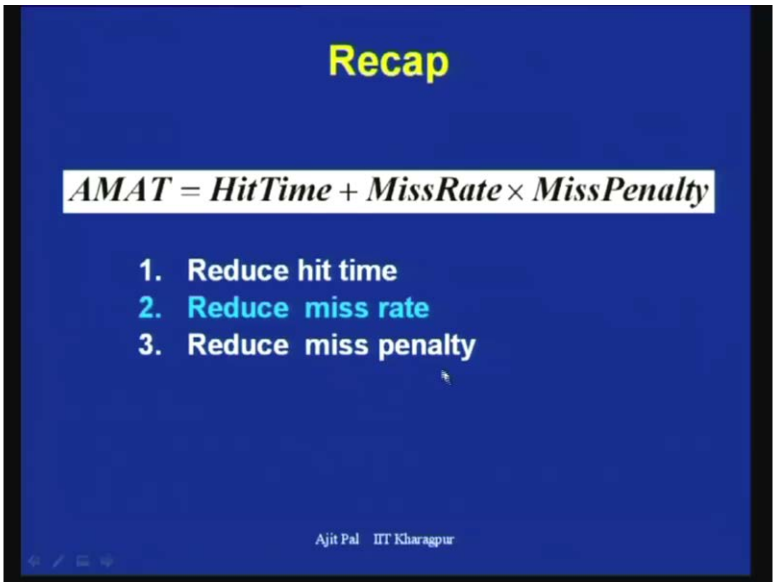
High Performance Computer Architecture

Prof. Ajit pal
Department of Computer Science Engineering
Indian Institution of Technology, Kharagpur

Lecture - 25
Cache Optimization Techniques (Contd.)

Hello and welcome to today's lecture on cache optimization techniques, this is the second lecture on this topic.

(Refer Slide Time: 01:03)



Recap

$$AMAT = HitTime + MissRate \times MissPenalty$$

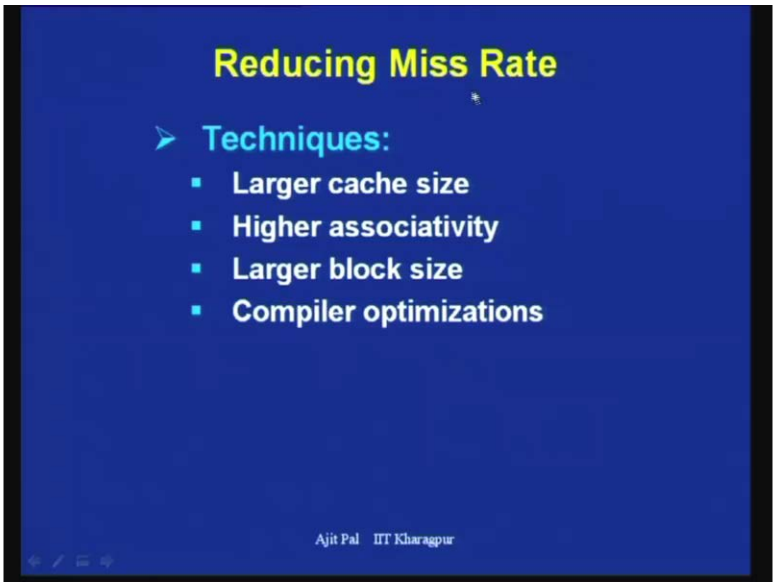
1. Reduce hit time
2. Reduce miss rate
3. Reduce miss penalty

Ajit Pal IIT Kharagpur

In the last lecture, we have discuss about the reduction of hit time as you know the average memory access time particularly from cache memory is dependent on three important parameters, number one is hit time, second is miss rate, third is miss penalty. To improve the performance, you have to reduce on or more of these parameters that means first technique that you can use is to reduce hit time. In my last lecture, I have discuss in detail various techniques by which the hit time from the cache can be reduced because CPU is reading instruction and data and writing also instruction and data in to I mean in the from the cache memory.

The performance is critically dependent on the performance of the cache memory that is the reason why is very important to discuss about various techniques by which the cache memory performance can be improved. So, today we shall focus on the second technique that means reduction of miss rate, how can we reduce the miss rate and in my next lecture I shall focus on reduction of miss penalty.

(Refer Slide Time: 02:29)



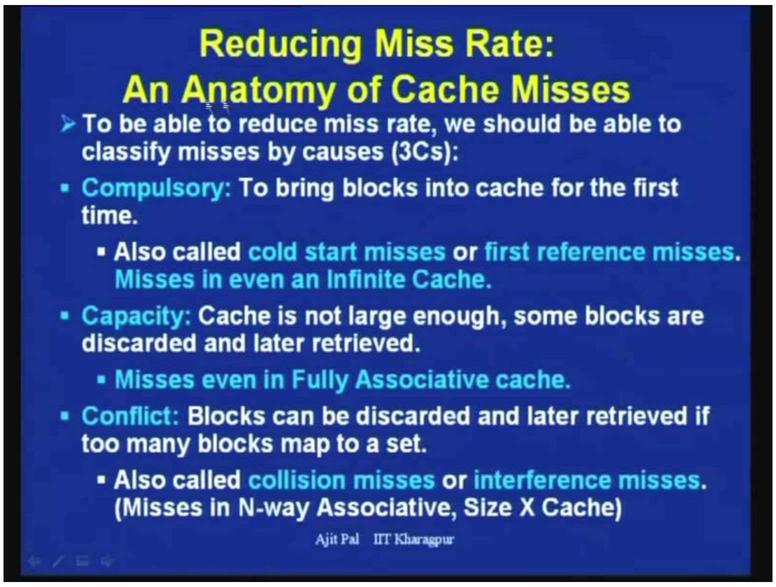
Reducing Miss Rate

- Techniques:
  - Larger cache size
  - Higher associativity
  - Larger block size
  - Compiler optimizations

Ajit Pal IIT Kharagpur

So, for reducing miss rate, you have to use some of the approaches which are listed. Here, number one is you can use larger cache size cache memory size, second is higher associativity third is large block size fourth is various compiler optimization techniques, so we shall discuss them one after the other.

(Refer Slide Time: 02:59)



Reducing Miss Rate: An Anatomy of Cache Misses

- To be able to reduce miss rate, we should be able to classify misses by causes (3Cs):
- Compulsory: To bring blocks into cache for the first time.
  - Also called cold start misses or first reference misses. Misses in even an Infinite Cache.
- Capacity: Cache is not large enough, some blocks are discarded and later retrieved.
  - Misses even in Fully Associative cache.
- Conflict: Blocks can be discarded and later retrieved if too many blocks map to a set.
  - Also called collision misses or interference misses. (Misses in N-way Associative, Size X Cache)

Ajit Pal IIT Kharagpur

Before we discuss this technique, let us focus on another very important aspect which will give you necessary background for reducing the various other techniques for miss rate. That is an anatomy of cache misses rather why cache misses occur what are the different types of cache misses that we can call as anatomy of cache misses. So, to be able to reduce miss rate, we should be able to classify misses by their causes why these misses occur. Based on the causes, the classification can be done into three broad categories, number one is known as compulsory misses. This compulsory misses is a compulsory misses occur because you have to bring blocks into cache for the first time as you know when you are turning the power on the cache memory is not containing any use full instruction on data.

So, the cache memory is containing kind of you may say garbage so for the first time you have to transfer instructions to the instruction cache and data to the data cache. So, this will this is inheritable and you cannot really avoid compulsory misses and these compulsory misses also called as cold start misses or first reference misses. That means first time when you are referring a particular block as you know the in the cache memory is referred in terms of blocks. That means will occur, so it is quite obvious that this misses will occur even when we have got infinite cache.

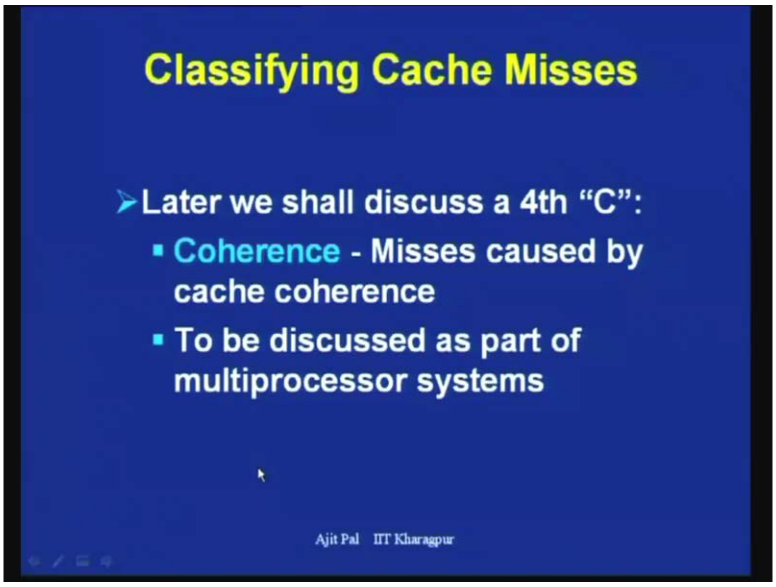
That means normally we know that if we have an infinite cache, then the miss rate will be very low. However, these compulsory needs will definitely occur because for the first time you have to transfer from the main memory to the cache memory. So, this this will happen, second is known as capacity miss cache is not large enough some blocks are discarded and later retrieved. So, this is due to smaller capacity of the caches so as we know because of cost we cannot have very large cache memory and as a consequence you have to you cannot really transfer all the blocks from the main memory to the cache memory.

So, you have to take only a subset of the blocks from transfer is subset of the blocks from main memory to the cache memory and obviously a whenever a particular block in the cache memory is mapped by a large number of blocks of the main memory. You have to replace that cache memory content and put a new block into it, so this will happen because some blocks are discarded and later retrieved misses occur even in fully associative cache. As you shall see, there are different types of associative, you can use and even when we use fully associative cache.

Then, this will happen that means you can place anywhere in the cache memory, so it is not fixed and even in this situation even in the scenario. This will occur capacity misses will occur last part not the list type of misses is due to conflict misses are known as conflict misses. So, blocks can be discarded and later retrieved if too many blocks map to a set so the conflict misses occur because as I mentioned you are you will be mapping many blocks to a particular set. As a consequence, you have to replace bring in and so on, so these are known as collisions misses or interferences misses.

So, these are the three types of misses that can occur and particularly misses in n way associative example as you keep on increasing the associativity and for a fix size cache, this factor will change, later on we shall discuss about it in more details.

(Refer Slide Time: 07:55)



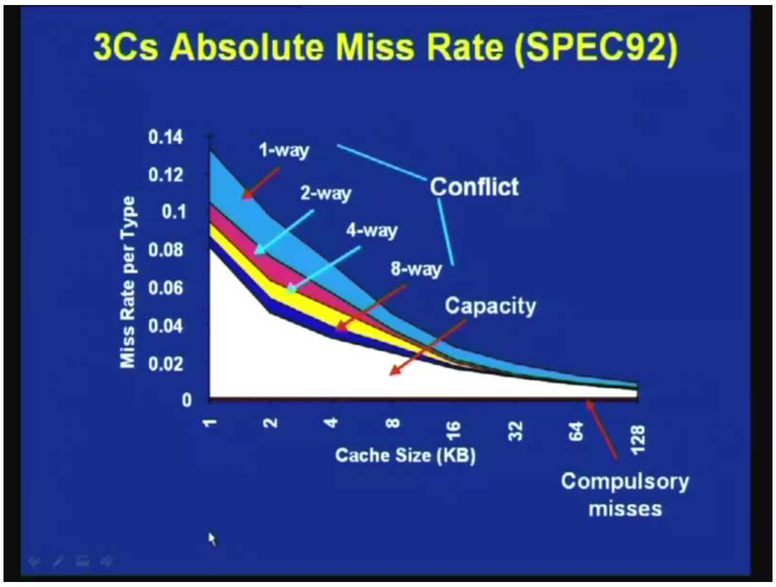
Classifying Cache Misses

- Later we shall discuss a 4th "C":
  - Coherence - Misses caused by cache coherence
  - To be discussed as part of multiprocessor systems

Ajit Pal IIT Kharagpur

Although I have broadly categorize the misses into three types or the misses can occur because of three reasons. There is another reason which may call as fourth c, so you have consider this is the fourth c, this is known as coherence misses this is caused by cache coherence. Later on, we shall be discussing about multiprocessor based systems, you have got multiple possessors having their own private caches and say at main say at memory main memory. So, in such situation another type of misses will occur and that that is known as coherence misses, so later on we shall when we shall be discussing about multi processors system this particular type of misses will be discussed in detail.

(Refer Slide Time: 08:54)



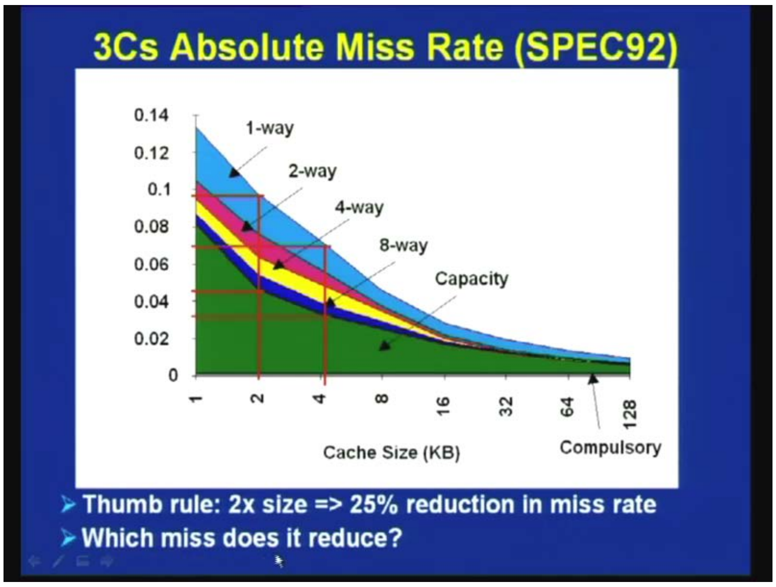
Now, let us consider the 3 C's and find let us see what the miss rate is for the different types of misses as you can see the bottom red line. I mean red that say very small portion here those are compulsory misses that means we find that the compulsory misses. I mean miss occur because of compulsory misses there are rate is much less compare to capacity and conflict misses. That means compulsory miss is insignificantly smaller compare to capacity and conflict misses.

This Watt portion corresponds to the capacity misses particularly even when you have got fully associative cache these are the miss rates. Obviously, as we increase the cache size as you can see the miss rate is decreasing and conflict misses are due to different types of associativity. That means capacity misses are because of the limited size of cache as you can see as you increase the size the miss rate will decrees because of capacity misses.

However, if you increase the associativity as you can see the miss rate decreases the top of portion that blue portion is due to I mean whenever you have got direct mapping that means your miss rate is maximum. Whenever you are using direct miss rate direct mapping and as you increase the associativity from one way to two way set as c t 4 way set as c t 8 way set c t as you can see the miss rate is gradually decreasing as represented by different colors.

So, that means the blue portion it corresponds one way that may set as c t that means direct mapping the red portion because of two way set associativity and the yellow one corresponds to four way set associativity. The blue one corresponds to eight way set associativity and the Watt portion as I have told even when you have got fully associative cache, there will be misses because of capacity misses.

(Refer Slide Time: 11:15)



3Cs Absolute Miss Rate (SPEC92)

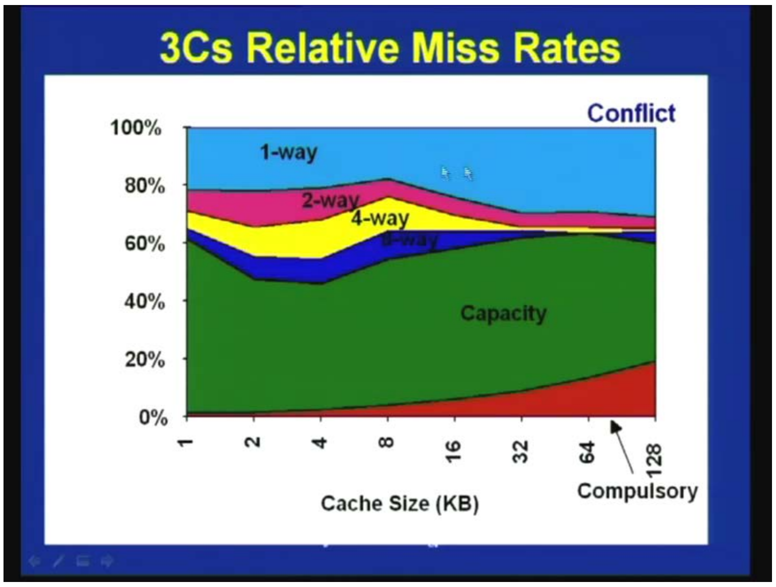
- Thumb rule: 2x size => 25% reduction in miss rate
- Which miss does it reduce?

Now, let us focus on how the miss rate decreases, I mean changes as you change the size for example, if the size of the cache is 2 kb, we find that miss rate is little more than 0.04 that is the miss rate. That means one out of that means above 4 percent, 4 percent cache misses occur when we have got 2 kilo Watt cache memory as you increase the size from 2 to 4 as you can see the miss rate is decreasing, it is little more than 3 percent.

So, little more than 4 percent to little more than 4 percent to little more than 3 percent, there is a decrease as you increase the size and thumb role is as you increase the size if you make the size double that means 2 kilo Watt to 4 kilo Watt, the miss rate decreases by 25 percent this corresponds to I mean this is corresponding to the full associativity. Now, let us consider whenever you have got direct mapping technique in such a case when you have got 2 kilo Watt cache memory, you can see the miss rate is little less than 0.1 that is less than 10 percent.

On the other hand, whenever you increase the size to 4 kilo Watt, then the miss rate reduces from little less than 10 percent to little more than 6 percent. So, it is roughly above 7 percent, so here also 25 reductions in miss rate occurs, obviously this reduction question arises due to which miss, and does it reduce? This reduction occurs because of capacity misses as I already mentioned in my earlier with the help of our earlier slides.

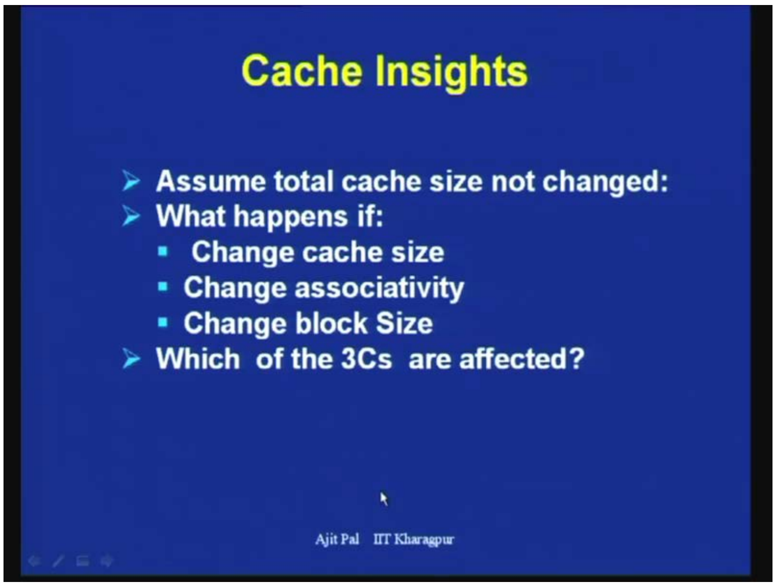
(Refer Slide Time: 13:21)



Now, earlier we have seen the different type of miss rates or different reasons compulsory capacity and the conflict misses and here the miss rate is shown and actually the absolute relative miss rates for different types some misses is shown here. So, we find here bulk of the misses occur because of capacity misses and as I mentioned earlier, the compulsory misses which is represented by the red portion, you can see the miss rate because of compulsory misses is quite small and an contrary to reduction in miss rate.

The cache memory size increases the compulsory misses increases because you have to transfer larger number of blocks from the main memory to cache memory. So, the percentage of compulsory misses in cases as you increase the size of the cache memory the capacity miss rate as you can see is remains from 0 to 60 percent. So, this is the capacity miss and derived for different types of different types of associativity, one way that is direct mapping 2 way, 4 way, 8 way, so for different types of associativity the percentage of misses are shown in this particular diagram.

(Refer Slide Time: 15:05)



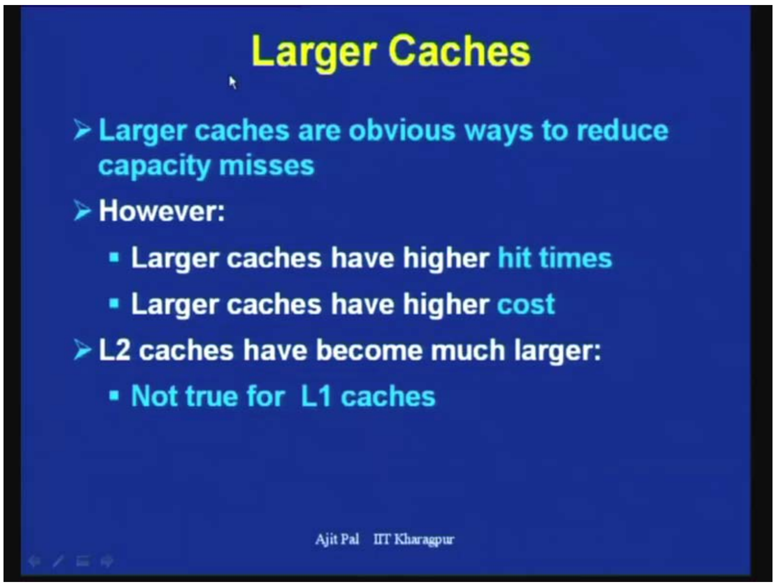
Cache Insights

- Assume total cache size not changed:
- What happens if:
  - Change cache size
  - Change associativity
  - Change block Size
- Which of the 3Cs are affected?

Ajit Pal IIT Kharagpur

Now, we have we can see that the miss rate can be reduced by controlling three parameters cache size associative and increasing the size of the block. Now, they affect the three types of misses that means 3 C s that we shall see and little more detail.

(Refer Slide Time: 15:34)



Larger Caches

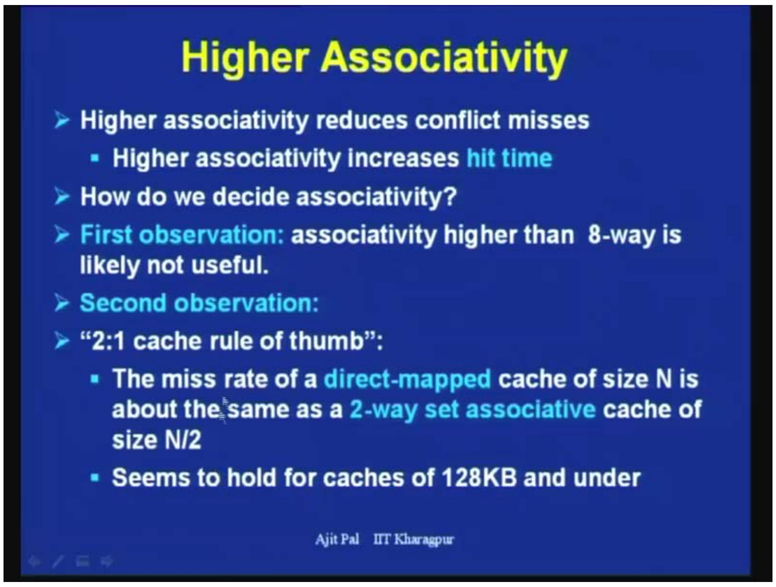
- Larger caches are obvious ways to reduce capacity misses
- However:
  - Larger caches have higher hit times
  - Larger caches have higher cost
- L2 caches have become much larger:
  - Not true for L1 caches

Ajit Pal IIT Kharagpur

So, first let us focus on larger caches sizes, we have seen as we increase the size of the cache memory the different types of misses changes. So, how the three type of misses are affected as you increase the size of the cache is consider here. So, larger caches are obvious ways to reduce cache capacity misses. So, capacity misses will reduce as we increase the size of the cache, however larger caches have higher hit times, so as we increase the size of the cache memory the cache memory becomes increasingly complex. As the increasing becomes increasingly complex, the coder portion will be quite complex and the delay for that access will be larger.

So, hit time will increase as you increase the size of the cache, so larger cache size have higher cost obviously as we as we put more and more cache memory the cost increases. So, these are the changes, so l 2 caches become larger not true for caches particularly whenever we consider l 1 cache, their hit time is very important parameters because most of the time data instruction will be read from the l 1 cache. So, they are try to keep the size quite small not very large, so hit time is very small however whenever we go for then size is a significantly larger compare to l 1 case.

(Refer Slide Time: 17:29)



Higher Associativity

- Higher associativity reduces conflict misses
  - Higher associativity increases hit time
- How do we decide associativity?
- First observation: associativity higher than 8-way is likely not useful.
- Second observation:
- "2:1 cache rule of thumb":
  - The miss rate of a direct-mapped cache of size N is about the same as a 2-way set associative cache of size N/2
  - Seems to hold for caches of 128KB and under

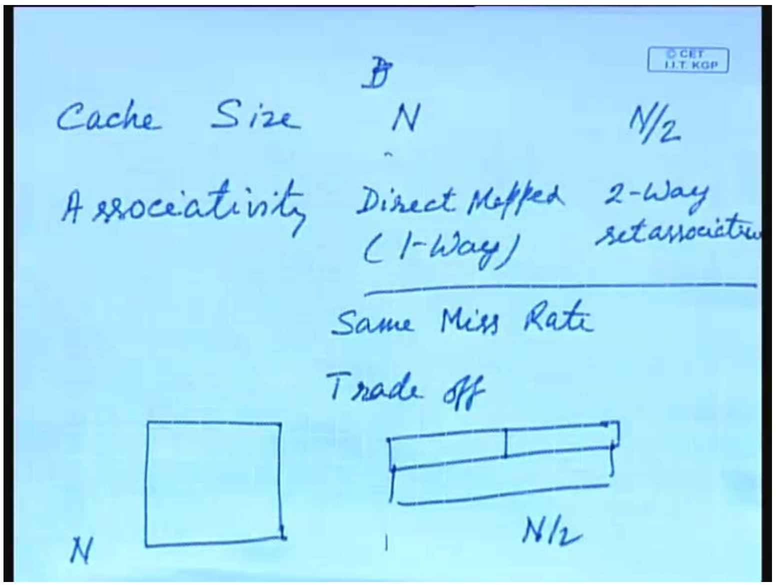
Ajit Pal IIT Kharagpur

Second important parameter as i told is higher associativity higher associativity reduces conflict misses as you have seen in the diagram. If we go for fully associative cache, and then the conflict miss will be minimum and as you and I mean reduce the associativity go towards direct mapping in hit rate increases so higher associativity increases hit time. So, the reason for that is whenever you go for higher associativity cache becomes more complex.

So, direct mapping is the simplest type of cache memory you can use conventional memory as the cache memory whenever you use direct memory. Whenever you go for two ways set associativity or four ways set associativity or fully associativity, then the cache memory becomes increasingly complex. As a consequence, the hit time will increase as it increase the associativity and another observation that we have seen from the diagram associativity higher than eight way is likely not useful.

We have seen in our previous diagram after eight ways set associativity there is no significant reduction in miss rate. If you increase the associativity, so there is no point in going beyond eight way set associativity if we incorporate associativity in the cache memory eight ways, eight ways are limit. Beyond that, it does not we reach a point of no returns beyond that there is no benefit that is the reason why the cache the associativity restricted to restricted of to eight way set of associativity. Second observation is then is that there is a 2 2 1 cache rule thumb, what is that the miss rate of a direct mapped cache of size n is about the same as a two way set associative cache of size n by 2.

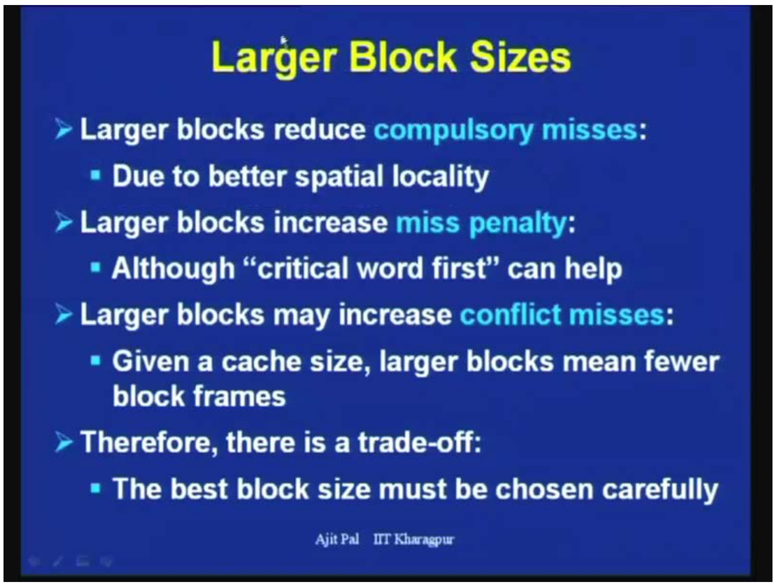
(Refer Slide Time: 19:46)



That means one parameter is cache memory size another is associativity, suppose the cache size is n and if it is direct map say associativity is direct map or we can say one way of set associative. This will give the same miss rate whenever you have got cache size of n by 2 and you have got 2 way set associative. So, we find that there we can achieve the same goal that means we can have the same miss rate either by increasing the associativity and reducing the cache size. What we can do we can go for direct mapping doubling the size from two way set associative.

So, these are you can say tradeoff, tradeoff between associativity and cache size for the same miss rate and this particular thumb rule actually holds for caches of 128 kb and obviously beyond 128 kb, we do not, I mean consider a cache memory size. So, this particular thumb rule is applicable cause for cache memory of 128 kb and under a smaller.

(Refer Slide Time: 21:42)



Larger Block Sizes

- Larger blocks reduce compulsory misses:
  - Due to better spatial locality
- Larger blocks increase miss penalty:
  - Although "critical word first" can help
- Larger blocks may increase conflict misses:
  - Given a cache size, larger blocks mean fewer block frames
- Therefore, there is a trade-off:
  - The best block size must be chosen carefully

Ajit Pal IIT Kharagpur

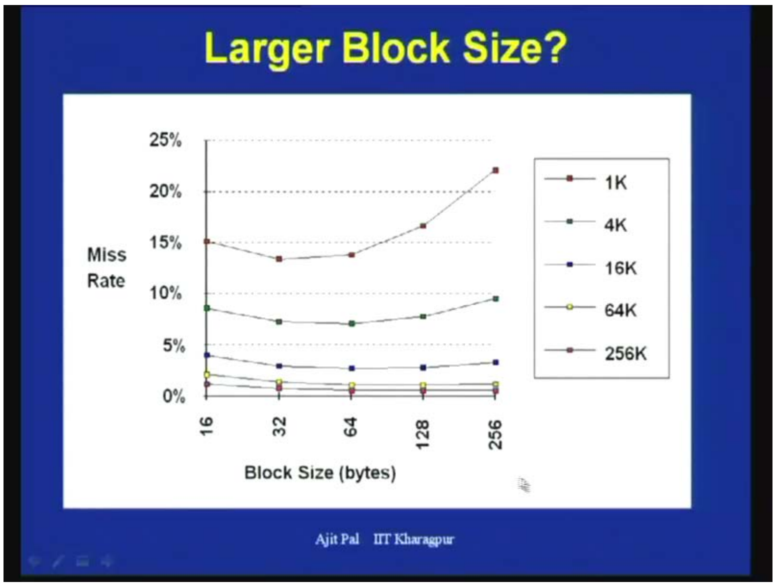
Now, let us consider the block size so larger blocks size reduces compulsory misses, so compulsory misses will be reduced as you increase the block size. That is because of spatial locality you know this this is governed by the locality of references as you increase the blocks size larger block size the adjusting block adjust in words are taken from the cache main memory to the cache memory. As consequence, the next reference whenever the next, I mean consecutive addresses is referred it will be available in the cache memory. So, that is the reason why increasing the block size compulsory misses reduces because of better spatial locality.

So, however larger blocks size increases miss penalty miss penalty increases the reason for that is whenever a cache miss occurs. If you have got multiple words in the cache memory, then you have to transfer all of them before you can resume execution of a particular instruction by the processes. That means before the control is transferred to the processor, you have to transfer all the words of a block from the main memory to the cache memory. That means that miss penalty increases however this can be problem can be reduce to a great extent by using critical word first. That means all though we shall be transferring all the words of a block, but the critical word which has been referred by the processor is transferred first.

So, the processor can resume its operation only after the critical word is transferred from the main memory to the processor also to the cache and subsequently other words can be transferred. So, this can be done which we shall be discuss in my next lecture where shall be consider various techniques for reducing miss penalty. So, this will be one of the technique and larger blocks may increase conflict misses, you see all though it reduces compulsory misses the conflict misses increases. The reason for that is whenever you consider a cache memory of same size without changing the size of the cache memory.

If we increase the associativity then what happens then for a given larger blocks means fewer block frames that means the number of suppose for the same size the number of same size n for direct mapping. If we go for two ways set associative, although we have got two words in a same block, but the number of frames reduces. It will become n by 2, so as a consequence the number of frames reduces. That will lead to increase in conflict misses, therefore there is a trade off the best block size must be chosen carefully. So, you have to see there are conflicting outcomes that means it reduces compulsory misses increase miss penalty and also it increase conflict misses.

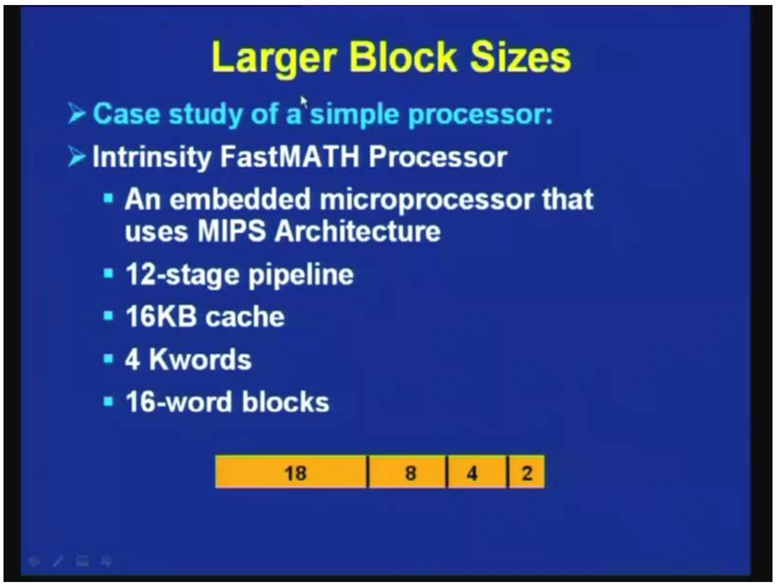
(Refer Slide Time: 25:28)


Larger Block Size?

So, you have to very judicially choose the size of the block as it is evident from this diagram you can see here as you increase the block size miss rate initially decreases, but ultimately after reaching a minimum value. It again increases that means particularly when the size of the cache memory is small, then we should use a relatively smaller block size in other words the block size as should never be comparable to the size of the cache memory.

So, when the size of the cache memory is large as you can see then if you increase the block size it does not affect much that means when the size of the cache memory is small it is affected more. If we increase the size of the block, but whenever we got a large cache memory by increase the size of the block it as a it will it does not affect much. I mean the miss rate is not affected much, so what we can say that the tradeoff is best on the available size of the cache memory, you have to decide about the size of the block.

(Refer Slide Time: 26:41)



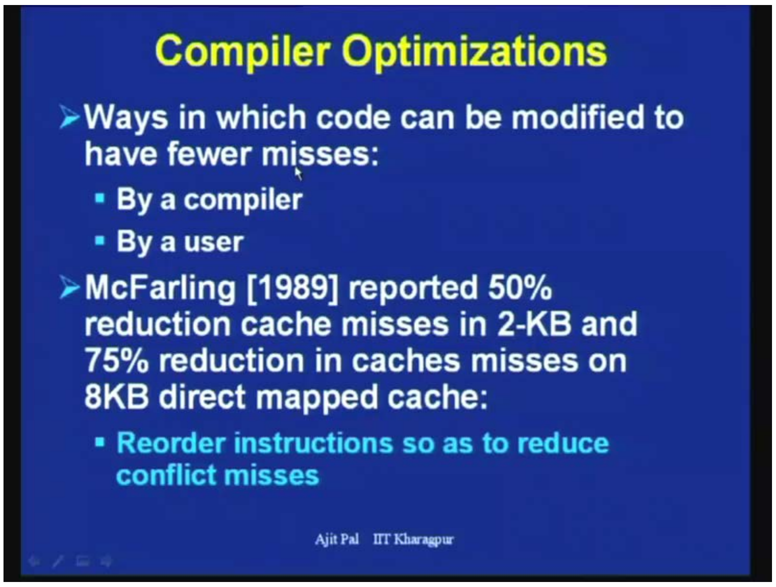
Larger Block Sizes

- Case study of a simple processor:
- Intrinsity FastMATH Processor
  - An embedded microprocessor that uses MIPS Architecture
  - 12-stage pipeline
  - 16KB cache
  - 4 Kwords
  - 16-word blocks

18	8	4	2

So, this is simple, I mean case study a real life processor intensity fast math processor that is a that is an embedded microprocessor that is that uses miss architecture. This particular processor is used in many embedded application, it has got 12 stage pipeline 16 kb kilo Watt cache memory each of 4 kilo words. I mean four that 16 kb cache means 4 kilo words and 16 word blocks. So, you can see here you have got a block size is quiet large because each block is containing 16 words. These are the different components; I mean this is the address 32 bit address which is divided into different parts we can see.

(Refer Slide Time: 27:34)



Compiler Optimizations

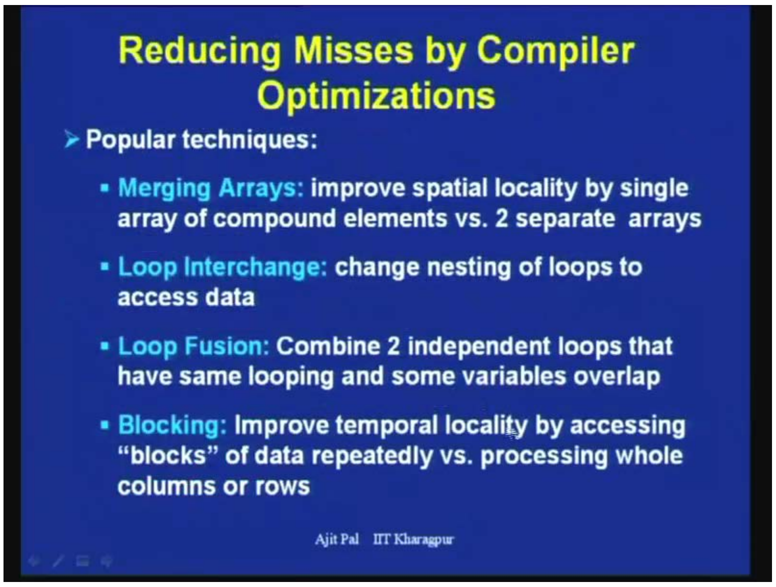
- Ways in which code can be modified to have fewer misses:
  - By a compiler
  - By a user
- McFarling [1989] reported 50% reduction cache misses in 2-KB and 75% reduction in caches misses on 8KB direct mapped cache:
  - Reorder instructions so as to reduce conflict misses

Ajit Pal IIT Kharagpur

Now, we shall focus on various compiler optimization techniques, so ways in which code can be modified to have fewer misses that means so far what we have discussed. Those are to be implemented in hardware and programmers or compiler writers and not bother about that. Either you will increase the size of the cache or you will increase the associativity or increase the size of the block. So, all these things are related to changing hardware and the programmers are very happy with that, but now we shall discuss some techniques which are which are which are actually based on optimization of the compiler.

This optimization can be either be done by the compiler or by the user themselves and McFarling reported back in 1989 that 50 percent reduction in cache misses using 2 kilo byte cache. That can take place by using compiler optimizations and 75 percent reduction in cache misses can occur on 8 kilo Watt direct mapped cache by using by adopting compiler optimization techniques. Essentially, what that what is being done re order instructions so as to reduce conflict misses. So, primarily by reducing the conflict misses because size has been kept fixed simply by reordering the conflict misses are reduce and that leads to reduction in I mean miss cache, so we shall discuss several techniques compiler optimization techniques first one.

(Refer Slide Time: 29:26)



Reducing Misses by Compiler Optimizations

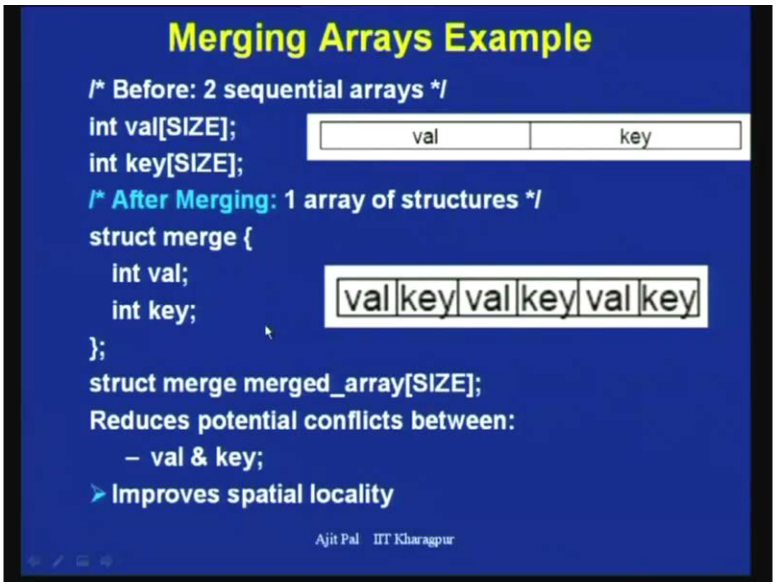
- Popular techniques:
  - Merging Arrays: improve spatial locality by single array of compound elements vs. 2 separate arrays
  - Loop Interchange: change nesting of loops to access data
  - Loop Fusion: Combine 2 independent loops that have same looping and some variables overlap
  - Blocking: Improve temporal locality by accessing "blocks" of data repeatedly vs. processing whole columns or rows

Ajit Pal IIT Kharagpur

These are the most popular techniques number one is merging arrays, we shall see how you can merge more than arrays into one and that will improve spatial locality by simple array of compound element verses two separate arrays that we discuss in detail. Second is loop interchange by change nesting of loops to access data, so there will be no change in the number of instruction to be executed. Simply by changing the order of the nesting of loops, you will be able to reduce essentially that will increase the conflict miss. Then, third technique is known as loop fusion, so we may be having two independent loops that have the same looping and same variable overlap.

So, the key parameter is that you are using some variables which are common in two different loops. So, instead of accessing separately in two different loops, you can have a one loop and those once variable is transferred from the cache memory to main memory to the cache memory. They will be used for both the loops, so that is known as loop fusion the last technique is known as blocking. This improves temporal locality by accessing blocks of data repeatedly verses processing whole columns or rows particularly processing arrays this will be very important.

(Refer Slide Time: 31:14)



Merging Arrays Example

```
/* Before: 2 sequential arrays */
int val[SIZE];
int key[SIZE];
/* After Merging: 1 array of structures */
struct merge {
  int val;
  int key;
};
struct merge merged_array[SIZE];
```

Reduces potential conflicts between:

- val & key;

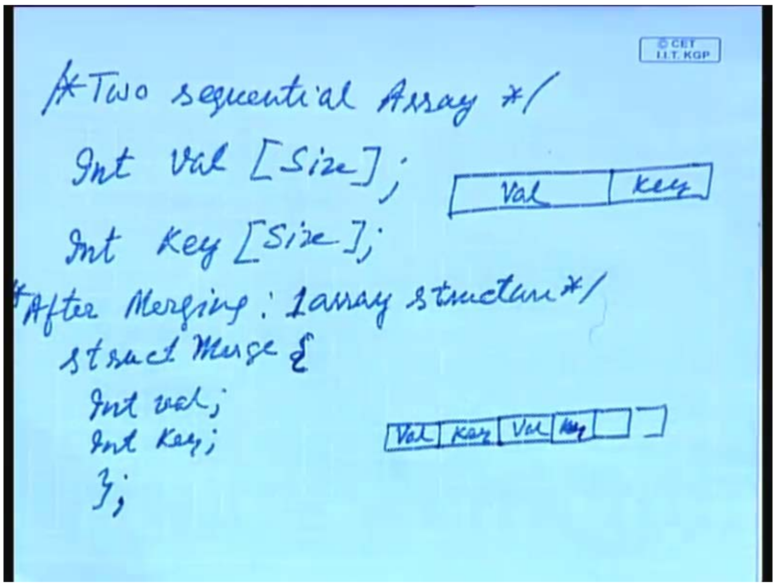
➤ Improves spatial locality

val
key
val
key
val
key
val
key

Ajit Pal IIT Kharagpur

So, these techniques we shall discuss one after the other, so this is the first technique margining arrays you can see here we have got two different arrays.

(Refer Slide Time: 31:29)



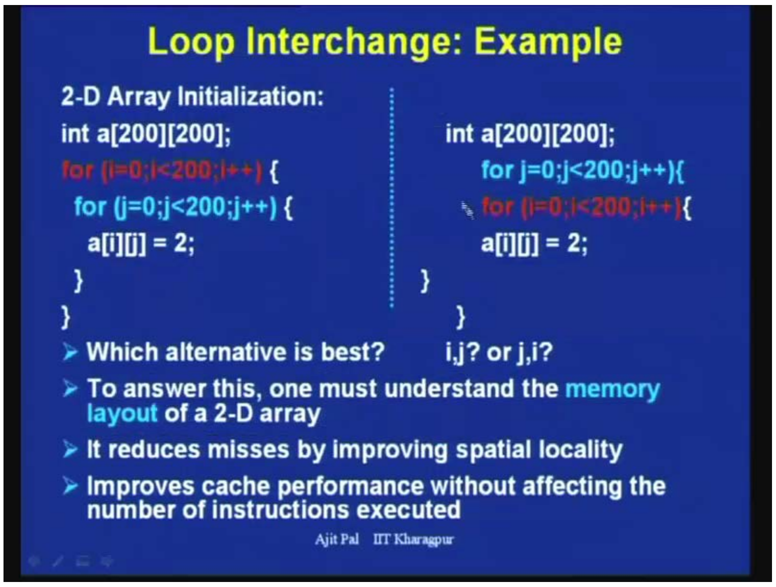
```
/*Two sequential Array */
 Int Val [Size];
 Int Key [Size];
/*After Merging: 1array structure */
 struct Merge {
  Int val;
  Int Key;
 };
```

Val
Key
Val
Key
Val
Key
© CET I.I.T. KGP

First one is two sequential arrays, one is two sequential arrays, so one is your interval is a size of the arrays given here and another is another array that is your key. So, these two are these two arrays are access separately and you can see the order in which they where it is access first you are accessing the array and then you are accessing key one after the other. However, after merging having one array structure what you can do you can strut merge int val and then int key.

So, in this particular case what will be done you will be accessing an element of val, then element of key and element of val element of key in this way will be accessing an as a consequence, what will happen the while doing this? The miss rate will decreases because the way it is done, so it reduces potential conflicts between val and key and because it improves spatial locality. So, because of improves spatial locality this will give you better result and the way will be this way you know that you will be storing in this way. Then, accessing in this manner and that will improve the performance because it improves spatial locality.

(Refer Slide Time: 34:08)



Loop Interchange: Example

2-D Array Initialization:

```
int a[200][200];
for (i=0;i<200;i++) {
  for (j=0;j<200;j++) {
    a[i][j] = 2;
  }
}
```

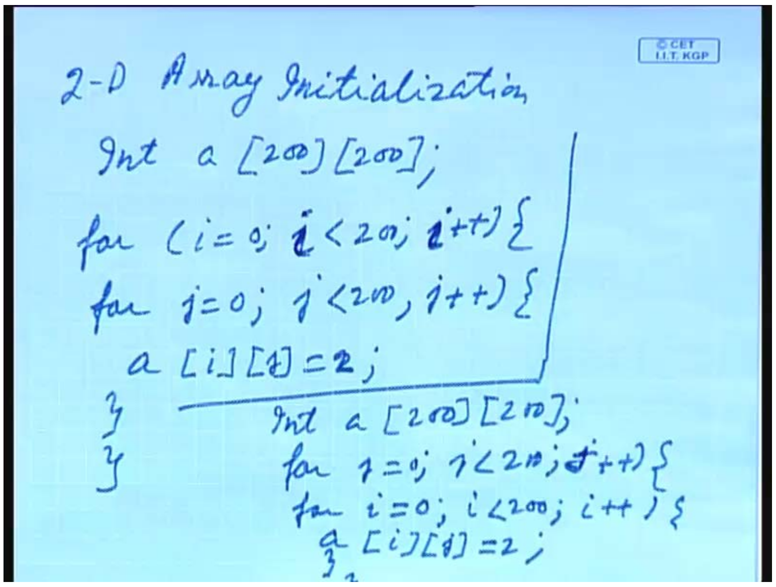
```
int a[200][200];
   for j=0;j<200;j++){
   for (i=0;i<200;i++){
   a[i][j] = 2;
 }
   }
```

- Which alternative is best? i,j? or j,i?
- To answer this, one must understand the memory layout of a 2-D array
- It reduces misses by improving spatial locality
- Improves cache performance without affecting the number of instructions executed

Ajit Pal IIT Kharagpur

Second technique that we shall be using is known as loop interchange a in case of loop interchange.

(Refer Slide Time: 34:20)



2-D Array Initialization

```
int a [200][200];
for (i=0; i<200; i++){
 for j=0; j<200, j++){
  a [i][j] =2;
 }
}
```

```
int a [200][200];
for j=0; j<200; j++){
 for i=0; i<200; i++){
  a [i][j] =2;
 }
```

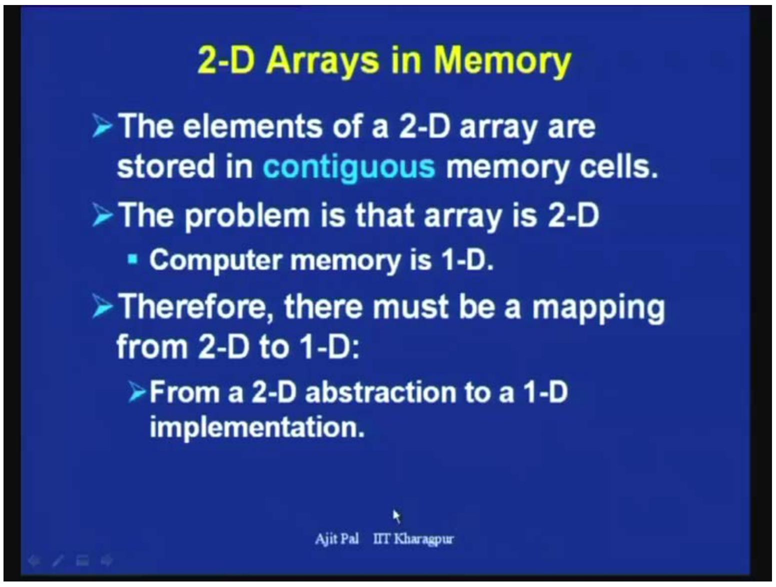
You have got 2 D array initialization so int a 200 elements 200 so it is a two dimensional array and for then you are accessing this manner for i is equal to 0 j less than 200 j plus and for j is equal to 0 j it is i i j less than 200 and j plus. So, then you are writing into it a i j and is equal to 2, so here what it will be done you will be in this case first you are reading a value taking a value of i and you increasing the value of j from 0 to 200 and you are writing into it.

So, that means the rate is done is a for one i you are accessing all elements of j and then you are taking another element of i all accessing all the elements of j. So, this is one you are doing it and alternatively what can be done instead of doing this, you can do this way int a 200, 200 you are you will simply change the order for j. Earlier, it was i for a, here it is j is equal to 0, j is less than 200 j plus and then for simple you have change the order see here it is i is equal to 0 i less than 200 i plus plus and then a i j is equal to 2. So, in this case what you have done you have simply the earlier j was nested in i, now i is nested in j, so because of this you know there will be lot of changes.

Here, question arises which one give you better result actually to answer this one must understand the memory layout of the 2 D array the way it is stored in the memory because your memory is one dimensional your accessing a two dimensional array. That is stored in a one dimensional memory as you know memory is organize in the linear way and that is access by the address. So, what you can do, it reduces misses by improving spatial locality whenever you change the order and improves cache performance without affecting the number of instructions executed.

So, the number of instructions executed will will be will be independent of what way you do so that means in both the cases number of instructions executed is same, but the way it is stored. I mean the way it is stored here because the array corresponding to that i th you know i is in i to increasing from 0 to 200. In fact, it has been stored sequentially corresponding to i and as a consequence, it will give you better spatial locality.

(Refer Slide Time: 38:39)



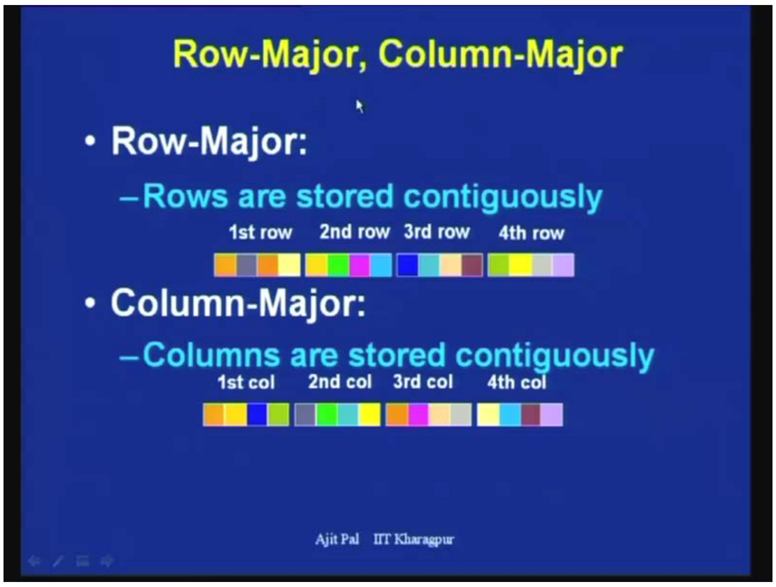
2-D Arrays in Memory

- The elements of a 2-D array are stored in contiguous memory cells.
- The problem is that array is 2-D
  - Computer memory is 1-D.
- Therefore, there must be a mapping from 2-D to 1-D:
  - From a 2-D abstraction to a 1-D implementation.

Ajit Pal IIT Kharagpur

That means the second option will give you better option better reduction that means the element of 2 D array are stored in contiguous memory cells. The problem is that as I told in computer memory is in one due a, therefore there is a there must be a mapping from 2 D to 1 D. So, from 2 D abstraction to 1 D implementation, the consequence this particular approach will give you better result.

(Refer Slide Time: 39:11)



Row-Major, Column-Major

- **Row-Major:**
  - Rows are stored contiguously

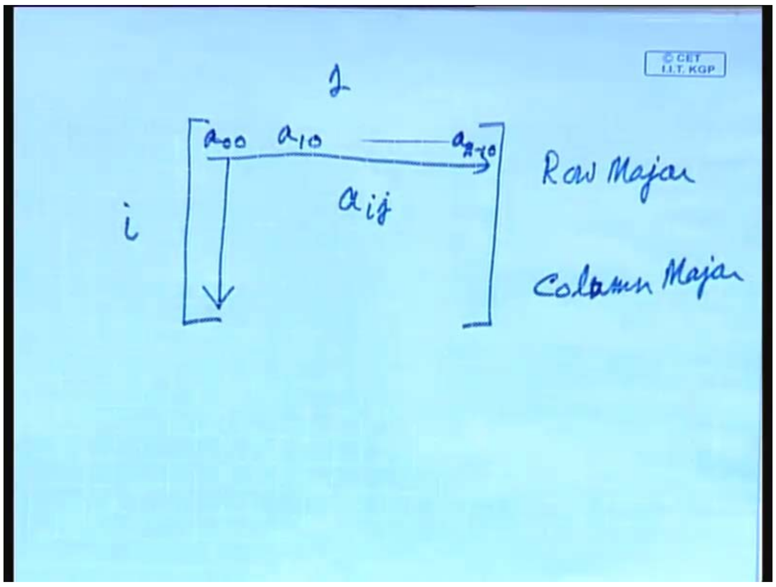
1st row
2nd row
3rd row
4th row

- **Column-Major:**
  - Columns are stored contiguously

1st col
2nd col
3rd col
4th col
Ajit Pal IIT Kharagpur

You must understand the way the rows and column the way they are stored in the memory, so you have got a two dimensional array.

(Refer Slide Time: 39:26)

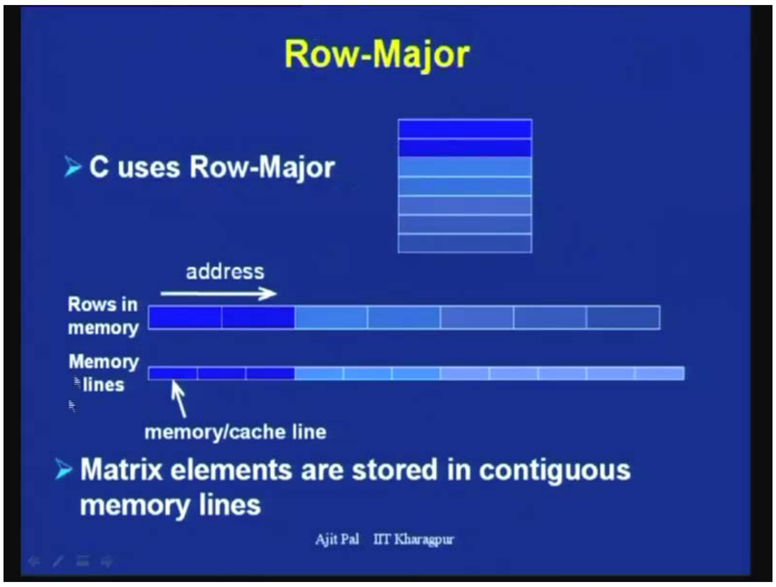


Here, it is i and j and here you have got different elements of a i j, so these are the rows and these are the columns. So, here it is a say 0 and this is the a 0 0, then here the row changes. So, 1 j is equal to 0 and so on and a n minus 1 j is 0. So, first row the first row and different columns and in this direction, you have got different columns. Now, the way they can be stored is one is known as row major second is known as column major.

That means if the rows are stored contiguously, if is the row elements are stored contiguously in this manner, then it is called row major.

On the other hand, if you stored in this manner this element, then the next elements of the column, then it is called column major. So, you can see the column major are stored contiguously first column, then second column, then third column, then forth column this is column major and this is row major.

(Refer Slide Time: 41:06)

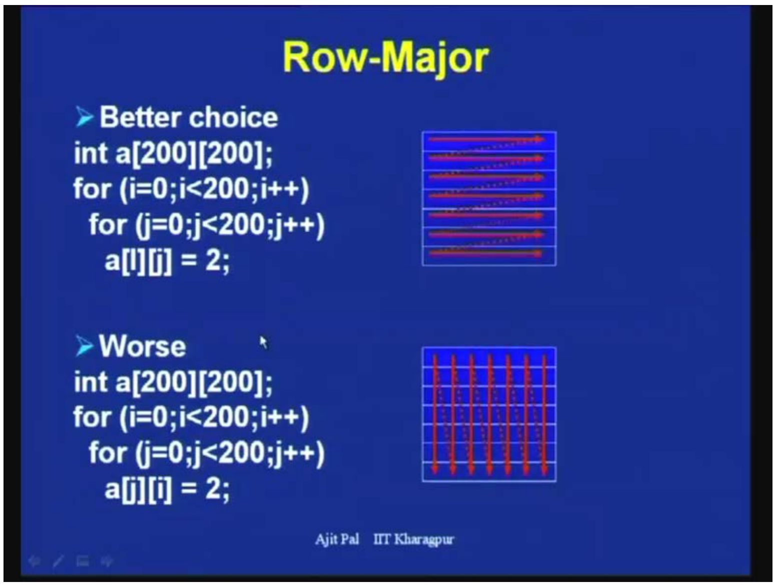


Row-Major

- C uses Row-Major
- Matrix elements are stored in contiguous memory lines

Actually the c program that is c uses row major that means you are storing in terms of rows. First, you are storing these rows, then this row then this row and this row as you can see rows in memory as you increasing the address different row elements are there. These are the different memory lines in of the cache memory, so matrix elements are stored in contiguous memory lines and they are stored in a row major way.

(Refer Slide Time: 41:39)



Row-Major

- Better choice

```
int a[200][200];
for (i=0;i<200;i++)
  for (j=0;j<200;j++)
    a[i][j] = 2;
```

- Worse

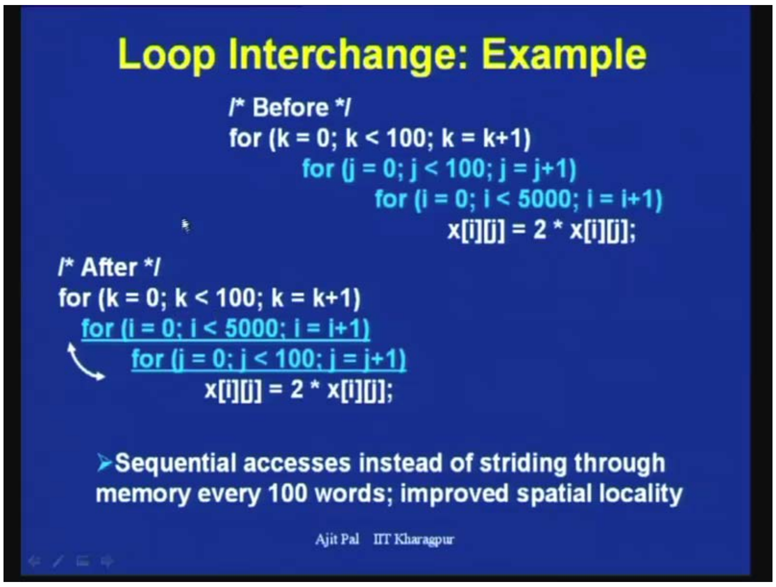
```
int a[200][200];
for (i=0;i<200;i++)
  for (j=0;j<200;j++)
    a[j][i] = 2;
```

Ajit Pal IIT Kharagpur

As a consequence, whenever you try to access them in the row by in terms of row elements with the help of this program. Then, you get good result on the other hand if it is stored in a different way that means the column measure way. Then, this program will not give you good result as you can see there will be that locality of reference that spatial locality is lost, since they are stored in row way.

You are accessing in terms of column and as a result this will be this will give you worst result so the locality of reference a spatial locality will be much less. As a result, there will be many misses that will occur in this case compare to the first case where your accessing row by row and does not elements are stored in the row measurement.

(Refer Slide Time: 42:41)



Loop Interchange: Example

```
              /* Before */
              for (k = 0; k < 100; k = k+1)
                     for (j = 0; j < 100; j = j+1)
                            for (i = 0; i < 5000; i = i+1)
                                   x[i][j] = 2 * x[i][j];
/* After */
for (k = 0; k < 100; k = k+1)
  for (i = 0; i < 5000; i = i+1)
       for (j = 0; j < 100; j = j+1)
              x[i][j] = 2 * x[i][j];
```

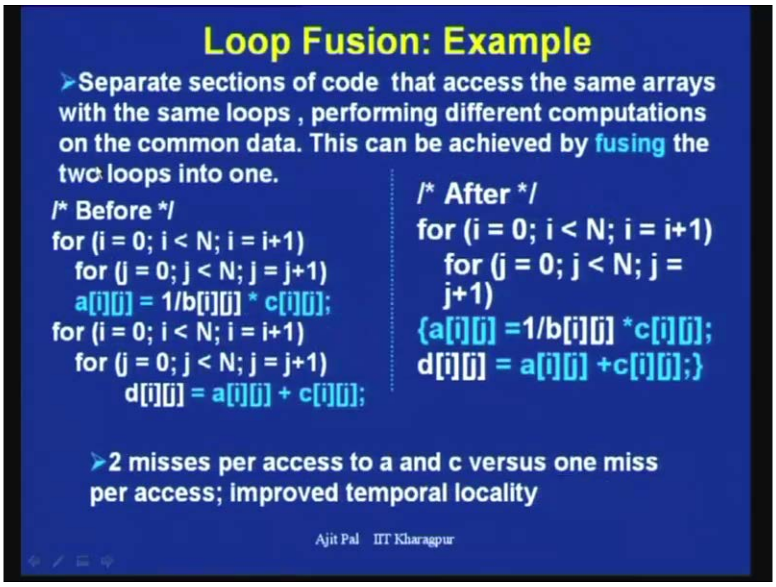
➤Sequential accesses instead of striding through memory every 100 words; improved spatial locality

Ajit Pal IIT Kharagpur

Another technique is known as loop interchange, so here what has been done, you can see you have got this is this is the your array k is equal to 0 200 and here j is equal to 0 to 100 and i equal to 0 to 5000. That means you are accessing different elements of x i j your writing in 2 and doing multiply with your multiplying with 2. Then, you are storing it here, now if you interchange i and j, so if you interchange i and j and then say for outer one remaining same that k is equal to 0 to 100. Here, it is i equal to 0 to 5000 and lower it is j equal to 0 to 100 and x i j is equal to 2 into 2000.

So, the sequential accesses instead of striding through memory every 100 words, so, this will give you sequential access of for different element of the array instead of striding through memory every 100 words. In this case, you will be striding 200 every 100 words, but in the second case will be accessing sequentially and this will give you improved spatial locality. So, if you perform this loop interchange this the second option will give you better result because of include spatial locality compare to the first one so this is the loop interchange example.

(Refer Slide Time: 44:20)



Loop Fusion: Example

- Separate sections of code that access the same arrays with the same loops , performing different computations on the common data. This can be achieved by fusing the two loops into one.

```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
      d[i][j] = a[i][j] + c[i][j];
```

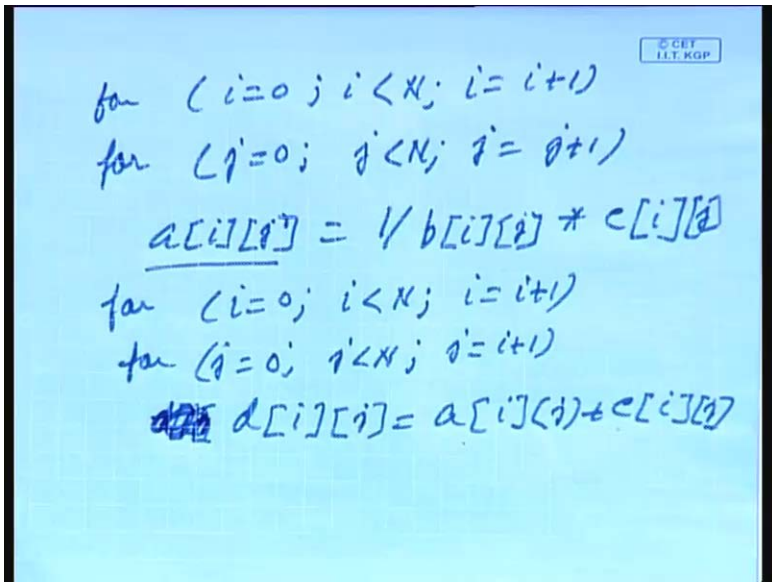
```
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j =
  j+1)
{a[i][j] =1/b[i][j] *c[i][j];
d[i][j] = a[i][j] +c[i][j];}
```

- 2 misses per access to a and c versus one miss per access; improved temporal locality

Ajit Pal IIT Kharagpur

Then, third is your loop fusion example, so in this case separate sections of code that access the same arrays with the same loops performing different computations on the common data. As I mentioned earlier, here you are having two different loops one here the first loop is that we write down the two different loops.

(Refer Slide Time: 44:47)

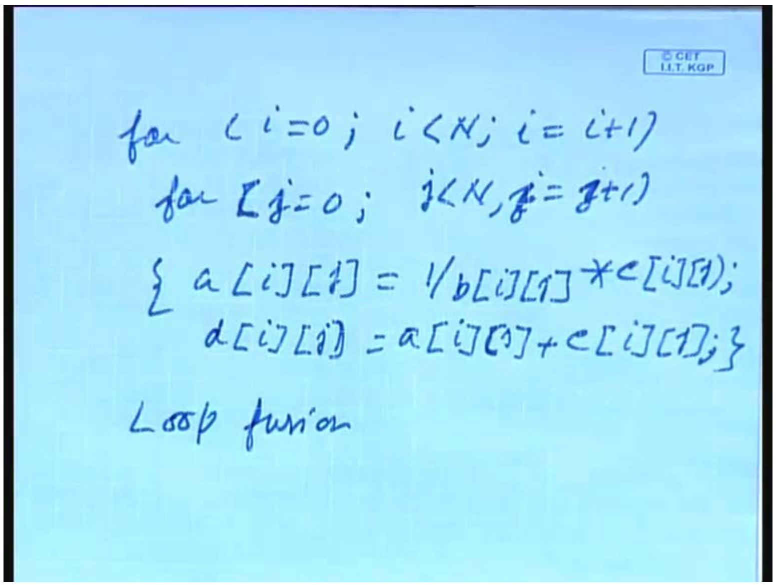


```
for ( i=0 ; i<N; i= i+1)
for (j=0;  j<N; j= j+1)
  a[i][j] = 1/ b[i][j] * c[i][j]
for (i=0; i<N; i= i+1)
 for (j=0;  j<N; j= i+1)
   d[i][j]= a[i](j)+c[i][j]
```

So, first one is for i is equal to 0 i less than n i is equal to i plus 1 and for j is equal to 0 j less than n j is equal to j plus 1 and a i j is equal to 1 by b i j into c i j. So, this is the operation you are doing and you are performing in this manner. This is another loop you have got for i is equal to 0 i less than n i is equal to i plus 1 and for j is equal to 0 j less than n j is equal to i plus 1 and you are performing d i j, d i j is equal to a i j plus c i j. So, you can see your performing two different computations, one is your this computation where a i that elements of a being access element of b being access element of c is being access and you are performing is computation.

Here is another computation, where you are you are computing i mean d i j and also your accessing different elements of a and c. So, you find that elements of a and c which your accessing are overlapping although your performing two different computation in two different loops instead of doing it a having two different loops. What you can do, you can merge into a you can have a single loop.

(Refer Slide Time: 47:17)



```
for (i=0; i<N; i=i+1)
  for [j=0;  j<N, j=j+1)
  { a[i][j] = 1/b[i][j]*c[i][j];
    d[i][j] = a[i][j]+c[i][j]; }
```

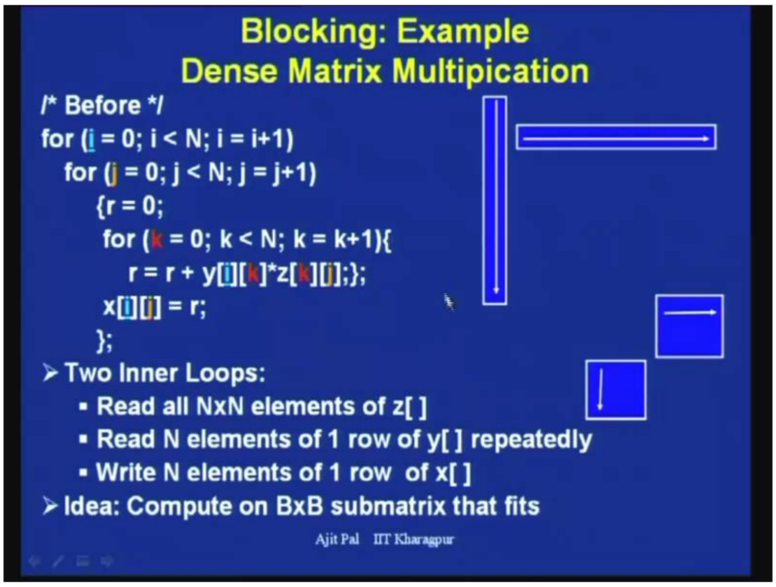
Loop fusion

For example i is equal to 0 i less than n i is equal to i plus 1 and then you have got to another j is equal to 0 j is less than n i j equal is to j plus 1 and you are performing a i j is equal to one by same thing which you have done earlier 1 by a b i j into c i j. Also, you will do this computation d i j is equal to a i j plus c i j. So, here instead of doing in two different loops your performing in a single loop, so as a consequence you are performing loop fusion and by doing this loop fusion, you will be having better result because two misses per access.

Whenever you access them separately verses for to a and c that means your accessing a and c which are common instead of two misses only one miss per access is occur whenever you do by using this loop fusion. So, this two misses per access will be reduced to one misses per access because of improved temporal locality as you know temporal locality your accessing something your retaining something

This will be needed in near future in fact that is what is happening in these two cases, so processing these this particular first computation and second computation you are accessing once transferring to the cache memory and reusing twice. So, this will improve temporal locality and give you better result.

(Refer Slide Time: 49:55)



Blocking: Example
Dense Matrix Multipication

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1){
          r = r + y[i][k]*z[k][j];};
       x[i][j] = r;
      };
```

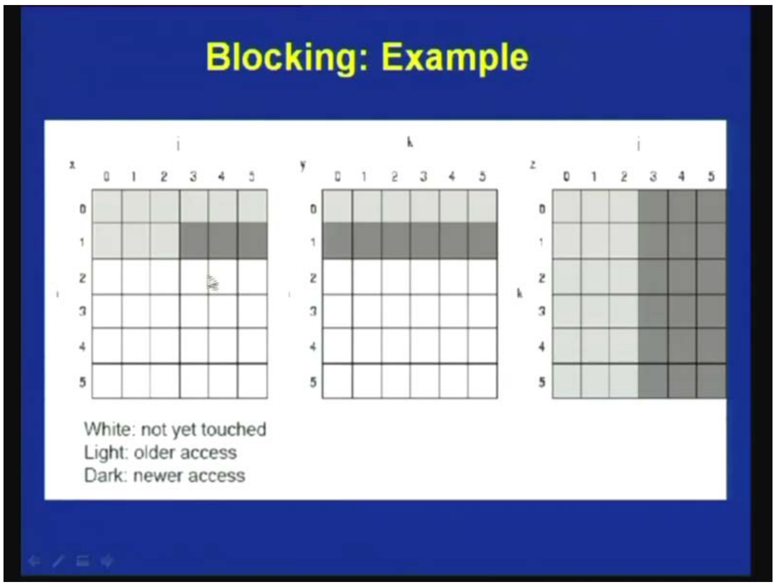
- Two Inner Loops:
  - Read all NxN elements of z[ ]
  - Read N elements of 1 row of y[ ] repeatedly
  - Write N elements of 1 row of x[ ]
- Idea: Compute on BxB submatrix that fits

Ajit Pal IIT Kharagpur

Then, the final technique that we shall discuss known as blocking, so whenever you are performing dense matrix multiplication this is the code which will be writing. So, i is equal to 0 colon i less than n i equal to i plus 1 for j is equal to 0 to j less than n j is equal to 1 r is equal to 0 for and you will be doing this computation for k 0 to k is equal to 0 k less than 1 k is equal to k plus 1. Then, r is equal to r plus y i k into z k j and also you will be doing x i j equal to r. So, this will do and whenever as you can see you have got two inner loops and you are reading all n by n elements of z reading all on all element of z and read n element of one row of y repeatedly.

So, this will be doing repeatedly because this inner loop and write n element and instead of doing that. That means instead of reading the entire rows entire column rows and entire columns what you can do you can compute on b by b sub matrix that fit into the cache memory. So, in this particular case it will not fit into the cache memory that means entire all the elements of row and all the elements of column will not fit sequence will be having lot of misses.

(Refer Slide Time: 51:56)



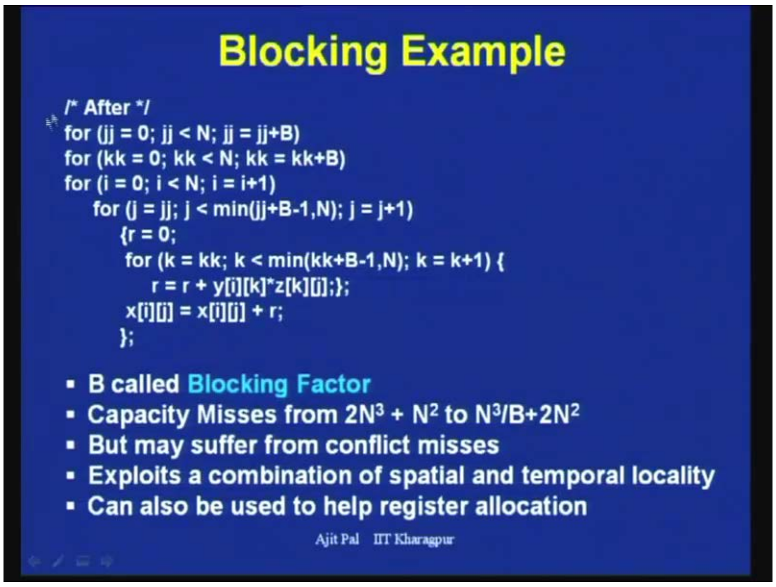
Blocking: Example

White: not yet touched
Light: older access
Dark: newer access

So, instead of doing this way you will be dividing into blocks of smaller size as it is shown here smaller blocks and the take the way it is happening shown in this particular diagram. So, here as you can see when you are doing without blocking, so when without doing whenever you are computing this is the x array this is the y array and this is the z array. The way there are accessed is shown the help of this diagram that Watt portion is not yet not yet touched.

Then, the light portion is corresponding to older access and the dark portion is the newer access this portion. That means you can see the older accesses I have taken place, but using only a small portion here. Similarly, for the for array y again your accessing your using only a small portion and this is the older access. Similarly for j you can see a large portion has been is corresponding to older access and this is the dark portion corresponding the new access instead of doing this.

(Refer Slide Time: 53:06)



Blocking Example

```
/* After */
for (jj = 0; jj < N; jj = jj+B)
for (kk = 0; kk < N; kk = kk+B)
for (i = 0; i < N; i = i+1)
    for (j = jj; j < min(jj+B-1,N); j = j+1)
        {r = 0;
         for (k = kk; k < min(kk+B-1,N); k = k+1) {
             r = r + y[i][k]*z[k][j];};
         x[i][j] = x[i][j] + r;
        };
```

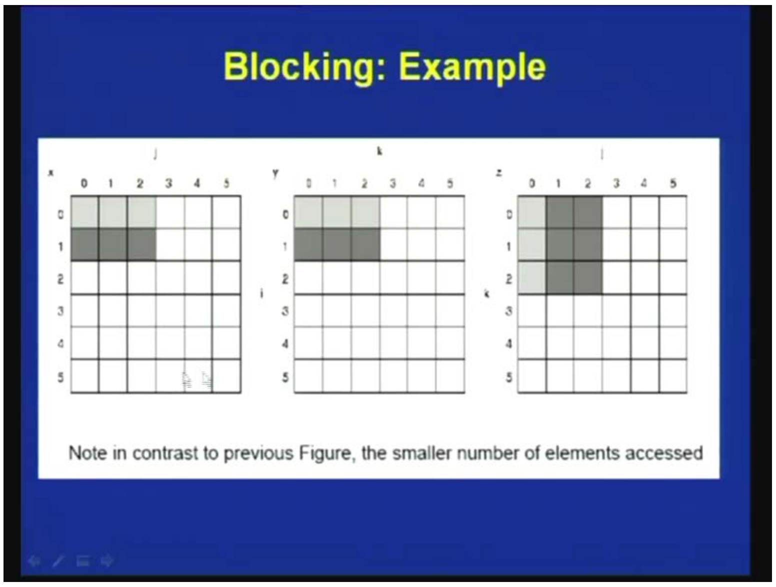
- B called Blocking Factor
- Capacity Misses from $2N^3 + N^2$ to $N^3/B+2N^2$
- But may suffer from conflict misses
- Exploits a combination of spatial and temporal locality
- Can also be used to help register allocation

Ajit Pal IIT Kharagpur

If you use the blocking and by changing the code here as you can see you have you have used a blocking factor b called divided into blocks. So, instead of doing the computation of the entire arrays and entire rows and columns you are doing block by block and this is how they are different. I mean the different access of arrays done is the show with the help of different loops and the capacity misses because of this blocking the capacity misses reduces from 2 to the power 2 into n to the power q plus n s square 2 cube by b plus 2 n square.

So, you can see your by having the large number of blocks large your defining n cube by b and that will reduce the capacity misses significantly because you know that you are transferring all the a small block instead of transferring entire rows and columns. As a consequence, your performance your capacity misses will be much smaller and this exploit a combination of spatial and temporal locality because computation is not really in does not involve. Simultaneously, the entire arrays it is computation is done over a small portion of the array by exploiting that idea you are using exploiting the combination of spatial and temporal locality to achieve this blocking. This can also be used to help register allocation.

(Refer Slide Time: 54:55)



Blocking: Example

Note in contrast to previous Figure, the smaller number of elements accessed

This is after doing the blocking how it is done is shown here in this particular case as you can see note in contrast to the previous figure smaller number of elements are accessed. Earlier, your accessing more number of elements smaller number of elements access for array x for array y and also for array z. You are doing the computation over this area only so one block then will be x go to another block in this way block by block.

You will do the computation and performance will be will improve the total number of computation will not change in this case however the performance will definitely improve. So, with this we have come to the end of today’s lecture, we have discussed various techniques for reducing the miss rate and we have seen how we can use hardware’s by increasing the cache size by increasing the associativity.

Also by increasing the block size we can reduce the miss rate and also you can use various compiler optimization techniques that we have discuss briefly in this lecture, which can be used to reduce the miss rate. So, in my next lecture, I shall discuss about the reduction of miss penalty, because that is the third parameters, which dictates the performance of the cache accesses.

Thank you.