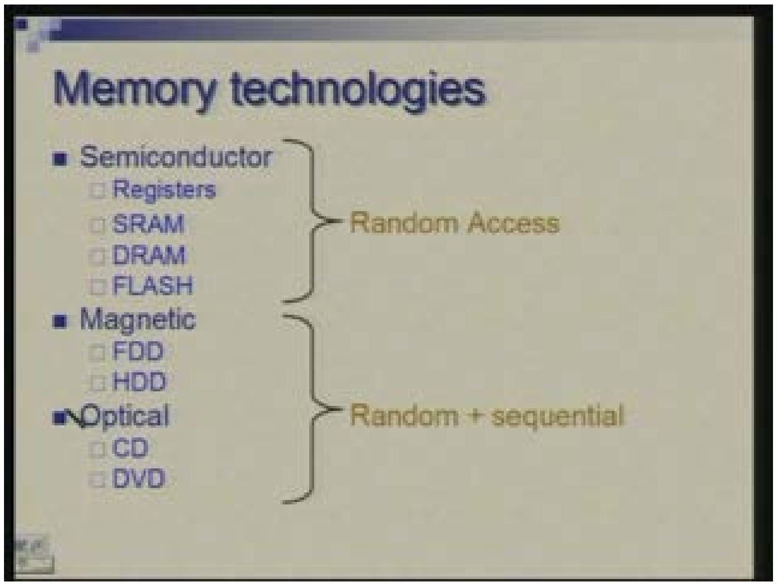
Similarly, here you again see similar things (Refer Slide Time: 4:17). Of course this part I have added, this part generate does not get advertised; what you see is what is you see processor as just one component and other things which are external to that. So, for example, there is typically several mega bytes of memory which is called RAM or the main memory and there are backup memory such as FDD or HDD.

Cache is a faster memory which is typically placed within this within the CPU chip; some part could be out of the CPU chip also so we will talk of those details later on.

So you would notice that whereas one begins by looking at how fast the processor is that is this GHz spec so you are talking of 2.4 GHz here, one also talks of various size of memory. So, size of memory is important and how various types of memories have been put together in a configuration is important. You would notice that I have taken two examples where the price differs substantially and what is the reason for the price difference. There is of course a better processor. You see the same GHz or same rate of clock but this is a more powerful processor in terms of the way it is internally organized......... also the other issue is that the cache is much larger here.

So what is causing this price difference?
One is better processor; secondly, RAM is same but cache is drastically different. And of course there is difference in some of the peripherals like 15 inch monitor and 17 inch monitor but the cache is equally important as the processor.

(Refer Slide Time: 00:06:34)



Memory technologies

- Semiconductor
  - Registers
  - SRAM
  - DRAM
  - FLASH
- Magnetic
  - FDD
  - HDD
- Optical
  - CD
  - DVD

Random Access
Random + sequential

So briefly we notice the presence of different types of memory technology. There is one category which is solid state semiconductor memory. Here there are no moving parts involved (Refer Slide Time: 6:54) whereas you have magnetic memory which we find in floppy disk drive and hard disk drive there are moving media for optical memory as CD or DVD these are also moving. In these the difference is in terms of the way information is getting stored; here information is in magnetic form so let us say on a on a on a surface of a storing medium there region which gets magnetized with one polarity or the other polarity and that distinguishes 1 or 0. Similarly, in optical memory, there regions which could be opaque or transparent and that makes a difference between 1 and 0 whereas these are purely electronic and consequently they are much faster.

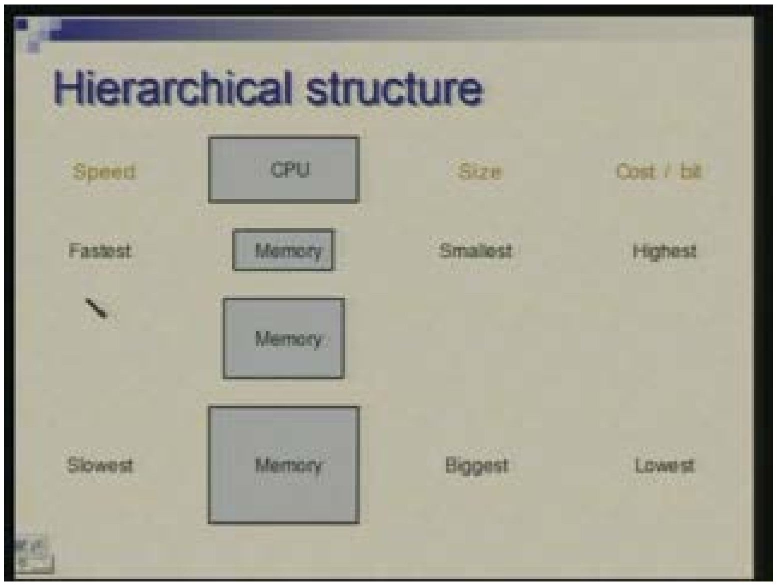
Here (Refer Slide Time: 7:47) some movement is involved which makes things somewhat slower whereas in semiconductor memory there is no movement involved it is only that you give an address and you get the data so these are comparatively much faster. There is also difference in organization in the sense that these semiconductor memories are random access that is you given an address it takes same time to get a word so all all are all words are organized symmetrically whereas in these memories which have moving parts they there is some sequentiality so you may be able to get some information faster some information slower because things are organized along tracks and one has to move along the track and also across the track in a sequential manner. So these two factors: mechanical movement and sequentiality make these memories slower.

But, on the other hand, the capacity which you get in these magnetic or optical memory is orders of magnitude higher than the semiconductor memory. So you could either say that for same price you can get much much more capacity or you could say that the capacity or the cost of storage per bit or per byte or per word is much lower in case of magnetic and optical memories.

Now, within each group there are differences. For example, DVD has much more capacity than a CD; HDD has much more capacity than FDDs; also HDD is faster much much much faster than FDD; within the semiconductor memory there we will focus our immediate attention which is that you have registers which are placed as an integral part of CPU they they are very fast whereas cache memories and main memory which are placed outside are slower comparatively. So, within these memories SRAM is faster than DRAM; SRAM in SRAM the information is stored in terms of flip-flops. So you imagine cross coupled gates which can be in one of the two stable states 0 or 1 and information can be stably stored there.

So reading and writing are sensing the state of a flip-flop or changing the state of a flip-flop is much faster than what happened in DRAM where information is stored in terms of charge on a capacitor so it takes much longer to charge or discharge capacitor. But again typically there is a tradeoff between speed and cost. So among these are the technologies which are slower offer you larger capacity and vice versa.

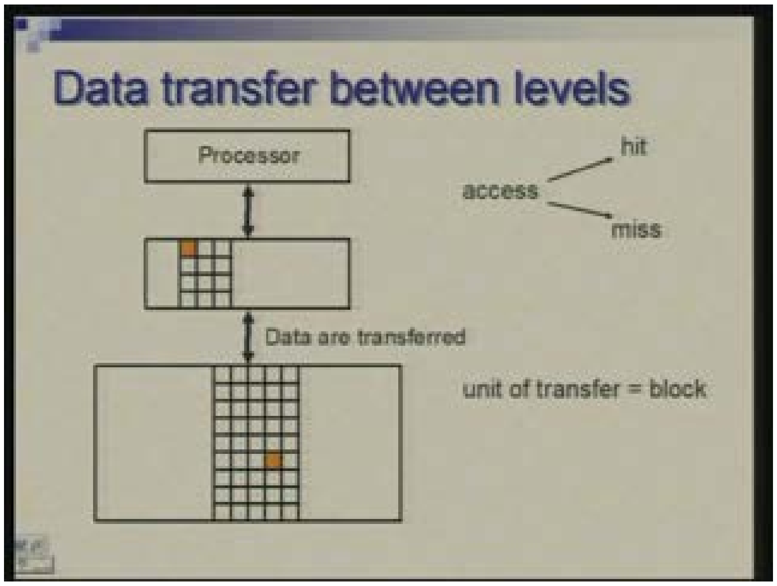
(Refer Slide Time: 00:10:55)



So the idea of hierarchical structure is to use not just one kind memory but several different kinds of memory and try to get the best of various properties. So you will place typically the fastest memory close to CPU, next a slower one after that and the slowest one farthest away and typically this the one which is closest will be smallest in size; the cost per bit will be highest and the one which is farthest will be largest in size and cost per bit will be the lowest.

So now what you would like is to get an impression or get an effective feeling of having the speed of this memory and the cost of cost or size of that memory. if you if you can get these two best of both worlds then it is ideal and in fact the techniques which have been developed over years in organizing the hierarchical memories tend to give you almost that so you will get speed almost same as the fastest memory and you will enjoy the capacity almost as much as the largest memory.

(Refer Slide Time: 00:12:26)



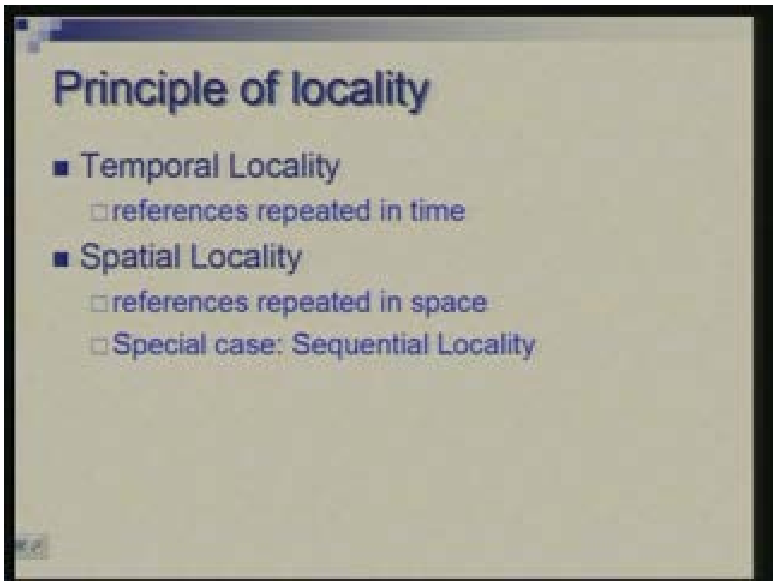
Data transfer between levels

So what happens is that you would have information placed in various memories. The processor would start by trying to get it from the fastest memory because that is an attempt hopefully you will get it there. If you if you are able to get it there then it is considered a hit if you do not it is consider a miss. So one has to organize in such a manner that most of the time it is hit let us say 90 percent of a time or even 99 or 99.99; depending upon where we are in the hierarchy one would target for very high hit rate or very small miss rate that is the hole that is the key so the whole requirement can be captured in one..... if there is one single parameter which you need to focus your attention most is this the probability of hit or the percentage of time you get a hit which you want to make as close to 1 as possible.

So hopefully things will work nicely; you will get most of the time hit; the information will be organized in such a manner that it happens. But once in a while when you get a miss then you go to the next level and try to get information from there. So, if you don't the same thing will happen at that level; if it is not available there you go to the next higher level and so on. So you are optimistic try to get at the level 1 if you do not get at level 2 if you do not get there go to level 3, 4 and so on so the unit of data which you transfer when you go down the level may also change and it is called a block.

So when processor is accessing the first level of memory typically it will try to look for a word. Let us say there is a load instruction or you are fetching an instruction so you want to basically read a word but when you do not find it here you go to the next level it is not necessary that you will look for just one word there you may bring in more information keeping the future requirements in mind so unit of transfer may differ as you traverse across the levels.

(Refer Slide Time: 00:14:36)



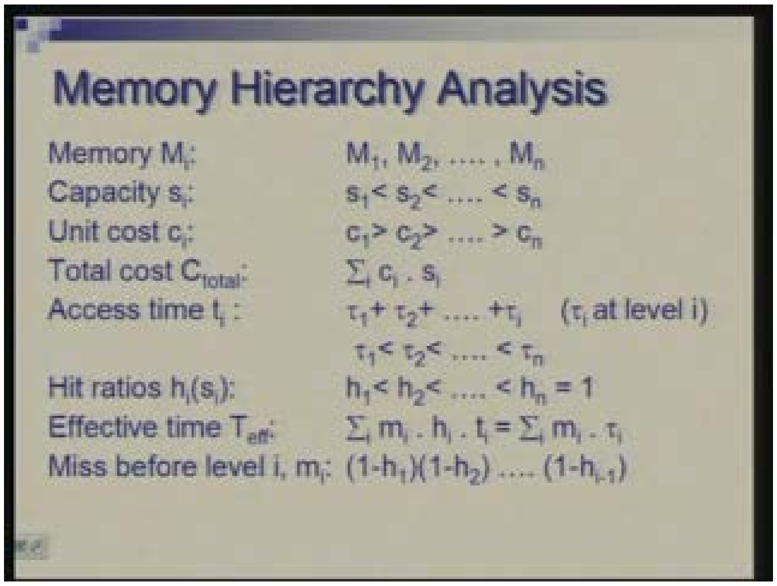
Principle of locality

- Temporal Locality
  - references repeated in time
- Spatial Locality
  - references repeated in space
  - Special case: Sequential Locality

So this whole thing works on the principle of locality which means that references to memory are localized in sense of time or the temporal sense as well as in spatial sense or in terms of space. Temporal locality means that if you have referred to some word you are likely to refer to it again.

So if you have accessed something and kept it at higher level in the hierarchy you are likely to reuse it. Spatial locality means that if you are accessing a word you are likely to access other word which is in the region around it. So address features close to that is likely targeted in the future. A special form of spatial locality is sequential locality. That means you are accessing neighboring addresses in a very particular way that is you are going one after another in a sequence.

(Refer Slide Time: 00:15:47)



Memory Hierarchy Analysis

Memory M_i:	$M_1, M_2, \ldots, M_n$
Capacity s_i:	$s_1 < s_2 < \ldots < s_n$
Unit cost c_i:	$c_1 > c_2 > \ldots > c_n$
Total cost C_{total}:	$\sum_i c_i \cdot s_i$
Access time t_i:	$\tau_1 + \tau_2 + \ldots + \tau_i$ (τ_i at level i)
	$\tau_1 < \tau_2 < \ldots < \tau_n$
Hit ratios $h_i(s_i)$:	$h_1 < h_2 < \ldots < h_n = 1$
Effective time T_{eff}:	$\sum_i m_i \cdot h_i \cdot t_i = \sum_i m_i \cdot \tau_i$
Miss before level i, m_i:	$(1-h_1)(1-h_2) \ldots (1-h_{i-1})$

So let us look at various levels of hierarchy in a little more analytical way. Suppose you have several level let us say M levels we call them M 1 M 2 and so on M n the capacity at various levels in terms of bytes or words or whatever units you have is S i so s 1 will be less than s 2 less than s 3 and so on so as you go down the level the size is increased; the unit cost decreases as you go down the level and access time....(.... 16:29) missed out one thing okay access time will increase as you go down the level that means things will become slower and slower.

Now, in terms of these parameters we can look at the total cost and total effect of time. So total cost is simply a weighted sum of costs at every level. Now there are lot of simplifications here; I am simply adding the cost of the storage but there is also additional cost of controllers and interfaces and so on. So just to give a broad picture I am just looking at the storage cost; there are other overheads which lead to be arranged in general.

So this is the total cost (Refer Slide Time: 17:13). So now what you are given in terms of various technologies are these c values. When you are configuring a system or you are designing a system what you are trying to choose is different values of s i's. So, depending upon what choices you make for given c values you get an overall cost.

On the other hand, the access time at level i is composed of individual accesses at various levels so tau i is the access time or the time which is spent at level i. But when you are actually accessing when you are getting the data at level i you would have gone through the previous level earlier. So the overall access time at level i is actually sum of accesses you have made that level as well as the earlier level so t i is sum of tau 1 tau 2 up to tau i and these taus are (Refer Slide Time: 18:19) are in increasing order as you go down the hierarchy.

You expect the hit ratios or hit percentages also to be increasing as you go down. So if you you may have let us say 90 percent chances of finding 99, next level may be 99.99 and so on. So as you go down since the capacities are larger and larger you are likely to have more and more information there and you are likely to find them with greater probability and everything must be present at the last level. So h n the last term will be 1.

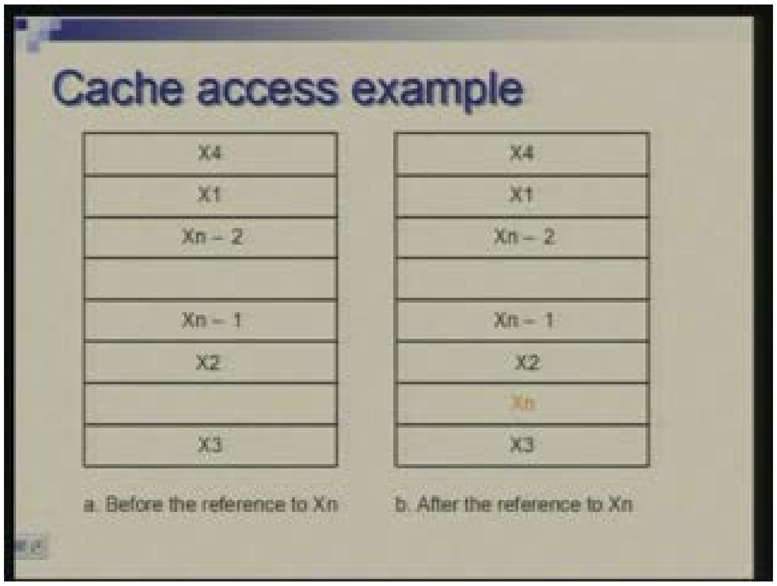
Now the effective time overall is some combination because sometime you may get information at level 1, sometime at level 2, sometime at level 3 and sometime at level 4 and so on so it is a combined effect and it will take into account how often we do this how often we do that so T effective is sum of these (Refer Slide Time: 19:38); the probability that you actually get data at level i when you have gone through previous levels unsuccessfully. So m i is a probability that there are misses all through before level i. So, if there are misses before level i and hit at level i then your search terminates at level i and you incur time t i. So basically you sum t i values with this as weightage.

So m 1 h 1 t 1 that means m 1 will be m 1 will be 1 always and it will m m i will decrease as you go down. The chances of misses are less and less as you go down. this sum (Refer Slide Time: 20:45) is also expressible in this form; you are actually making access to t i the t i component gets into play when there is a miss at levels before i whether you are hitting at i or hitting at a level higher than i sorry level lower than i higher; larger value of i means I will use that to indicate lower level. So first level is the highest level, second is lower and so on.

So if you are going up to level i or beyond tau i will come into play so a simpler form is that you sum tau i with weightage m i so either expresses this way or that way. In fact you can also express m i in terms of h values that is m i means that you have misses at all the levels up to i minus 1. So h one is hit probability and 1 minus h 1 is miss probability. So miss at level 1, miss at level, miss at level 3 and so on up to misses all the way up to level i minus 1. So product of all these is that you have miss at all the levels before i.

Now what we are trying to say here; now basically there are two things which need to be considered: the total cost and effective time. So what you are choosing is basically s values. So you are given different technologies characterized by c values and tau values and you need to choose s values because s is going to affect the overall cost and s is also going to affect the miss probabilities at various levels and one has to choose so that you get the combination of cost and time. This is not a very accurate way of analyzing; there are many issues which have been actually somewhat overlooked so it is a simplified model but what it tries to do is it tries to bring this cost and time together in a same formulation. So this was generally about hierarchy of memory.

(Refer Slide Time: 00:23:11)



We would talk of two particular hierarchal interfaces: one is the cache which is the first memory or first few levels of memory before the main memory. So there is always a main memory which forms the reference point so before that the level higher than that are called caches; you may have one level of cache, two levels of cache and there are now systems with even three levels of cache and what is beyond at lower levels beyond the main memory is called virtual memory. So we will talk of main memory virtual memory interface; we will talk of cache and main memory interface. So let us start with cache.

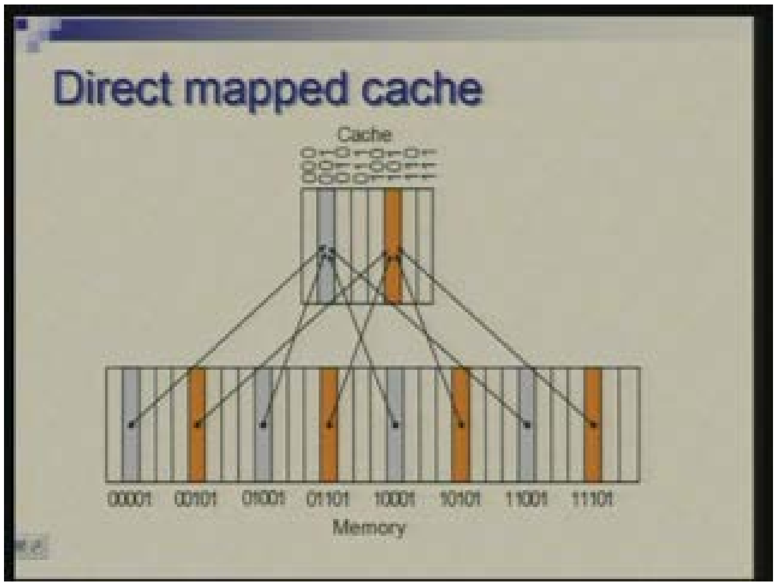
The idea is very simple that at any given time the cache..... let us imagine only one level of cache in the initial discussion. So, in the cache you will have some subset of the data and instruction which you have in the main memory and at any given time you may when you need something from memory you first make reference to cache if it is there you pickup otherwise you get from the main memory.

So, for example, suppose at some point you refer to a variable Xn or location Xn and cache contains these in different locations; you find Xn is not there, you bring it and position it at appropriate point and then if subsequent references are made to Xn you will find it there so they may be a miss initially and hopefully there will be several hits later on so that you can get fast. If every data or every instruction is referred to only once and there are no localities then of course you will get the worst of the time you will get the time same as the slower of the memory. But fortunately that does not happen because if you look at the program most of the time program goes sequentially unless there is a branch so there is a sequential locality and the loops bring in temporal locality because same instructions get executed again and again.

In fact if you have a small loop the whole loop will eventually sit in the cache and you will keep on executing all the iterations in the loop from the cache only. Also, for data you have data structures such as arrays, records were consecutive locations get accessed at some computation.

Now question is what do you place where?
You have memory main memory which is of larger size, cache of smaller size so which part you place in the cache and where you place in the cache. One simple organization is called direct map cache where given a memory location there is a fixed place in the cache where it can be kept; so, given a memory address from that you can find out its location in the cache.

(Refer Slide Time: 00:26:15)



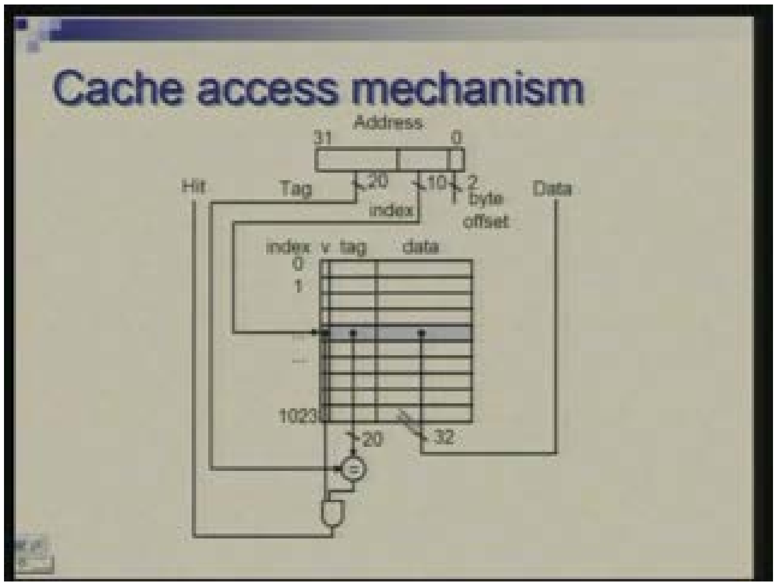
Direct mapped cache

So, for example, here you have a cache of size 1 2 3 4 5 6 7 8 words and you have memory which is four times the size that is thirty two words; just a simple toy example to illustrate the idea. So we can imagine the main memory also to be divided into segments or portions of size which are same as cache.

So let us say you look at the first eight words of the memory; first quarter of the memory then you can see a direct correspondence between locations of memory and locations of cache. Since memory is four times you will have four such quarters and you can see that all grey locations of the main memory map to this grey location of the cache and all orange locations map to this orange location and so on. So in each part of the first location of each quarter maps to first location and so on.

Now this makes the task very simple. Given memory address which you are trying to refer to what you need to do is you need to figure out which location of the cache it is likely to be and if at all; you go there and check if it exists there and if it is there your job is done otherwise you need to do something else.

(Refer Slide Time: 00:28:09)


Cache access mechanism

So, in terms of the hardware structure what you do is from the address of memory..... now I am showing this illustration with 1024 words in the cache and memory addresses are 32-bit addresses which means there are 4 Giga bytes or 1 Giga words of the main memory so last 2 bits are byte number or byte offset within a word so let us keep that out the remaining thirty words define one of the 2 raised to the power 30 memory main memory words.

The cache here is consisting of 1024 words which mean it can be addressed by 10 bits. So let us say out of these 30 bits the lower 0 bits will take us to the possible place where a word can be found. What we need to know now is whether this is the word we are looking at looking for or not so we need to carry some identity with each word in the cache.

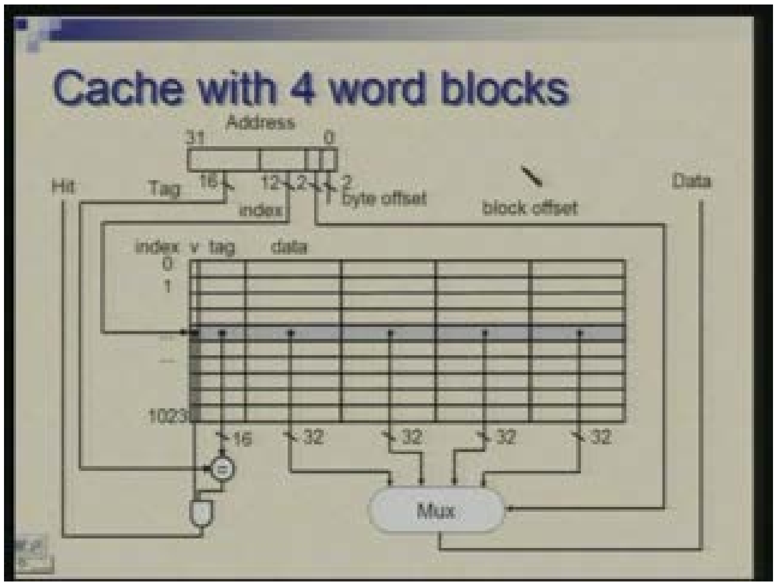
So, for example, as in the previous case if we go to this one you have 5-bit addresses here and 3-bit addresses there so out of the 5 bits of the address the lower three can be used to index and reach one point in the cache but the cache will have to tell which of the quarters this has come from; the word sitting at the gray position here could be this one or this one or this one or this one (Refer Slide Time: 29:48) so it needs to carry a 2-bit tag to indicate from which quarter it has come from.

Correspondingly in this picture the remaining 20 bits will identify one of the million possible words of main memory which can map to same word in the cache. So therefore each cache word needs to carry a 20-bit tag which will identify which one of those million things it is holding at the moment.

As you see here that this is an overhead (Refer Slide Time: 30:25). so rather in just storing 32 bits of data or instruction we carry additional 20 bits and I am showing one more bit which is called V bit or valid bit which will tell whether this entry in the cache is filled up at all or not because you would start typically with an empty cache; the whole thing is a main memory and only as you bring things into the cache something becomes valid. So you need to distinguish an empty location from one which is filled up something useful.

Now the process can be looked at as follows that out of the 30 bits address you look at 10 bits use that to index into one of these entries, take the remain 20 bits of address and match with the tag here. if the tags match and valid bit is 1 then you say it is a hit otherwise you would say if it is not valid that means it is empty, if it is valid but tag is not matching that means there is a valid entry but something else is sitting there and this is not what you want so in that case it will be a miss so this signal indicates a hit or a miss and if it is a hit then here is a data which you can read out; write process could be similar.

(Refer Slide Time: 00:31:48)


Cache with 4 word blocks

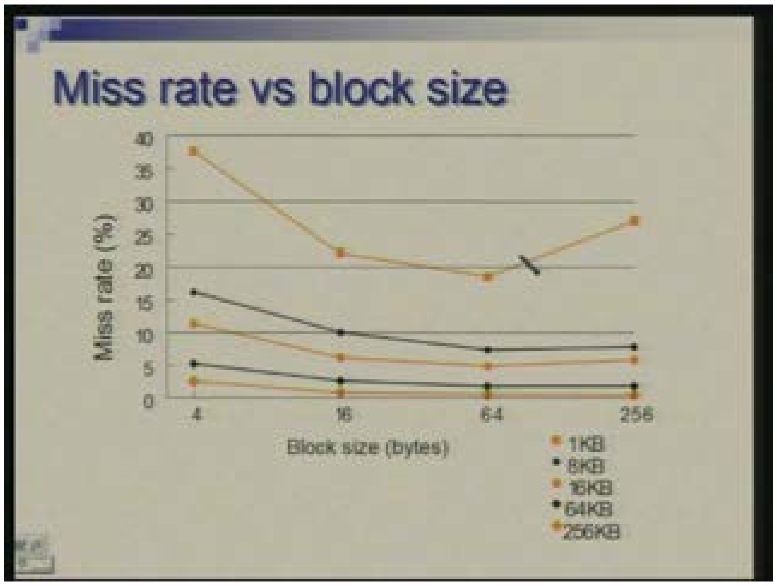
This was a very simple case where the block which I mentioned as the unit of data transfer between two levels. Now we are talking of interface between cache and the main memory; do we transfer one word at a time or we transfer more words?

Suppose we look at 4 word blocks that means every time there is a miss in the cache you are going to get not just one word but four words with the hope that others will also be useful at a rate of point of time because of spatial locality locality of space. so now same structure I extend horizontally and in each row you can place not just one word of data but multiple words of data and rest of the mechanism is similar except that now the field size change.

Of course the situation is not quiet same here I am assuming a larger cache here they are actually there is some inconsistency the cache is much larger here but this number is not correct (Refer Slide Time: 33:09) I am assuming a 12-bit index here which will this should be 4095 to this part 12 minus 1. So now there are four thousand odd blocks in the cache each block consisting of four words so the total capacity of the cache is 16k words and these 12 bits in the usual manner index into one of the rows.

We do similarly tag matching; the tag size is different now because more bits have been occupied by other fields. There are 2 bits now which specify a word within a block and the last two bits will specify a byte within a word. So, to get a word which we are looking for what we need to do is have an index into this whole array, pickup one row, look at the tag and the valid bits first, do the tag matching and hand it with valid bit to decide hit or a miss. If it is a hit you can read out four words and select one out of those four by using block of set or the word number within a block as a selection signal. So this is the data which comes out.

(Refer Slide Time: 00:34:32)



So, what is the advantage of having a block different from one; having a larger block?
As I mentioned that with a larger block you are trying to capture locality of space. So, if a word is missed out and you are getting from main memory why not get more. So, with a hope that there will be a near future references to those words also then you do not have to worry about miss at that time. So one would expect that as block size increases there will be improvement in the performance or the miss rate will go down which indeed happens. So here is a series of graphs which show miss rate depicted as a percentage versus block size and different curves correspond to different total cache size. So you would notice for obvious reasons as you increase the cache size the miss rate is going down.

For example, let us let us go across the curves; the top one corresponds to 1KB the next one corresponds to 8KB, 16KB, 64KB and 256KB. As you are increasing the cache capacity you are capturing more and more information and more and more likely that you will have a hit and therefore miss rates are lower.

Secondly, as the block size is increasing you are capturing more spatial locality so there is a further decrease in miss rate as you go towards right. But peculiarly you will find that after some point there is a tendency of miss rate to increase and it is much more pronounced in smaller caches; why do you think that could happen?

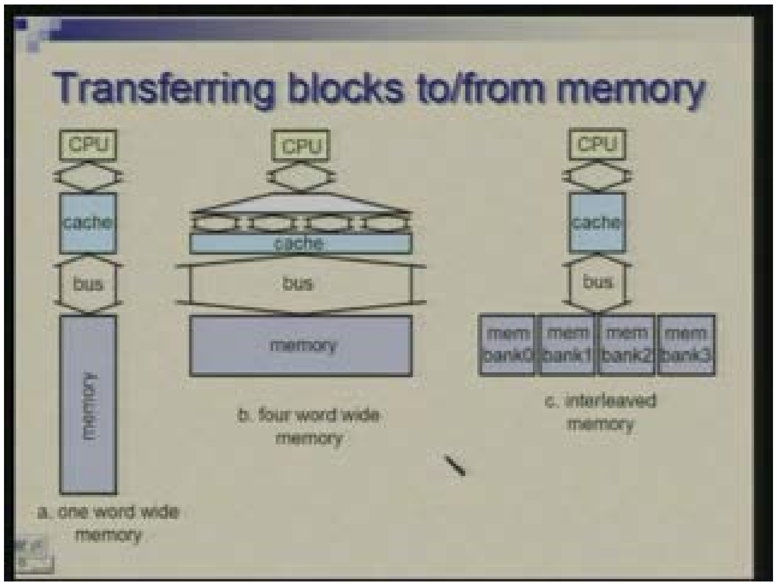
[Conversation between student and professor: the original values in the cache get replaced by the [inaudible......36:46]] yeah, you are saying something close to that but you have not worded properly.

What happens is that when cache is small and you try to make larger and larger blocks the total number of localities you are capturing are much fewer. So typically a program would behave that there will be few localities that is in some region of the program you will spend lot of time and another region you will spend lot of time so there could be many localities in the program as well as in the data so you will be accessing some data structures more; you will spend more time with those and large larger block size means fewer blocks so when you are bringing a new block you are also replacing something which is already there. You might end up in throwing something throwing out something which is useful because there are only few smaller number of blocks which you can accommodate and that is the reason why beyond certain point it does not pay you off to increase the block size. So how far you can go gainfully also depends upon the cache size; if the caches are large you can have more block size.

There is there is one more impact of block size which does not show up here is that the time to process a miss will also possibly increase as the block size increases. So the overall performance will not be....... we will work on it in subsequent lectures but the overall performance is not just miss rate. Miss rate is a crucial factor but we will derive the expressions for over performance or what impact cache misses have on the CPI. We have earlier seen that CPI is a key indicator cycles per instruction so effectively we will see that processor spends more cycles in executing a given number of instruction and that will be influenced by miss rate and also the time it takes to process a miss. So with large blocks that also becomes worse and we need to take into account this whole picture.

So let me throw some light on what happens when a miss occurs.

(Refer Slide Time: 00:39:03)


Transferring blocks to/from memory

We are trying to transfer now multiple words between memory and cache; how it is done?
One approach is shown here on the left side that you have CPU, you have cache you have memory and you can have buses of one word width connecting CPU and cache and CPU and cache and memory. So now CPU cache connection is fine that you are looking at one word at a time but when you want to transfer a multiple word block between memory and cache then each word has to come sequentially over this bus. So one may tend to think of this kind of organization that between cache and CPU there is a path which is one word wide but between memory and cache we can have wider path let us say four words wide.

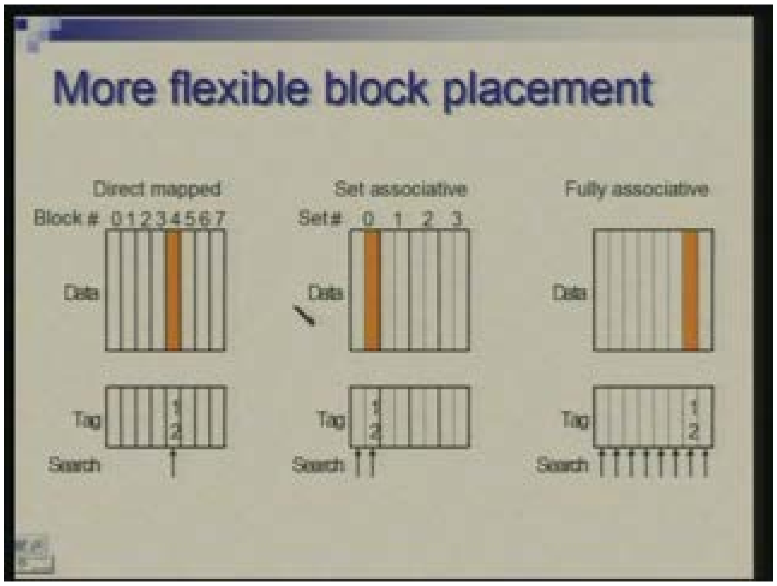
So what it means is that memory gets organized in such a manner that when you give an address you get four words simultaneously. So there is of course cost associated with having a path like this. Let us say your word is 32 bits then four words would mean you are transferring 128 bits at a time; you have 128-bit bus which will make things expensive but definitely it will give you better performance that is in one bus cycle not necessarily in CPU cycle; in one cycle of the bus you can get the whole block transferred.

There is another organization which tries to get again the performance of this at the cost of this organization. So here we get good performance but at high cost. So can we get reasonable performance at a reasonable cost; that is the issue.

What we do is we organize memory as banks. Rather than making it a wide memory connected through a wide bus we chop it into let us say four parts call each as a bank and interleave the addresses that is keep address zero here in bank 0, address one in bank 1, address two in bank 2, address three in bank 3, 4 here, 5 here, 6 here, 7 here and so on so addresses are interleaved in this particular manner and now each block you are looking for each four word block is spread across four memory banks.

So when an address is sent the address can be fanned out to all four; all four will respond with the one word each which they have to contribute to the block but we keep the bus narrow and keep the cost lower so that the words get transferred one by one. So memory memories have some access time; after you give the address they take some time to get the word so that happens in parallel for all the blocks or all the four words of the block. If you have narrower but faster bus then these words once you have got them from memory can be quickly transferred to the cache. So you will get more or less same time as you get here; of course there will be additional transfer time so it is not quite as fast as that but closer to this than that and in terms of cost it maybe closer to this than that so that is a reasonable arrangement.

(Refer Slide Time: 00:42:44)


More flexible block placement

Now, the organization of cache which we have seen is very rigid in the sense that a given word of memory or given block of main memory can be placed at a fixed place in the cache so that also cause this problem because every time you bring you may possibly replace and if you happen to be if a program happens to be in a state that it is quickly referring to two or more location which are mapping to same location in the cache then you are doing very badly you know you every time you bring 1 you are replacing something useful so you need an arrangement where there is some flexibility of placing things wherever you want.

So look at the picture on the left where I am showing a direct map cache. So now a given block can be placed at a given location and you have to actually rely on tag matching to see whether the thing is sitting there or not.

On the other hand, suppose you want to have full flexibility then you will take a word and place it anywhere in the cache but the consequence of that is that you need when you are looking for this word with which you have placed you do not know where you need to look at all the possible locations and you cannot do that search sequentially; you need to do you need to look at all the locations where the given block can be found in parallel so it has to be done using a memory which is called associative memory so cache has to be organized as an associative memory where you will basically supply the tag value and see if it matches anywhere or not.

What you do in direct map caches that you go to a particular place look at the tag and see there is a match you are not looking around anywhere else. In fully associated match in fully associative cache you will take the tag and try to see all the places and see if it matches anywhere if it matches somewhere then there is a hit if it does not match anywhere there is a miss. Such memory can turn out to be very expensive because obviously with every word or every block there has to be hardware which will match the tags so those comparators must be there.

Again there is a compromise which tries to form sets out of the whole cache area and it gives you limited flexibility. What it says is that given word in the memory it can be placed anywhere within a set so it gives a limited flexibility. If you make the set size as 1 then this reduces to direct map cache.

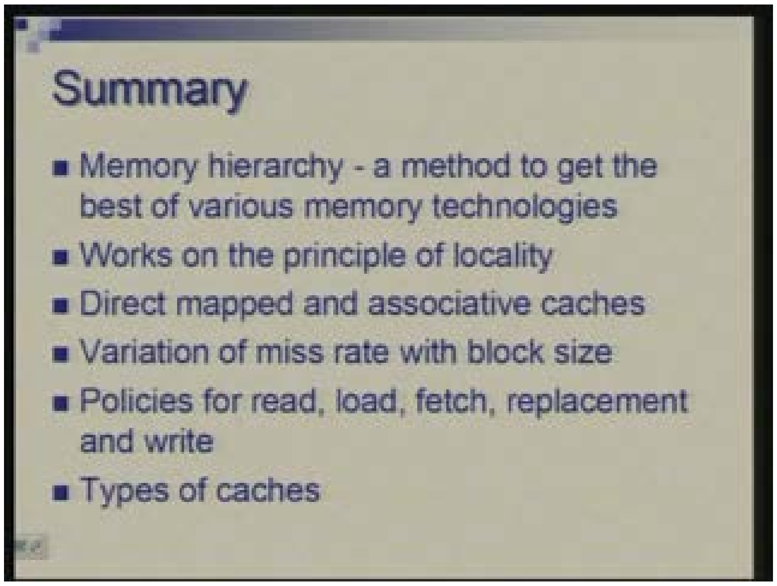
If, on the other hand, you make the whole thing as a single set then you make it a fully associative memory. So set associative is somewhere a compromise between these two cases.

In this picture (Refer Slide Time: 46:05) we are showing the whole cache divided into four sets. What it means is that given data can be anywhere in the cache anywhere in a particular set. Suppose this is a set which you have placed in then you need to look at two places which means that the tag matching needs to be done in a limited context in these two places and if one of them has it then you have a hit otherwise there is a miss.

I think time is running out and I will skip the cache policies and stop here.

So let me summarize what we have discussed today.

(Refer Slide Time: 00:46:52)



Summary

- Memory hierarchy - a method to get the best of various memory technologies
- Works on the principle of locality
- Direct mapped and associative caches
- Variation of miss rate with block size
- Policies for read, load, fetch, replacement and write
- Types of caches

We looked at memory hierarchy as a means to get best of various types of memory technologies; get speed of one and the cost of other and this works on the principle of locality that means same data or instruction gets referred to again in time or gets referred to with locality of addresses. We have talked of direct mapped cache as a simplest cache organization and seen briefly the possibility of associative cache. We also looked at miss rate variation with block size; we have not yet looked at the policies and the types of cache we will do that in the next lecture. Thank you