High Performance Computing

Prof. Matthew Jacob

Department of Computer Science and Automation

Indian Institute of Science, Bangalore

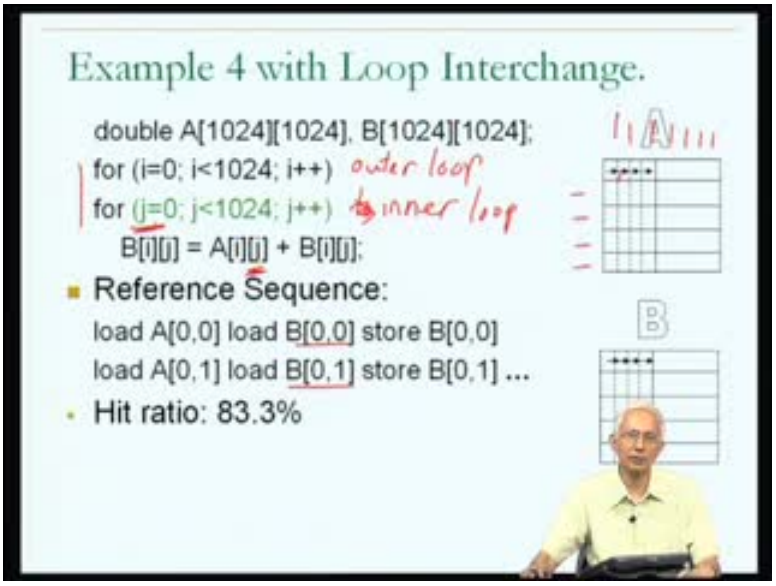
Module No. # 06

Lecture No. # 31

In the current lecture, we will continue looking at an analysis of simple loop like programs and the impact, as far as the cache is concerned. Our interest is to try to understand how that cache hits or misses will be determined by the nature of the program, and what we can do to improve the cache behavior as far as the program is concerned. We are looking at simple examples, to allow us to do a complete analysis.

In the previous lecture, we had been looking at an operation on 2 dimensional matrices. The operation was basically, adding the elements of two 2 dimensional matrix term for term i ,refer to this as 2 dimensional matrix sum and we had a loop which had a problem with it in the sense that, if you refer back to the slide one from the previous lecture, the loop was dealing with 2 matrices A and B, each of which was 2 dimensional matrix of 1000 and 24 rows, each of 1000 and 24 columns and I was adding the element of A to the corresponding element of B, and putting this as a new element new value of the element B, by going through the matrices column by column. That is what the 2 for loops that we have used among 2.

(Refer Slide Time: 01:04)



Because of the way that the for loops are structured, it amounts to going through the matrices A and B, column by column. But, as we learned in the previous lecture, for a C program, one could safely assume that the elements of the matrices are stored in memory row by row. And therefore, by accessing the elements of the matrix column by column in this program, I would not be benefiting well from spatial locality of reference.

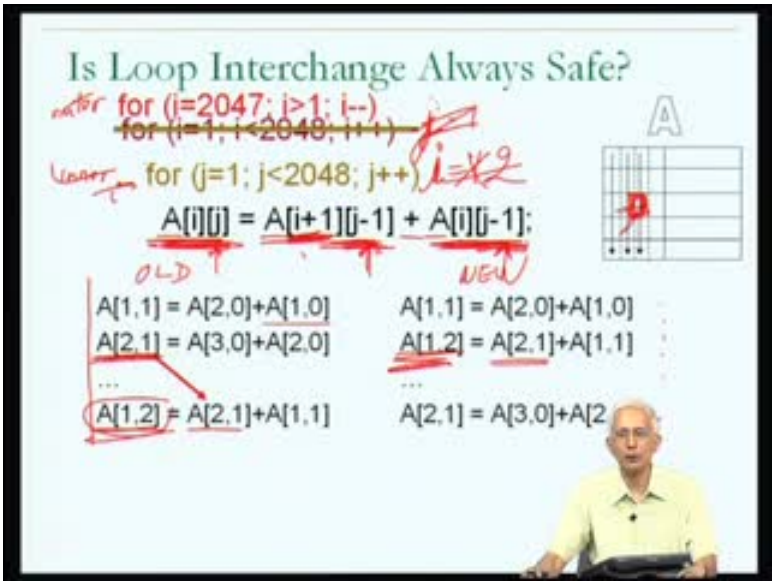
So, the idea which we looked at last time was how one could just interchange the j loop and the i loops, without changing the meaning of the program and get that benefit. For example, if I modify the program so that the i loop was the outer loop.

So, always the outer loop and the inner loop. So, previously, the j loop was on the outside but, I've now moved it on the inside. So, the J is the inner loop, and we notice that now the situation is that the reference sequence is that I load A of 0 the first time through the loop we are dealing with I equal to 0 and J equal to 0 the second time where the loops we are dealing with I equal to 0 and j equal to 1 and so on. In other words, we are actually going through the matrices row by row, and in terms of the diagram, which we had used the previous class I view the 2 dimensional matrix as being rows these are the rows and these are the columns of the 2 dimensional matrix.

And this particular program now, is accessing matrix A by going through AF00 followed by AF01, which is the second element on the first row.

So, this program is now going through the matrices row by row, which is the same as the storage order and which would therefore, benefit from spatial locality reference. The same is true with the matrix B. Notice that B of 00, and after that we access B of 0 1 and this basically has to do with the way that the for loops are structured. For this 2 dimensional example, we notice that, it is beneficial for us to have the index which is the inner loop as the second index of the matrices. In other words, the column index of the matrices. And then, we get this benefit of going through the matrices row by row. So, that is something to look for in the programs as we write them.

(Refer Slide Time: 03:48)



But, towards the end of the previous class we ask this question. So, you know that you want to get benefit of spatial locality of reference. But, can you just arbitrarily interchange loops is in that potentially going to change the meaning of the program. So, let us look at an example, where it is in fact, going to change the meaning of the program. Here is a slightly more convoluted example. In this example, I am not dealing with two 2 dimension matrices. I am only dealing with 1 2 dimension matrix which is called A. But, we are doing a more complicated operation on the matrix.

So, if you look at the way things are set up, I have the outer loop, which is using index j. So, once again, we consistently talk about this as the outer, and this as the inner the outer loop is the 1 with the index j and you'll notice that the index j is the second subscript is

used for the second subscript of the matrix a in all the 3 places that matrix a is refer to in inside the loop.

Remember from our previous slide that, what we want is that, the inner loop index should be the column index to the column subscript for the matrices. But, in this particular example, would we have is, each time through the loop we are adding some element of A to some other element of A, and making this a new element value of a third element of A. So, the calculation is little bit more complicated than the kind we seen before. Let me just, the diagram what is happening here the first time through this loop we have i equal to 1 and i equal to 1 and therefore, what this is being calculated is A of i plus 1, which is going to be remember that, the first time through this loop i is equal to 1. I am sorry, j is equal to one and i is equal to 1.

Therefore, A of i plus 1 is going to be A of i plus 1. j minus 1 is going to be A of 2 0 and A of i j minus 1 is going to be A of 1 0 .Therefore, the first time through the loop, we are adding A of 2 0 to A of 1 0, and making this a new value of A of 1. Similarly, the second time through the loop, we are going to have j still equal to 1 but, i equal to 2. Because, the second time through the loop, the inner loop index changes to 2 and we therefore, have A of 3 0 being added to A of 2 0, and this becomes the new value of A of 2 1 and so on.

So, this is what happens across iterations of this particular loop W nested loop. Now, if you think about the calculation that is actually happening, you may ask is just a meaningful kind of a calculation or it is just a trivial example to show something wrong, that will never occur in real life. Just remember how the modifications to the matrix element A of i j is happening.

So, basically the new value of A of i j is given by the old value of A of i plus 1 j minus 1. What is i plus i i plus 1 it should be interpreted as to find out, the new value of the element in the ith row and the Jth column. Then you have to take the old value from the element in the i plus first row and j minus first column and add to that the element from the ith row and the j minus 1st column. So, if I am concerned about how the element i j changes, I notice that I have to look at the element which is in the same row as it but, 1 column to the left and add to that the element which is in the next row but, the previous column and therefore, this diagrammatical illustration over here, it shows how A of i j is

modified using the element to its left as well as the element which is below it and to its left.

So, this is just another way of updating a matrix and we will some relevance for different kinds of competitions it is not a trivial example, it is a systematic way to update a matrix in along blanks, what we have seen over here.

Now, unfortunately, with this particular w nested loop, we notice that we are not going to be able to benefit much from spatial locality of reference in that. This seems to be stepping through the matrix in the wrong order, and how we getting some inside into what order will be beneficial. I am just using the observation from the previous example. You remember from the previous example, we came to the conclusion that it is beneficial in C programs to have things set up. So, that the index of the inner loop, in other words i is used as the second subscript of the matrix. In other words, I do not want to see i as the first subscript, which is what is happening over here. I want to see i used in the second subscript.

But unfortunately in this particular piece of code, j is the second subscript for all the references of the 2 dimensional matrix A.Therefore, this is the situation where we would want to try to interchange the loops. Make the j loop the inner loop, and make the i loop the outer loop, which is what I will try to do.

So, here I have arbitrarily just interchanged the loops. So, I've just done the interchange. I've taken the line which has 4 j and put it after the line, which has 4 i .Now, if actually step through the meaning of this modified program, which is what I've done over here. So, on the left, we had what the program used to do and over here we have what the program does now that I have done this loop interchange. You notice that it calculates in the first iteration A of 1 1, which is A of 20 plus A of 10 in the second iteration. Remember that the inner loop index, which is now the j index, is going to change and .Therefore I, instead of modifying A of 2 1, we modify A of 1 2 and you'll able do the calculation of what the entices you notice that it is a different calculation from what was happening in the case of the old loop but, that is as long as the net result on the matrix is the same.

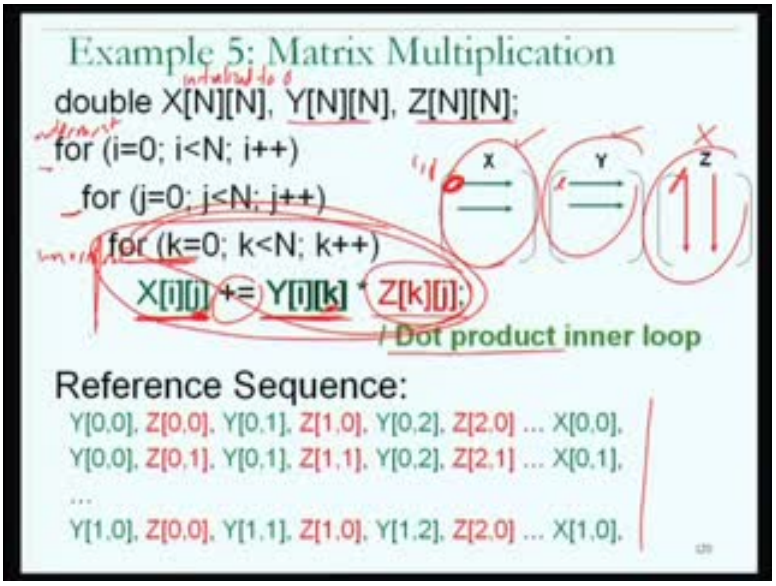
So, you could step through the entire set of the different operations, and then look and see whether there is any reason to suspect that net effect is going to be different from the

net effect under the old version of the program. And the giveaway, that something could be wrong would come if you look at the old version of the program, where in the code segment that I have over here, you'll notice that in calculating the value of A of 1 2, the old value of A of 2 1 is used. But, the old value of A of 2 1 is 1, which has already been computed in this loop, in the old version of the program.

Whereas in the new version of the program, when I calculate the value of A of 1 2 ,I am not using a newly computed value of A of 2 1 but, the original value of A of 2 1 which means that the meaning of the program is different. In other words, it is not safe to just interchange these loops by just moving the j loop header after the i loop header. The net effect would be to change meaning of the program, the program does something different.

Therefore, I would have to do this interchange of loops a little bit more carefully. One cannot just move the loop headers and expect that the program will behave the same way. One has to properly understand, what this program was doing. One wants to end up with the situation where the i loop is the outer loop and the j loop is the inner loop but, One may have to do something more fancy. For example, in this particular example, if you think about it for some time, you realize that one could correct for the problem with the meaning of the program by having the i loop modified. So that, rather than going from i equal to 0 1 up to 2000 and 47, it goes from 2000 and 47 down to the lower value. Now, this is the somewhat more complicated example but, I merely mention it to illustrate that there will be situations, where loop interchange is not safe unless it is done carefully. But, loop interchange will be beneficial from the perspective of improving locality of reference of a program. With this, we can look at a slightly more complicated example. We come to an example which many of you are familiar with. Example: which is matrix multiplication two. Here, we’re talking about, I am going to be referring to 2 dimensional matrix multiplication.

(Refer Slide Time: 11:31)



So, we have two of the examples which I am going to use. We have two square matrices and we multiply two square matrices to produce a product matrix, which itself is a square matrix and many of you will be familiar with the way that matrix multiplication could be done in this particular piece of code. Rather than telling you what the size of each matrix is, I am just using N as the size of the dimension. So, N could be 2000 and 48 N could be 1000 and 24 whatever it is.

So, we have 2 N by N matrices, Y and K .And what this A loop is supposed to be doing is, multiplying the matrix Y by the matrix, I am sorry the matrix Y and the matrix Z. Multiplying the matrix Y by the matrix Z and making this product the product matrix is the new. It is the value of the matrix X. So, we are computing X equals Y, multiplied by N where X Y and Z are all 2 dimensional matrices of size n by n double.

So, over here, we have a possible way to do this. In this particular piece of code, I am assuming that the double matrix X has been initialized to be all 0. So, X is initialized to be full of 0, and the matrices Y and Z contain the values such that they are to be multiplied. So, I have a triple nested loop, the i loop j loop and the k loop each time through the loop we've multiply a row of Y by column of Z and make that the new value of ...For example, in this particular piece of code, we multiply the ith row of Y by the jth column of Z dot element by element computing. In fact, the dot product and make this a new value of the i jth element of X.

This is one way to do the matrix multiplication for these 2 dimension matrices. So, notice that I am doing plus equals and that I have initialized the matrix X to be 0. This is the substantially more complicated example.

So, we need to try to understand this. It will be quite difficult for us to try to go through this by listing the complete reference sequence, and trying to think of the base address of the different elements, and computing what might be conflicting, which is something else. So, it is fortunate that we have moved away from that mode of analysis to something just slightly higher level. But, just for proper understanding of what is happening at the, at a slightly higher level, let me rush through some parts of the reference sequence. So, at the way I had described it, do you notice that the way that this implementation of matrix multiplication is working is that, it multiplies the ith row of Y by the jth column of Z, in order to compute the element for X of i j, and it does this by multiplying Y of i 0 by K of 0 j and then it multiplies Y of i 1 by Z of 1 j and so on.

So, we could come up with the reference sequence but, may first observe that we are in a situation where, first of all Y of 0 is multiplied by Z of 00. Then Y of 0 1 is multiplied by Z of 0 1, Y of 0 2 by Z of 2 0 and ultimately, that sum is what is put into X of 00. So, in this particular, in implementation, I am assuming that in implementing this addition, the compiler is not adding to the variable X each time through but, it is actually accumulating this product inside a register and storing the value of the accumulated value into the element X, only at the end of the iteration.

So, there is only one reference to X each time in its each of this lines but, there are N references to Y and N references to Z, and it is basically multiplication of a row of Y by a column of Z, to generate a single element of X. Now, if you look at how things are proceeding in this loop, for example, let me just look at X of i j .When I look at X of i j I, notice that there is the outer most loop. So, the i loop is the outer most loop and the K loop is the inner most loop in, now have 3 loops. So, I cannot talk about outer and inner. I have to talk about outer most and inner most but, if I look at the reference through which X is refered to in this sequence, I notice that the column subscript is j and that the row subscript is i .The first subscript is i the second subscript is j but, this is in in keeping with our insight that we had derive from the previous examples which is that we want the inner loop. So, between the i loop and the j loop, we notice that the j loop is inner to the i loop .The j loop is inner to the i loop and, it is the j loop in other words, inner of the two

loops which is being used as the column subscript for X of i j. Therefore, in some sense, I would be able to make the observation that the as far as this program is concerned, it is referring to the X matrix row by row, which is what I am using in this notation.

Similarly, if I look at the reference to the matrix Y, I notice that the second subscript column subscript is the K is coming from the K index which is the inner most loop which means that for the which is the desirable property that we had which is leading me to believe that once again for the Y loop as far for this for this program as far as the references to y are concerned once again y as being refer to row by row.

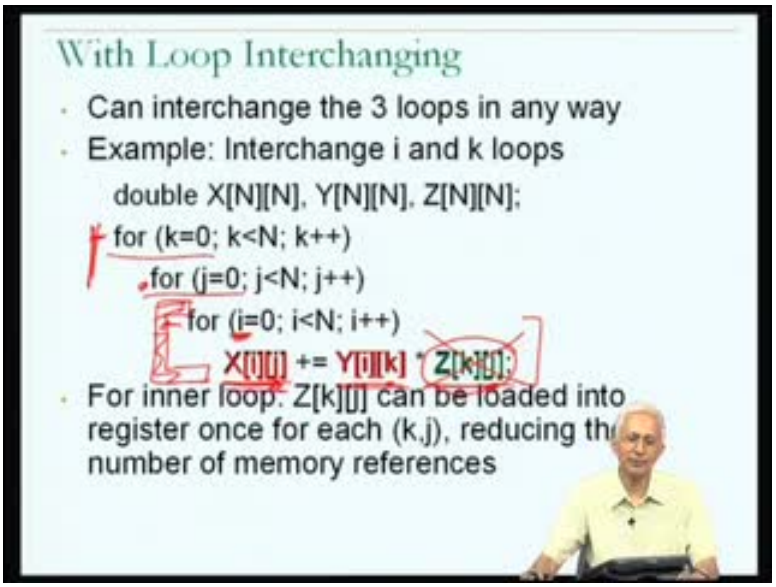
However, when I look at the Z reference to Z, I notice that it has K as its first subscript and J as it second subscript. But, K is the inner most of those two. Between J and K is the inner most loop, which means that, this is a problematic reference in that .Looks like, the 2 dimensional matrix Z is being accessed column by column, and this being C code from the looks of it, we understand that references to X and Y will benefit from spatial locality of reference. But, the references to Z will not benefit from spatial locality of reference.

Which is why, in this reference sequence down below, I have shown the references to X and Y in green. So, out of using green as a positive color and I am showing the references to Z in red as a negative color. In that, the references to Z are not going to benefit from the cache spatial locality of reference. Therefore, this is one way to set up matrix multiplication but, may not be the best way in terms of fully exploiting spatial locality of reference and temporal locality of reference, for all the elements that are being accessed.

Now, one comment which I would want to make at this point: if you look at the way that I was describing the way that this matrix multiplication functions, or this matrix multiplication nested loop is working, I described it as a situation, where each time the objective was to calculate particular element of X by multiplying. To calculate the i comma jth element of X ,we multiply the ith row of Y by the jth column of J. And in effect, what we were doing was to compute the dot product of the ith row of y by the jth column of Z .And in effect, what the inner most loop was doing was computing that dot product in the inner most loop. we had K varying from 0 up to N and Y of I K was multiplied by Z of K J, and this was accumulated .And if you look back it our discussion

of the dot product you'll notice that this was. In fact, this is the same as what we talked about then.

(Refer Slide Time: 19:46)



So, this implementation, matrix multiplication is actually using the inner most loop where the inner most loop is computing a dot product. There are; obviously, other ways that matrix multiplication could be implemented, and let me just suggest that other another possibility or another way to think about different ways and matrix multiplication could be implemented. One could actually think about interchanging these loops and thereby, getting different versions of matrix multiplication rather than, this dot product inner loop version of matrix multiplication. And you may want to think about whether it is fair to interchange the 3 loops arbitrarily. In other words, instead of having i followed by j followed by k loops, I could have K followed by J followed by the I loop. In other words, some arbitrarily changing of the order in which the three: the I the J in the K loop are included.

And as a simple example, just looking back- remember that I had the i loop outer most, the j loop intermediate, and the k loop as the inner most. What if I interchange the i loop and the k loop and that is exactly what I've done here. I've left the the content of a loop is exactly the same x of i j plus equals y of i k multiplied by Z of k j. But, all that I have done, I've interchanged the i loop, used to be the outer most loop ,the i loop is now the inner most loop. The k loop used to be the inner most loop, it is now the outer most loop.

And it turns out that, this too would be a correct implementation of matrix multiplication. This could be done appropriately and the question is whether this would behave any better than the previous implementation of matrix multiplication.

So, we could run through the similar kind of analysis, what we did before. Remember that at this point, the way that we are analyzing these versions of different implementations of matrix multiplication, is by looking at the statement inside the loop. And for each term inside the statement, we are trying to assess whether the matrix, the 2 dimensional matrix which is being referred to there is being referred to row by row or column by column. And assuming that this is c code, we are trying to, we would want the references to happen row by row. So, we look at i j and we notice that the j subscript is outer to the i subscript.

Therefore, this is not good. We look at Y of i k, and we notice that the k subscript is outer to the i subscript and once again this is not good. We look at the Z of k j and we notice that this is good, because the second subscript, the j subscript is inner more to the k subscript.

So, in some sense, what we've done here is to create a situation where the reference to z is happening row by row. So, all the references to the z matrix are happening in row by row fashion, exploiting spatial locality of reference. But, the price that we paid is that the references to y and references to the x matrix are happening column by column. The previous version where we had the dot matrix inner product, I am sorry the dot product inner loop implementation of matrix multiplication, we had the references to x and y happening row by row and the reference to z was column by column.

In this implementation things have switched around completely. But, note that there are other interchange reorderings of the three loops, which might have different benefits. So, we want to think about that to the extent that you have time.

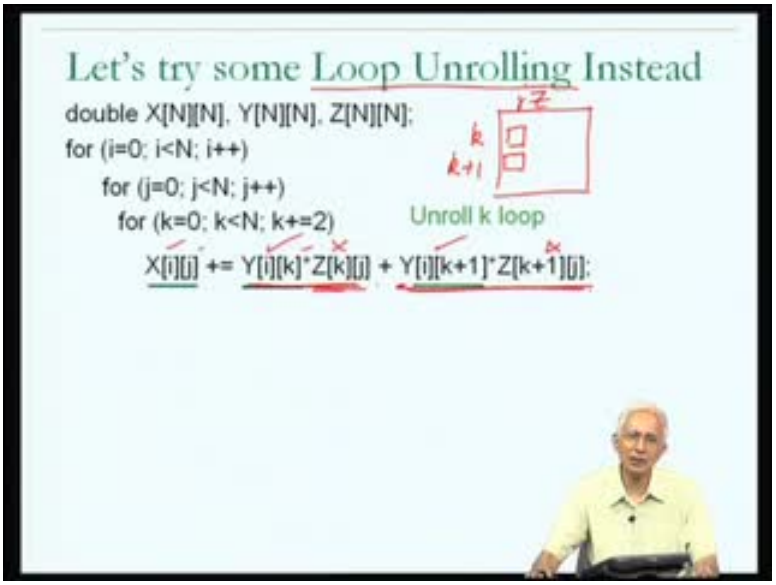
Now, one observation which I did want to make is, for this particular implementation that we have on the screen right now, you will notice that as far as the inner loop is concerned. The inner loop is, one way i varies from 0 through n and therefore, as far as the inner loop is concerned Z of k j is a constant. Z of k j is an element of the matrix z and it does not depend on the inner most loop at all. Therefore, for all of the n iterations

of the inner most loop the value of Z of k j is the same. It just because the value of Z of k j is determined by the value of k and the value of j.

Therefore, Z of k j does not change within the inner loop and hence can actually be loaded into a register once for each value of k and each value of j, reducing the number of memory references substantially. And therefore, one should not really view Z of k j as being a memory references at all, since it is going to be loaded into a register. And within the inner most loop it'll be access out of that register and therefore, thinking of the z references as being good, because they are happening row by row, it turns out was of artificial benefit, because in reality one would not think of the references to z as being referenced out of memory but, rather has being maintained by the compiler hopefully in a register.

Therefore this in effect has to be viewed as being a very poor way to multiply matrices.

(Refer Slide Time: 24:11)



Now, rather than looking at the different inter-changings of the ordering of i j and the k loops of matrix multiplication, I we will try to understand how we can improve the quality of the matrix multiplication loop. A triple nested loop by using some other techniques that we have seen, and one technique that we have seen some time back when we were talking about improving the quality of instruction sequences: its static scheduling and so on was the idea of loop unrolling. You will remember in loop unrolling, we try to do more than one iteration of the loop each time through the actual

loop itself by duplicating the code of the loop, in order to do two iterations of the loop in each of the instances of loop.

So, if I look back at it, my i j k, the original dot product inner loop version of matrix multiplication, this is the original version that we had. Let us try to think of this loop, now from the perspective of loop unrolling. Let us see, if we actually try to unroll one of these loops, what would happen. Loop unrolling produce benefits. When we are looking at instruction scheduling, it is conceivable that loop unrolling will either produce benefits or give us some insight into how to improve the quality of matrix multiplication, this matrix multiplication piece of code.

So, how shall I, what shall I do in terms of unrolling? Let me start by trying to unroll the k loop. So, I will, what I will do is, each time through the k loop rather than computing just x of i j, I will try to compute, I am sorry .Rather than just computing one value of k, I will try to do the computation for two values of k. So, I will do two iterations of the k loop in each pass through the k loop .And we know how to do that. Let merely involves each times to the k loop I increment k by two and then within the loop I compute not only y of i k multiplied by Z of k j but, also compute y of i k plus 1 multiplied by Z of i Z of k plus 1 j and I add both of these terms to y x of i j.

So, let me just go back in the unrolled. Before doing the loop unrolling, If I did one multiplication and addition each time through the loop, when I unroll the loop I do two multiplications each time through the loop and therefore, I am reducing the number of times that the k loop has to be executed.

Now the question which we need to ask at this point is : how does this impact the locality properties of the program ? And we, you'll recall from the previous version of program that, we viewed the x of i j as being a good reference, the y of i k as being a good reference and the z of k j was a bad reference. It was one which was the matrix k z was being referred was being accessed column by column, because the second subscript was inner- outer mode to the first subscript.

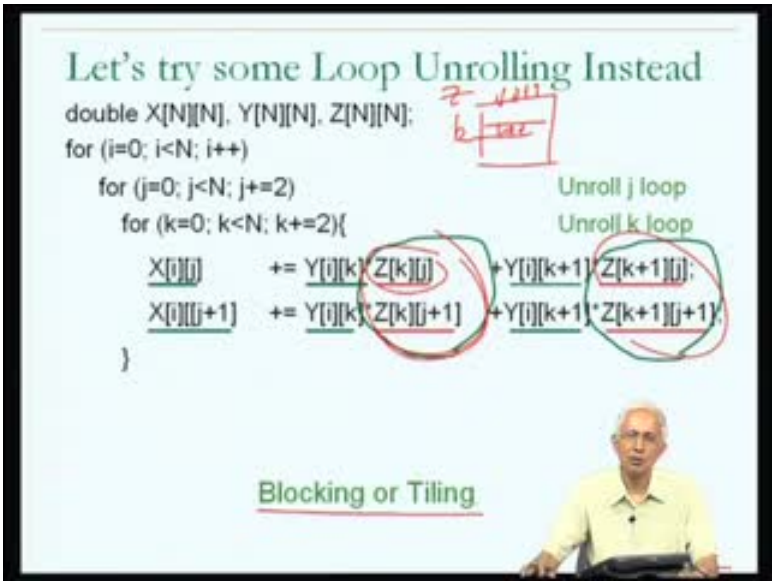
Now, has this changed at all by unrolling the loop? If you look at the unroll version of the loop once again, the x of i j is a good reference, y of i k is good reference ,y of i k plus one is still a good reference, because the k, the column subscript is based on the k

loop which is an inner most loop. And once again the z references of Z of k j and z k plus 1 j are both bad references.

So, that has not changed in terms of coloring. I will say that, this is good, this is good, this is good, this is bad and this is bad that has not changed but, what has changed is that each time through this loop I am referring to z of k j as well as z k plus 1 j .And what is the relationship between z of k j and z of k plus 1 j? Remember your picture of the matrix 2 dimensional matrix z.

So, here we are referring to k and j and k plus 1 and j. So, Z of k j is one particular element of z. and Z of k plus one j is the element in the next column. I am sorry, in the same column but, the next row. Therefore, we're not actually benefiting from the references to Z of k j Z of k plus 1 j at the level of the second subscript is not good but, this not even situation where Z of k j and Z of k plus 1 j are in the same row .Therefore, we do not see any benefit from doing this but, that might be just we because we have not done the unrolling they we should have done.

(Refer Slide Time: 28:33)



So, let us look at another possible mechanism for unrolling. For example, once again I go back to my original loop but, what I will try to do now is, I will try to unroll 2 loops. I will try to unroll both the j loop and the k loop, in other words I will try to do, in each iteration of the k loop, two multiplications and in each iteration of j loop, I will try to put the code for two values of j. Now what will this amount to, in terms of the code itself?

You'll recall that, the way that we handle the enrolling as far as the k loop was concerned, was by incrementing k by 2 and then within the program body, we did two iterations of whatever the k loop was supposed to do.

We'll have to do something very similar over here. If I want to unroll both the j loop and the k loop, then I could start by unrolling. For example, we know that, in order to unroll the k loop, what I had to do was to include two terms corresponding to two iterations of the k loop. What does it mean to unroll the j loop? It basically means that, I need to do two iterations of the j loop, which will mean, having the body of the loop occurring twice, one with the value j and one with the value j plus 1. Therefore, unrolling both the j loop and the k loop, it'll make the body of the loop a little bigger. We now have 2 c statements, each iteration of the k loop. Two statements are coming, because I am unrolling the j loop. And each of the statements has two terms in the sum, because I am unrolling the k loop as in the previous example. At this point, let me just clear the screen.

So, with the successful unrolling in both of these dimensions, this is what we have. And now, let us look at what we have on the screen in terms of the locality of reference, spatial locality. Once again, we will observe that things have not changed. If I look at each of these terms in isolation, because we have not changed the ordering of the loops, and we're basically not changing the order in which subscripts are occurring in anyone of these references. Therefore, there is no need to look at whether x of i j is good, whether Z of k j is good or not. We know that, all the references to x and all the references to y are happening row by row, and all references to z are happening column by column. That has, that could not have changed, because we have not changed the order of the loops, and we have not changed the order of the subscripts inside any of these the array references. But, what has changed, what has changed is that in the body of this loop, we now have instance of referring to Z of k j as well as Z of k plus one j. But, we had that in the previous unrolling, that we tried. What is new, is that we now have a references Z of k j plus 1 as well as the reference Z of k plus 1 j plus one .And what is the relationship between Z of k j and Z of k j plus 1? The answer: if you go back to your picture of the multi-2 dimensional matrix Z, is that Z of k j and Z of k j plus 1 are consecutive elements in the same row of a matrix Z.

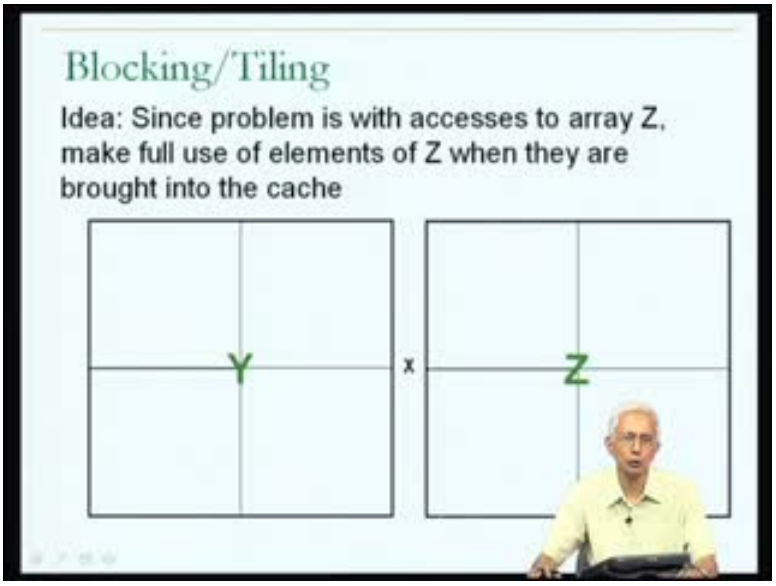
So, what we have in fact, done by this more ambitious unrolling of both the j loop and the k loop is, we have artificially created some spatial locality of reference. Why do I say

that? I say that, because even though the reference to Z of k j might be a miss, when this program executes a little later, in the same program there is going to be a reference to Z of k j plus 1, and Z of k j plus 1 is a neighbor of Z of k j. In effect, what we have been able to do by doing this unrolling is, to generate some spatial locality of reference, the spatial locality of reference, as far as Z of k j and Z of k j plus 1 is concerned. Just little bit more spatial locality of reference, as far as Z of k plus 1 j Z of k plus 1 j plus 1 is concerned.

And therefore, the unrolling has helped us by generating some spatial locality of reference, and improving the hit ratio, because we are actually getting spatial locality of reference from a program, which until now was not showing spatial locality of reference to the references to the matrix z. Recall, we were not concerned about the references to matrix X or Y. They were happening row by row, which is what we want them to be but, we have, we're able to do this purely through loop unrolling.

So, there is something in doing loop unrolling, that we are getting or going to get a benefit, from the perspective of spatial locality of reference. Now, what this could lead us to is, that whole new perspective on how to set up matrix multiplication, rather than trying to carry forward this idea of loop unrolling and increasing the amount of spatial locality of reference, that might be available. We could try to distill out the essence of what is being achieved by this loop unrolling. And that is what be, I will hence what call blocking or tilling but, let me just explain what I mean.

(Refer Slide Time: 33:28)



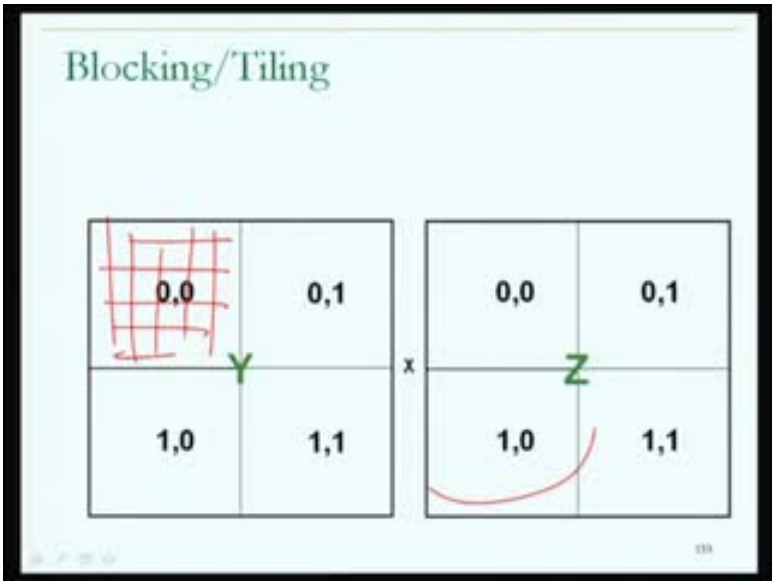
Now, the idea of blocking or tilling is, that we identified from our quick analysis of the matrix multiplication program which had i j and k, where i was outer most loop and k was the inner most loop, that for the dot product in inner loop version that we were dealing with the references to the matrix z were doomed in that they were happening column by column and they had a problem with them.

But, what we had done, what we saw was that, by doing this unrolling of the loop j and the loop k a little bit, we were actually able to make use of elements of Z a little bit more. Like when the element of Z j k was brought into the cache, we were able to use its neighbor, and that was happening entirely because of the unrolling. Therefore, if we carried this idea through and we say, every time an element of z is brought into the cache, we try to use all the neighbors of z before they are eliminated from the cache by replacement. Then, this will provide a different perspective of how the matrix multiplication could be set up right.

So, the idea in general would be something like this. Again I am just going to try illustrating it using a diagram. May sound a little bit confusing, but will help us when we look at the code which is going to follow. So, ultimately I want to multiply a large matrix Y by a large matrix Z. We are assuming that they are square matrices, for the ideas that we were going to come up with, will generalize to other arbitrarily matrices as well.

Now, let's suppose that, I try to view what would happen if I broke each of the matrices up into four parts. So, I just look at the, I divide each of the matrices into 4 equal parts. And I know that, in order to multiply these 2 matrices and get the product matrix x, I could, instead of multiplying a full row of y by a full column of z to generate one element of x, I could do the multiplication in parts, as was happening in the case of the versions of matrix multiplication, where I interchange the loops.

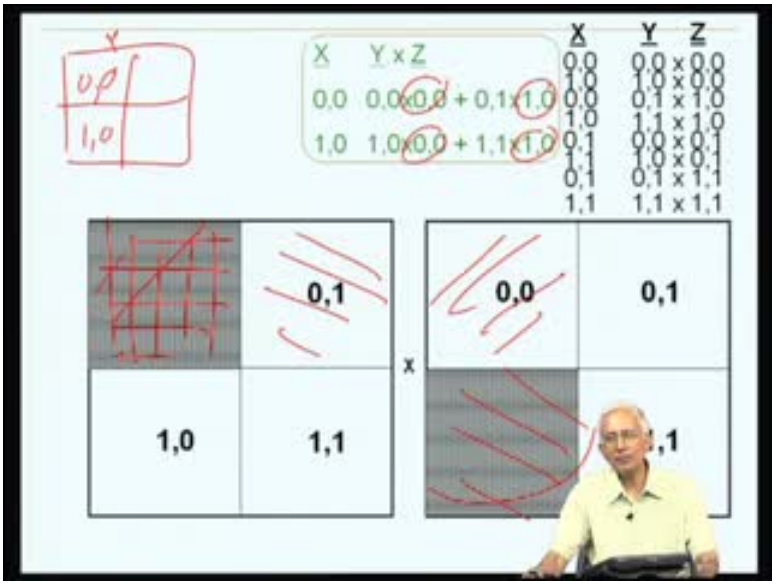
(Refer Slide Time: 35:25)



So, if I number each of these parts as 000110 and 11, and once again, I am not talking about 00 as one element of the matrix Y, rather it is one fourth of the matrix Y. It is some collection of rows and columns. So, one fourth of the matrix is in, of the matrix Y is what I am referring to as Y of 00.

Now, in order to compute the product matrix x, I will have to multiply Y of 00 by Y of 0, Z of 00, and I will have to multiply Y of 00 by Z of 01, I will have to multiply y of 00 by Z of 10, as well of y of 00 by Z of 11. But, instead of doing them one by one without regard to reusing the values, I will do it in the more calculated way.

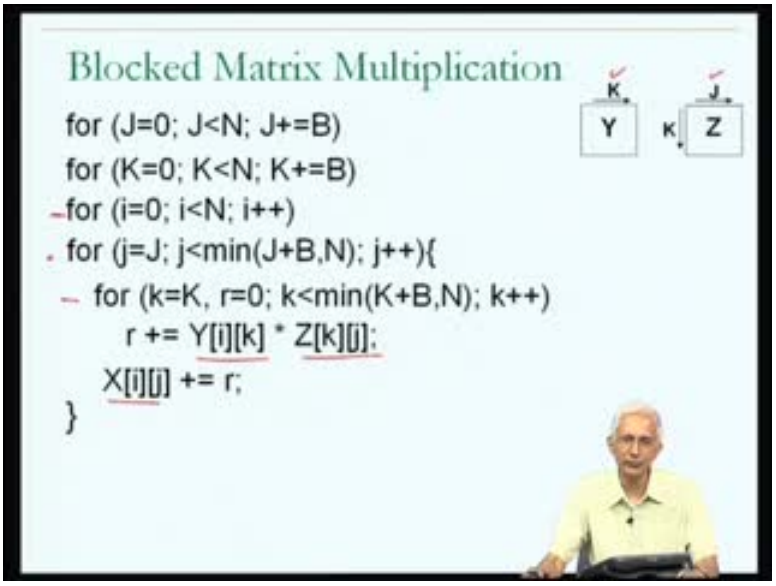
(Refer Slide Time: 36:17)



For example, I notice that, in order to generate the quarter of the matrix x, which I will call x of 00, I really need to multiply Y of 00 by X of 0, Z of 00 and y of 01 by Z of 10. That is how I would calculate the matrix x upper quadrant. So basically, to generate the elements in the matrix x, which are in that particular region of the matrix x, I need to do this multiplication, as well as the other multiplication which I am showing over here, this multiplication. And that will amount to doing the multiplication of the rows of x is y by the columns of z, in order to generate the elements in this region of the matrix x.

Now, when I look at this sub division of the work, that has to be done to generate x of 00 and expand it to the work that has to be done to generate x of 10. In other words, this quadrant of the matrix x. I notice that Z of 00 is reused and Z of 10 is reused and therefore, this gives me the clue as to how I could do the reuse of z once again. I am not concerned about the accesses to the matrix X, of the matrix Y since they are happening row by row any way but, I am giving the competition to trying to reuse the elements of Z whenever elements of Z or a block of elements of z is not into the cache. We want to do the other calculations which involves those elements of the matrix Z

(Refer Slide Time: 37:59)



And that leads us to some calculation, at a lower level to figure out exactly which elements have to be computed, in order to get the benefit of the reuse of the elements of the matrix Z. This could actually be encoded into, what is called the blocked or tilled version of matrix multiplication. And this is the kind of implementation, that would find in linear algebra package or elsewhere or read about in a course of numerical analysis possibly.

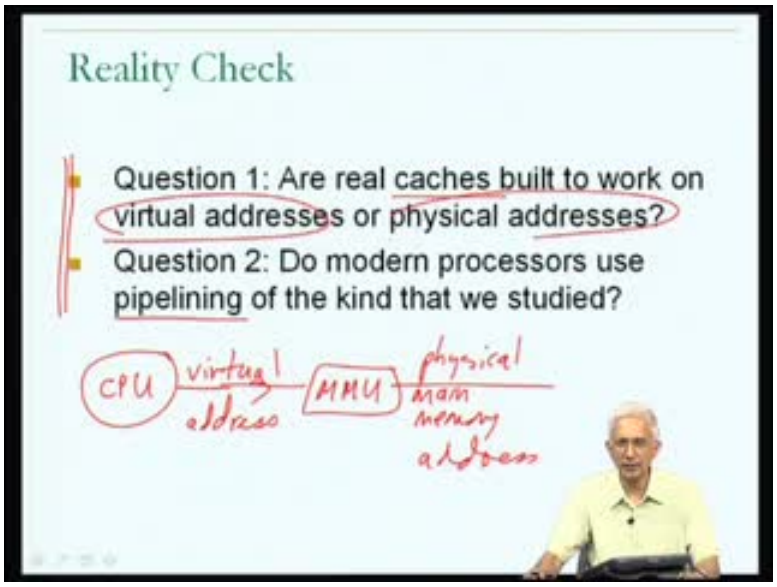
This is a general idea. Now we will be, rather than thinking about the matrix multiplication whereas, i j and k loop may have to think of the matrix multiplication is happening at two levels : at the first level, we are thinking about multiplying blocks of Y by blocks of Z and therefore, I have a block index for Y, which is capital K and the block index for Z which is capital J and therefore, at the highest level, we think of the matrix multiplication in terms of multiplying a block of Y by a block of Z along the lines of what we had seen in the previous slide, multiplying a block of Y by a block of Z.

Now, in order to do that multiplication of a block of Y, by a block of Z. Given that each of those blocks is made up of many elements, they will have to be three nested loops, just that we had before an i loop j loop and the k loop to do the individual element multiplications. But, by structuring the matrix multiplication in this way, we are going to be able to get the benefit of reusing the elements of Z and therefore, getting a much

better overall spatial locality of reference and therefore, by improved cache performance for the resulting program.

This kind of an implementation, though looks somewhat more complicated, may in fact, end up being faster because of the benefits that it gets from improve spatial locality of reference, and would be well worth experimenting with. It turns out that, the matrix multiplication example was substantially more complicated than earlier ones that we had seen, and hence our discussion of the matrix multiplication example did not go down to the level of calculating hits or misses but, was conducted at level where we are analyzing the properties of the different elements, that we see in the terms relating to the calculation that was happening within the loops of the matrix multiplication.

(Refer Slide Time: 40:02)



Now, with this I like to wrap up our discussion of the different examples relating to doing cache calculations I will come back a little bit later to some closing comments on that but, before closing today's lecture I wanted to refer to two questions which should come up at this point in time.

Let me just remind you, after this point in the course we've understood a lot about the underlying hardware in a computer system. You've understood a lot about the underlying software in a computer system and we have also understood something about how they relate to each other and how they could per have an impact of the programs that we are trying to run on the computer system. And two of the important things that we saw along

the way were, that computer systems now must have cache memories. I motivated this from the perspective of the speed disparity between processors and memories, which is getting worse and worse, thanks to various technology trends. And the other important idea that we saw was the idea of pipe lining. The idea that the throughput with which instructions could be executed could be substantially improved, rather than trying to make the amount of time that it takes to execute a single instruction as fast as possible.

Right now, just to bring us up to some more recent developments in computer architecture, I wanted to talk about two questions to give some minor corrections to some of the our perspective of some of these things. Now, the first relates to caches and the question is: are caches really built to work on virtual addresses or they built to work on physical addresses? And let me just remind you what we mean by physical addresses and virtual addresses. You will recall that the processor generates an address, and typically we've understood that the addresses were virtual addresses, and that the address had to undergo translation to generate an actual physical or main memory address, and this translation was done by a piece of hardware which we call the memory management unit .

And it just a physical or main memory address, which could actually be make meaning as far as the hardware main memory is concerned. But, we understood that this translation step, did have to take place to translate the virtual address to a physical address. Now, when we talked about caches, I did not actually indicate whether the cache was organized to take in a virtual address or to take in a physical address, which is why we will actually refer to this in little bit more detail before proceeding.

So, the question is: what is the relationship between the cache and the address translation? Does the address translation happen before the address goes to the cache, or does the address translation happen after the address has gone to the cache? Now, that the picture which I just drew was telling you that the c p u generates a virtual address, whether it be the address of an instruction or the address of a piece of data, is just an address from the address space of a process and is therefore, a virtual address and the m m u, the piece of hardware that does the address translation- translates it into a physical, a real main memory address.

So, it is possible that people could build caches. So that, the cache takes in a physical address, in other words the cache could be built so that, it views the physical address in terms of its least significant bits, which are the block offset. Its intermediate bits, which are the cache index, and the most significant bits which are the cache tag. Alternatively, things could be set up so that, the virtual address itself is what goes to the cache and the cache hardware is built. So that, it views the virtual address in terms of least significant bits being the byte offset, intermediate bits being the index into the cache and so on.

So, these are really two different alternatives as far as the design of a cache is concerned. Caches could be designed to work on physical addresses or they could be designed to work on virtual addresses. These are two design options. Now if the cache was design to work on virtual addresses then we should know that there is the danger that the virtual address that is currently be required by the processor might not be present in the cache and if it is not present in the cache then the cache will have to be loaded with that particular piece of data from main memory but, main memory is organize in terms of physical addresses and therefore, before the address can be sent to the main memory in the event of a cache miss the address will have to be translated therefore, the virtual address will have to go through the m u to generate a physical address which can be sent to main memory.

However, you should know that, if the virtual address, when it goes to the cache is a hit, then the cache can immediately provide the data and there is no need for the address translation to happen therefore, the second option that we look at, over here is actually to be viewed with some favor because, it is an option where the address translation can be avoided when it is not necessary. If you think about the first option, the one on the top, the address translation happens regardless of what happens inside the cache, because the cache has been organized to work only on physical addresses.

Now, if you think about the our discussion of the cache hardware, the cache hardware just takes in an address. It does not really use any knowledge about whether the address is physical or virtual. It just views the different bits of the address, based on the properties of the cache. The number of bits used for the offset is decided based on the size of a cache block. The number of bits used for the index is decided based on the associatively, whether it is direct mapped or associative as well as the number of the size of the cache and the size of the cache block .Therefore, it does not really make a whole

lot difference to the design of the cache, as far as we can see between whether it takes in a physical address or a virtual address. Addresses just have to be viewed in, by looking at the appropriate bit field from the perspective of the cache hardware.

But we do need to have some terminology now. Otherwise, I will have to keep on referring to the first option or the option on the top and the option on the bottom and therefore, I will actually refer to a cache which has been designed to work on physical addresses as a physical addressed cache and a cache which has been designed to work on virtual addresses as a virtual address cache. And once again, let me repeat, both of these are design options. They could be processors that have physical address caches and they could be processors that have virtual address caches.

Now, the question is: which of these sounds like a more beneficial or which of these sounds like a better idea? And I am sorry for facing the question in this roundabout way. But that is essentially what we are trying to discuss next. Which would be a better idea from the performance side? A virtual address cache, which in which the cache uses virtual addresses or a physical address cache in which the cache uses physical addresses? And let me put some of the pros, of the positive and negative points of each of these on the screen.

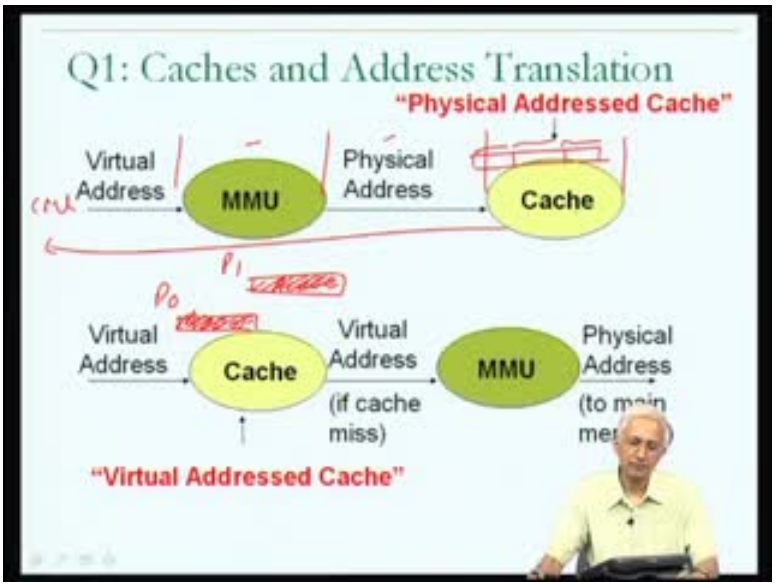
Now, one benefit you can see of the physical addressed cache is that the amount of time for a hit is going to be higher. So, this is a negative. Why do I say that the amount of time that it takes for a cache hit is going to be higher? Just go back to the diagram of the physical address cache. How much time does it take for the data to come back from a physical address cache to the processor? The answer is: its going to take as much time as it takes to access the cache but, in addition to that, it is going to take the amount of time that it takes do the address translation.

Therefore, the amount of time to get a piece of information out of the cache is going to be more than, what is the case for the virtual address cache in the event of a cache hit. Therefore, the hit time is going to be higher for the physical address cache than it is for the virtual address cache, and this is a negative. It is not a good thing. It is a bad thing. That is why I talked about, which is less preferable, what are the different negatives associated with each of these. So, there is one strong negative, as far as physical address cache is concerned. What about the virtual address cache?

Now, the thing to bear in mind about the virtual address cache is, we're talking about a situation where the cache is organized in terms of virtual addresses. So, the address which comes to the cache is a virtual address. Now, one must bear in mind that the virtual address has different meanings depending on what the process is.

I could have a virtual address, which relates to process zero, and in another virtual address which relates the process one and the bit patterns associated with a two virtual addresses might be exactly the same. Both of them could be hex a 000 but, we know quite well that hex a 000, from the perspective of process, one means something quite different from hex a 00 from the perspective of process 0 .Each process has its own virtual address space.

(Refer Slide Time: 48:30)



So, from the perspective of virtual address caches, we have to bear in mind that within that virtual address cache, you might have data and instructions of different processes in the cache at the same time, and it is quite possible that 2 of the things in the cache at the same time for process p 0 and process p .One might have the same virtual address but, might prefer to different entities because of address translation that one processes is protected from another process. What it is this mean? This means that we cannot really afford to have data or instructions from different processes, in a virtual address cache at the same time

So, one possibility is that when we switch from one process to another which is, would I call the context switch, we may have to empty the cache. So, when process zero was running, it is for the different virtual objects or process zero to be present in the cache. But, as soon as the operating systems switches to process one, suddenly all the data and instructions of process p zero, which were inside the cache under the virtual addresses that is a 000 and so on, do not mean what they should from the perspective of process p one and therefore, one possibility is that, the operating system might use instructions provided by the hardware to empty the cache in validate all of the entries inside the cache.

If this is not done conceivably, process p one, when it starts executing would access the instructions and data left there by process p zero and those are not the instructions and data that process p one should be using. Now, alternatively and we need alternative because this is an extremely expensive option remember the cache contains 32 kilobytes of information that has been accumulated as having been found to be necessary for the process in the near future if all of it is certainly thrown out of the cache then the programs will suffer in performance.

So, the alternative might be to complicate the cache directories so that, in addition to containing information about the tag associated with each cache block, we also contain information about which particular process, that particular cache block relates to. So, in addition to having information like tag information within each cache, directory entry also includes process id or process information, and in doing a look up into the cache check, not only whether the tags match but, also check whether the process id is match. Is the process id of that particular cache block the same as that of the process which is currently running? And if they are not the same then, that should not be viewed as a hit but, should be viewed as a miss.That is the data belonging to some other process.

Now, that is one. This is one negative as far as virtual address caches are concerned. One has to be very careful about handling data and instructions of different processes being in cache at the same time. Now another problem which could occur, as far as virtual address caches are concerned is something called a synonym. In a synonym, as you know from other context refers to two names for the same entity. So, we talk about synonyms as being two words which means the same thing or two words or two names for the same entity.

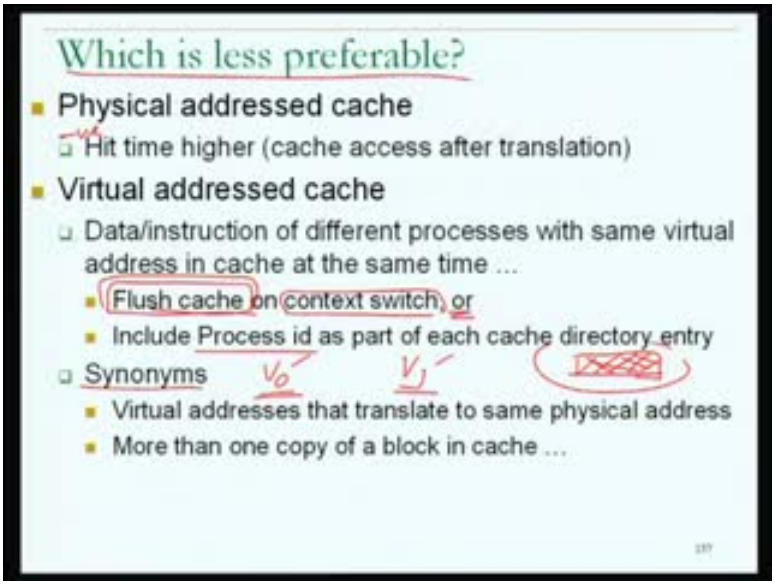
Now, in the context of virtual memory, one could talk about synonyms as being two virtual addresses. They translate into the same physical address, and this could happen for example, If I have two processes that have some shared data. You'll recall that, we talked about the idea of 2 processes. If they have to corporate towards a common objective, they might be able to corporate by actually having shared variables and the operating system might provide support for that by allowing the mappings of the two virtual address of those two variables, to be mapped into the same physical address.

So, this possibility does apparently help us in writing cooperating programs made up of many processes. But, if one does have a situation where there could be two virtual addresses they translate into the same physical address. What does this mean from the from the perspective of a virtual address cache? It now means that the two different virtual addresses may be virtual address zero and virtual address one, which actually refer to the same piece of data, because they have the same physical address differ in the bit patterns. And therefore, they would actually end up occupying different locations in a virtual address cache. What does that mean in terms of the data? It means that, I could actually have two copies of the same piece of data inside a virtual address cache, under its different names and this, under one of the names I might modify that piece of data.

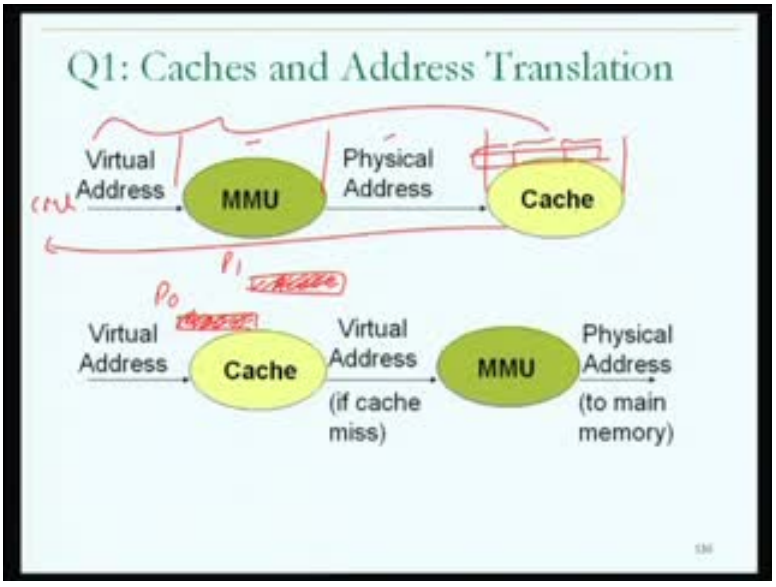
And therefore, I could have a situation where the shared variable has one value from the perspective of process p one and another value from the perspective of process p two and that is a dangerous situation to have, and could arise whenever we have more than one copy of a block inside cache.

So, on all of these counts, it looks like, having a virtual address cache may complicate the situation and not necessarily be a good idea. When we talked about virtual address cache in the previous slide, it looks like the virtual address cache was a clear winner over the physical address cache. Why? Because the physical address cache has a much longer hit time and the virtual address cache has this benefit that you do not even have to do the address translation if it is a cache hit.

(Refer Slide Time: 53:28)



(Refer Slide Time: 53:31)



So, looks like a clear choice, a clear beneficial choice but, when we looked into the mechanics, what has to happen behind the scenes? We understand that both of these have some negatives, and that we do need to look into this little bit further, which is what we will do in the next lecture. In the next lecture, I will actually go back and complete our discussion of these two questions. We will close our discussion of question one by talking about what we might expect to see in the different caches of modern processers. We have still not fully understood whether we should expect to see physical address caches or virtual address caches.

But, we do understand now that there are these two possibilities, and that the real processors that we deal with might be using either physical address caches or virtual address caches, and we will then come to the second question. We will talk a little more about what is happening in the pipe lines of modern processers. So, we will postpone both of these two topics to the next lecture and I will stop here for today. Thank you.