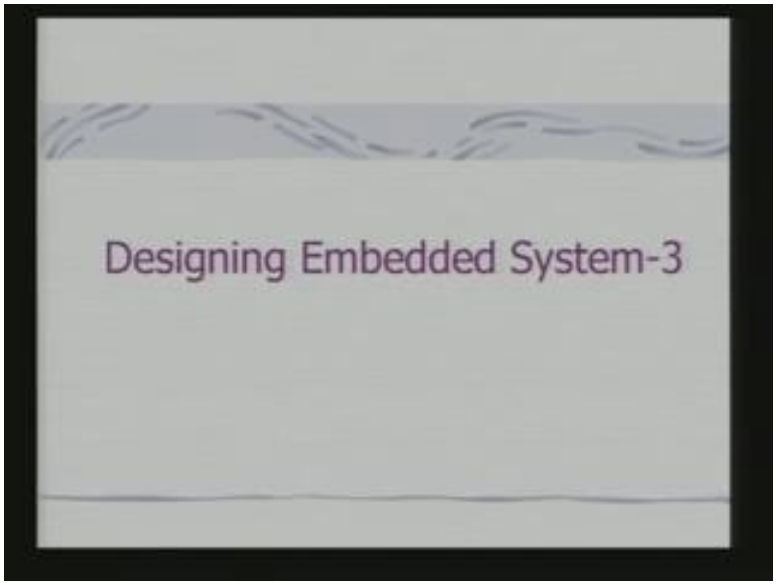
Embedded Systems
Dr.Santanu Chaudhury
Department of Electrical Engineering
Indian Institute Technology, Delhi

Lecture – 30
Designing Embedded Systems – III

So far we have discussed the different tools for representing the design specification. Today, we shall look into the design process.

(Refer Slide Time: 01:11)



And there also we shall see how the different kinds of representations schemes are adopted to carry on with the design.

(Refer Slide Time: 01:22)

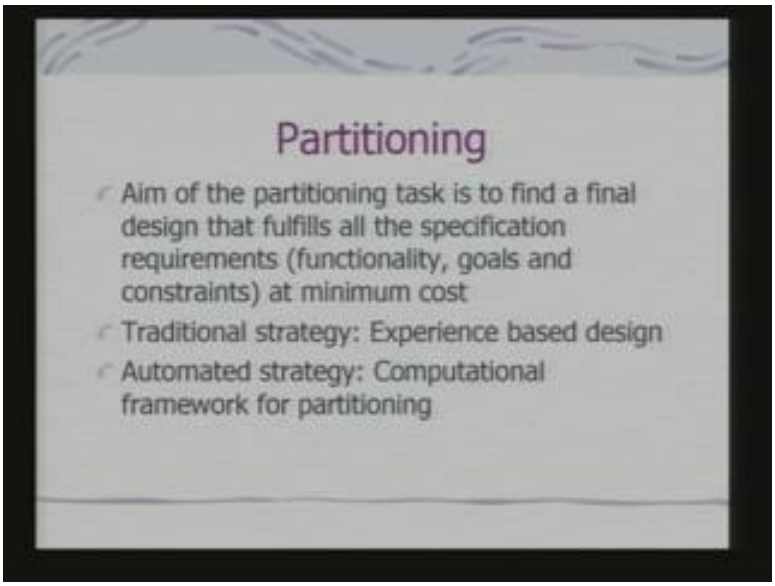
Fundamentally, an embedded system design implies both hardware and software design although there are other aspects of the design which, I had already mentioned earlier. So, hardware software co-design means joint design of both hardware and software architecture. This implies that we need to do hardware and software partitioning. Hardware and software partitioning, deals with the assignment of parts of the system description to heterogeneous implementation units.

So, when we are represented the overall designs then; we have identified the different components. If you remember the UML representation, we have got say the activity diagram which has represented the different tasks along with the concurrency indications. The task diagram is also specification of the tasks that are to be performed in a system to meet the functional requirements, as well as the constraints can be imposed there. Now, once we have that kind of a representation, we need to map this in to hardware and software.

Now, mapping in to hardware means designing or use of special purpose hardware like ASIC's or it may be mapped on to FPGs. And FPGs are also used for rapid prototyping. If you talking about the software, the parts of the tasks which get mapped on to the software it means; they have been mapped on to the standard embedded microprocessor or a microcontroller.

So, software part of a tasks is typically mapped on to a standard microcontroller that you using. Those parts of the tasks which gets mapped on to the hardware to gets mapped on to ASIC's or any other special purpose hardware. Also you would look at the memory and other requirements through this process. So, the key issue for an embedded system design is this hardware software co-design, which implies hardware and software partitioning.

(Refer Slide Time: 03:54)



So, partitioning tasks basic aim is to find a final design that fulfills all the specification requirements: functionality, goals and constraints at minimum costs. You can all understand why this partitioning requirement will come in. If we can map everything; all aspects of the tasks on to an embedded microprocessor or a microcontroller satisfying all the constraints. The most significant constraints will be the timing constraints. Then obviously, they do not need to use a special purpose hardware.

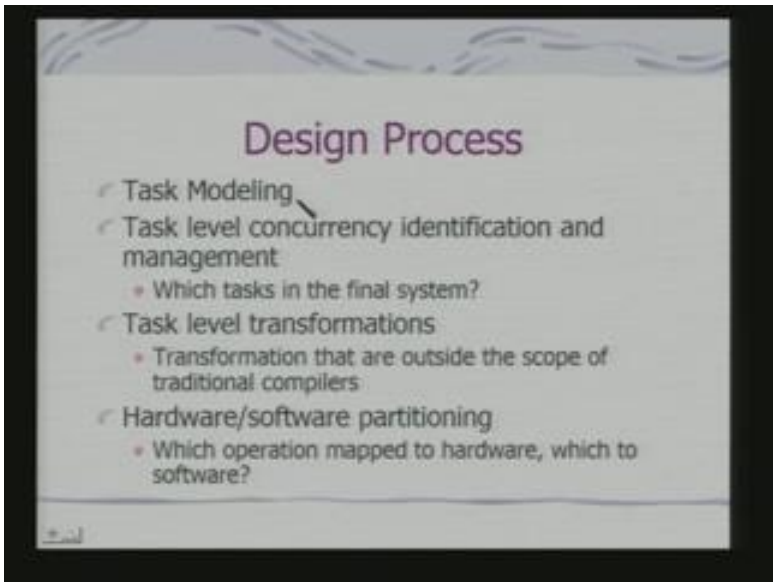
The question of mapping of the tasks on to special purpose hardware will not arise. Also it may be the case we might need to select an architecture for doing the software mapping that becomes what is called an architecture mapping problem. That means, which processor to use simply put in which processor to use to realize certain functionalities in software.

So; that means, after we have finish the partitioning we actually should get a design which is likely to satisfy the design constraints. As well as provide the target

functionality. The traditional strategy would be an experienced based design that is you manually use it, try to analyze and do it.

Now, this experience based design may also involved usage of variety of computer aided design tools. On the other hand, you can think in terms of a completely automated strategy where you get a computational framework for partitioning and the task can be done completely in an automatic fashion.

(Refer Slide Time: 05:54)

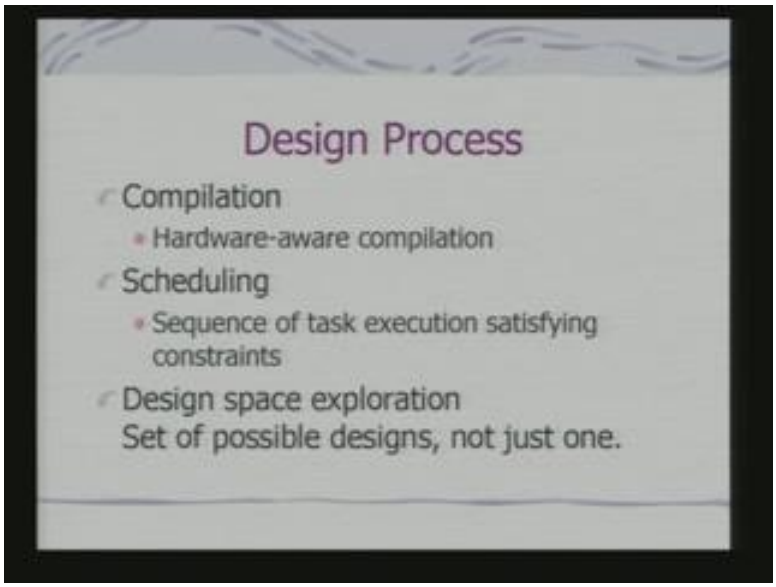


So, design process involves therefore, task modeling using any kind of modeling tools. Then task level concurrency identification and management, which tasks are in the final system? How the tasks are to be put in concurrent models or concurrent modules? We also need to task level transformations. Transformations which are outside the scope of traditional compliers may be. Or even if they can be done by the compilers you should know what kind of transformation are possible, which can be applied to the tasks.

What are the objective of the transformations? Objective of the transformation to make the system more efficient. That should be followed by the hardware software partitioning. That means till this point, we are dealing with the task representation. We are trying to drive an optimal representation of the task. Then we should look at the hardware software partitioning that is, which operation is mapped to hardware which to software.

In fact, when we look at use of automatic tools many of this transformation and the partitioning aspects can be put together and combined as part of facilities provided by the tool.

(Refer Slide Time: 07:19)



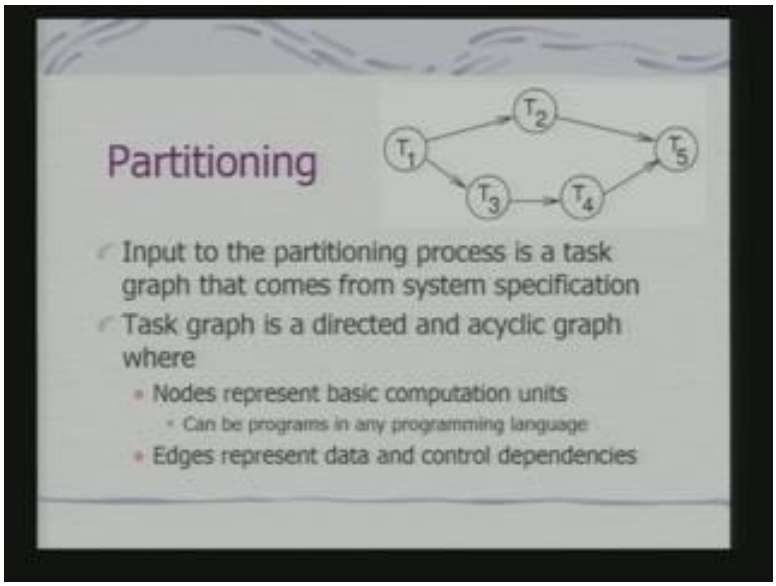
Next when you have mapped on to this hardware and software you need to do compilation. Because; you have got a code which has to be compiled and mapped on to the actual processor. So, this is what you call a hardware-aware compilation. That is making the compiler aware of the hardware features. Then, we need to have scheduling. That is, sequence of task execution, execution of the tasks such that, the constraints are satisfied. How to schedule the tasks? The scheduling of the task could also mean scheduling of the tasks, if at all required on the special purpose hardware.

Now, all this process involves what is called design space explorations. Design space consists of alternative possibilities. So, we need to search among this alternate possibility to find out a design which not only meets the functional requirements and satisfies the design constraints; but may be cost wise minimum.

So, take an example: if I am using for an application a 32 bit processor. With the 32 bit processor like I can definitely meet my timing constraints. So, that is one design option. There could be other design option, another design option could be using a 16 bit processor and use may be a special purpose hardware along with it. And meet the functionality as well as the design constraints, timing constraints which one to choose?

May be the 16 bit processor with special purpose hardware finally; turn out to be less costly. There may be other issues also; that means, both may satisfy the timing requirements. And we would like to choose one which would likely to consume minimum power. So, that is what we referred to as the design space exploration.

(Refer Slide Time: 09:27)

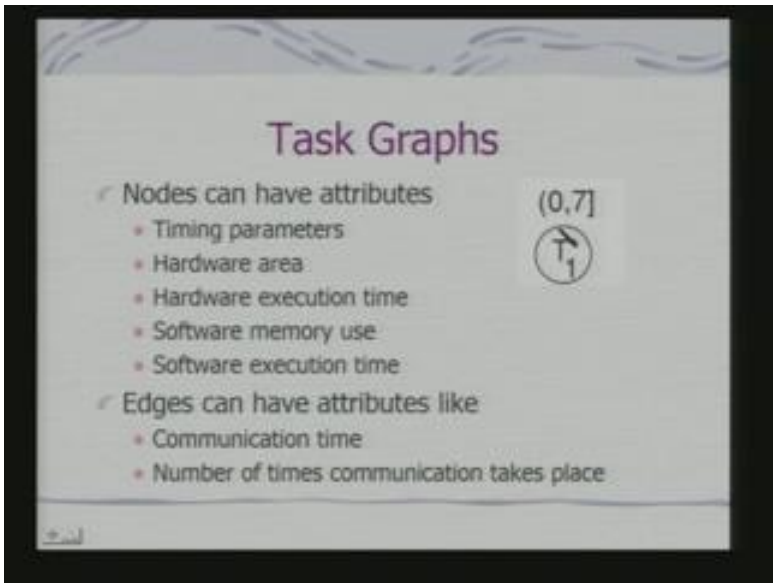


For the purpose of partitioning, we look at the tasks. And we say that we shall have a task graph. Now, these task graph is very similar to what you have seen activity diagram in terms of UML notation as well as task diagram in UML notation. But we are referring to this task graph as an independent representation model. So, input to the partitioning process is the task graph that is comes from system specification.

In fact; the system specification we can represent using a variety of tools, using UML tools. And from there we can derive the task graph if required. Task graph is the directed and acyclic graph where; nodes represent basic computation units can be programmed in any programming language. Because if I am using a programming language to specify the computational task here. What I am getting? I am getting an absolute unambiguous representation of the computation to be done at the node; that means to be done as part of the task element. Edges represent data and control dependencies; that means data is required here. So, there is this dependencies between the tasks. Now your basic partitioning problem is what? Mapping of this tasks to either hardware or software.

Now, we may not like to go straight away or jump in to this partitioning process. Because we might do some transformations on the task graph itself to make the whole process more efficient.

(Refer Slide Time: 11:25)



So, the task graph has got nodes. Nodes we have associated with attributes to specify the design constraints. This is an example of an attribute specification timing parameters. So, I say that T 1 arrives at 0 that is the starting point and its deadline is 7; that means, it must be finished within a time unit 7. That I can put in appropriate units associated with the time interval constraints.

Now, if we are talking in terms of mapping of this task graphs in to hardware elements there can be other attributes which may become relevant. Hardware area, if you are designing an ASIC. What area is required? This may be measure in terms of a total number of gates required. Gates would translate the number of transistors.

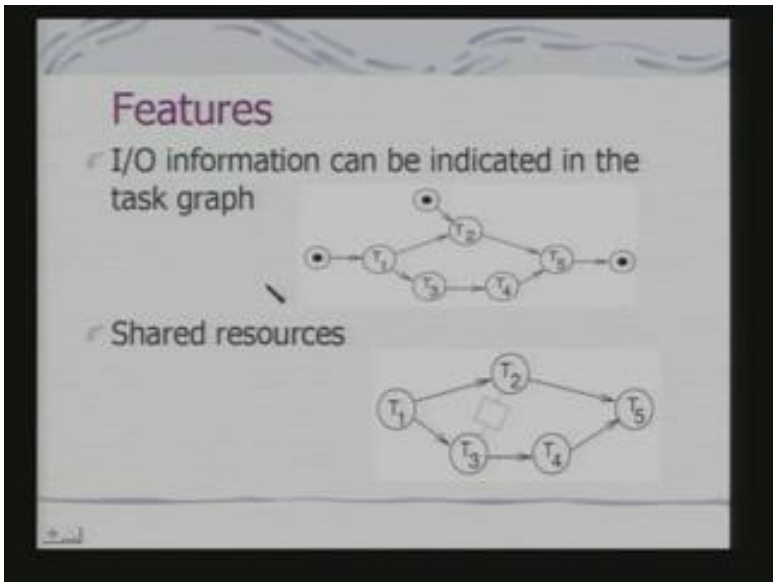
What is the integration technology that you using? That will give you the total hardware area that may be required for implementation of a node in a hardware. Then hardware execution time. If you are mapping in to software, we can estimate what will be the memory requirement by this node. Then what will be the software execution time. The software execution time will come from architecture mapping; that means, depending on which processor that you using.

That means you can realize is that, the moment I am searching for different alternate designs. That is I am exploring my design space. I can have the task graph associated with different attributes. So, I can have different instances of the same task graph with different value of the attributes associated with the nodes depending on the kind of partitioning that I am considering.

Edges can also have an attributes, like communication time. Because if it is, if you are thinking in terms of mapping the task to be different processors. And if these processors are connected via network, then what would be the communication time? Then number of times the communication takes place. How many numbers of times communication has to be done?

In fact we can also have with the nodes the number of times the task has to be repeated. Because if you remember, we say that this tasks graph we have modeling as an acyclic graph. So, the reputation of the computation has to be indicated as an attribute for the node. There are other features which can be used in the context of the task graph.

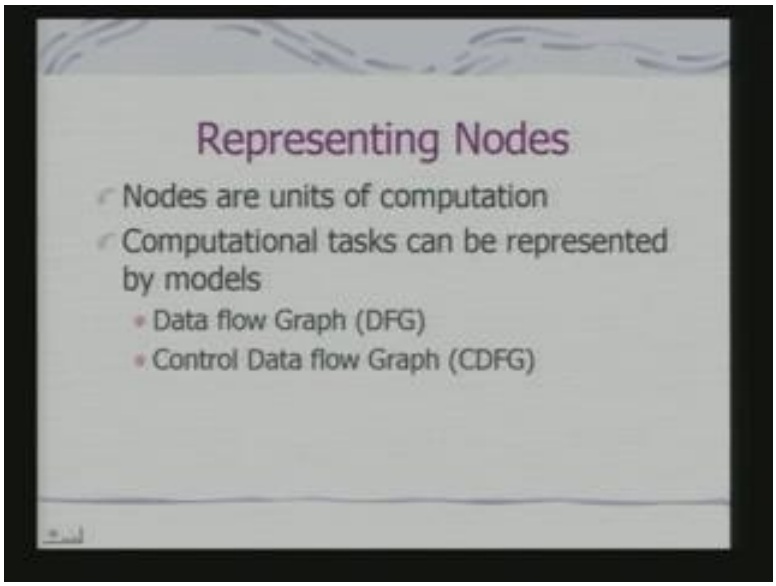
(Refer Slide Time: 14:12)



There would be IO operations occurring. So, how to indicate IO operation? Which task will request IO? So, that is indicated by this kind of a node. This is indicating the T 1 requires an input, T 2 requires an input and T 5 produces an output. Tasks can actually share resources. In fact if you see that these T 2 and T 3 as not have dependencies. So, they can be concurrently scheduled.

If they are concurrently scheduled, is it they will require a shared resource. If you want to indicate that they are dependent on a shared resource. We can indicate the shared resource in the task graph itself with these kind of a model. So, this indicates that there has to be shared memory between the tasks T 3 and T 2.

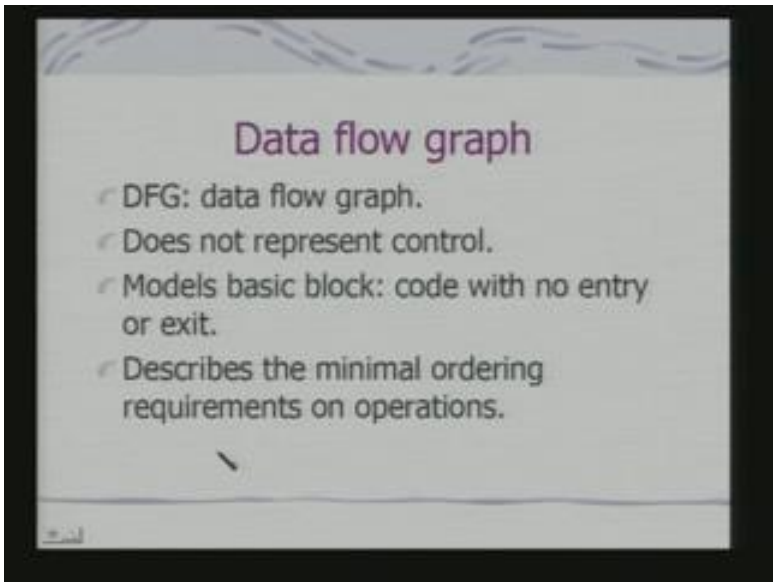
(Refer Slide Time: 15:16)



Next comes, how do you represents the node? You have talked about attributes of the nodes and we said that the nodes really represent computations. And if the node represent computations. That computation can be expressed in terms of any programming language.

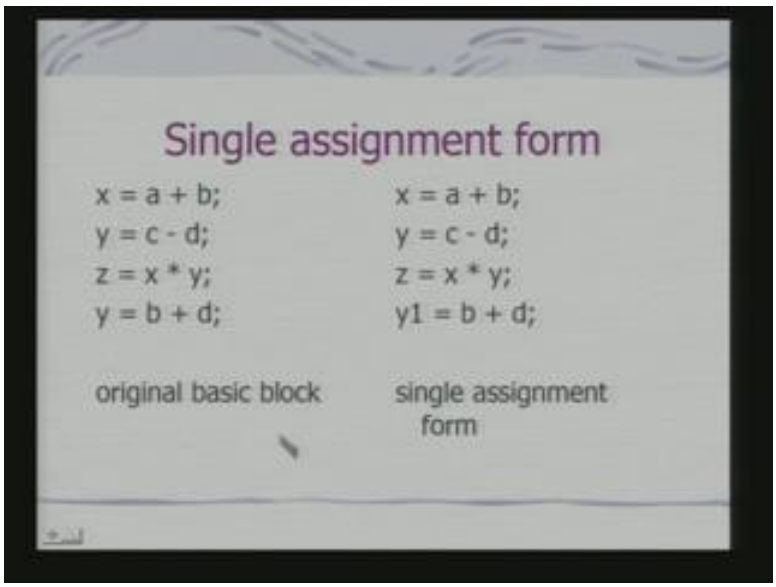
In this context, we are considering in today's lecture we are considering the representation in terms of imperative languages; that is procedural languages. And there can be models for representing computations that is to be carried out at the nodes. The two basic models which are used one is your data flow graph model another is control data flow graph models. In fact these models are also used by compilers for the purpose of optimization of the code. So, what is the data flow graph?

(Refer Slide Time: 16:16)



Data flow graph basically represents the sequence in which the data is processed by a code. It does not represent therefore, control and it models that the basic block: the code with no entry or exit, formally no entry or exit. There has to be although an entry point and exit point. And through this you can describe the minimal ordering requirements on operations.

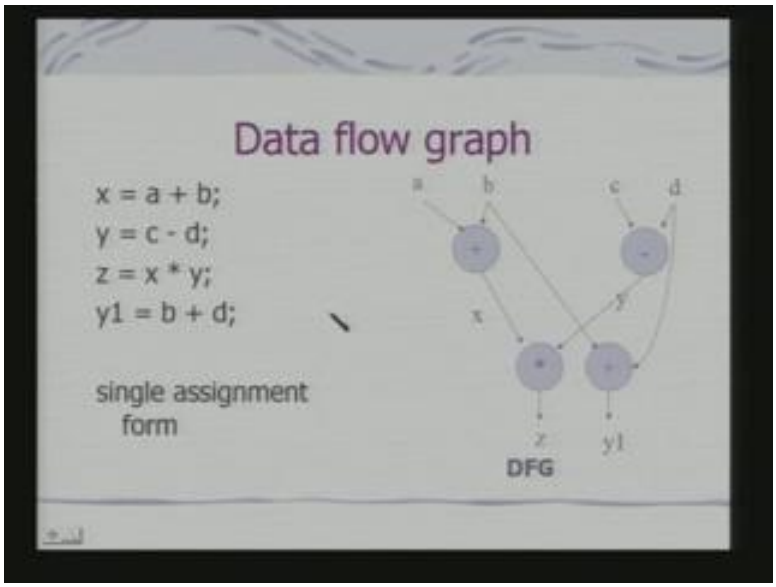
(Refer Slide Time: 16:50)



So, let us take an example: this is the simple assignment set of assignment statements. So, these can be the computation needed to be done at a node. So, this is a basic block.

Now these blocks can be rewritten, where I have introduced another variable. Why? Because y was being used twice instead of that I am using another variable y1. So, that each variable has got a single assignment. Now, what is the implication of this?

(Refer Slide Time: 17:39)



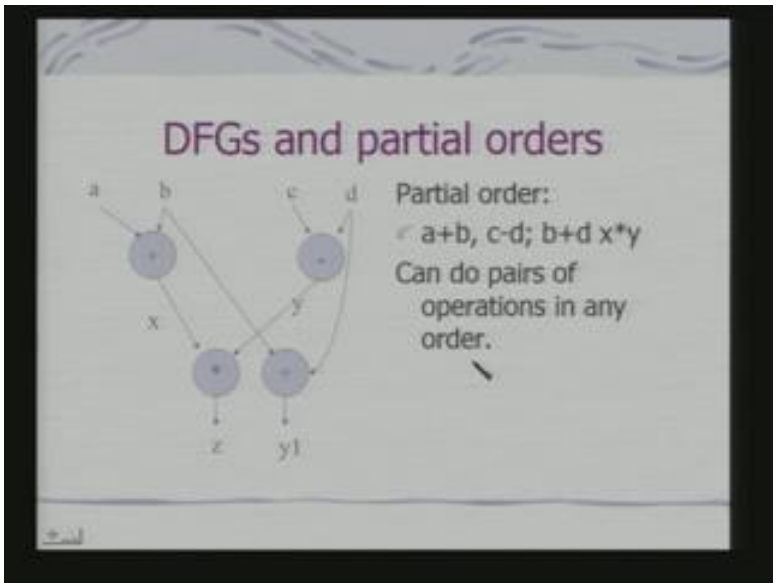
Let us look at the basic data flow graph. If you have this you can understand that this is the sequence of assignments and this is your data flow graph. What did it showing you is that, if you have got variables a b and c d then these operations to take place. And this is indicating a dependence and this is another dependence with respect to y.

So, what you can realize is that the moment that I have got this y replaced by y 1. I can meet this computation independent of this step. Because, otherwise you have to synchronize between the two. I have to do y equal to c minus d first and then I have to do z. Then only, I could have done y equal to b plus d. By doing a single assignment, I have reduced the dependencies.

Now, these dependencies can be use for a optimization of the code, scheduling of the operations, reordering of the instructions. When you are mapping the computation on to a software unit to be done on a embedded processor. This can be even used if I am trying to design a hardware to implement this set of operations. But the basic point is that, this data flow graph is a model to represent this computation.

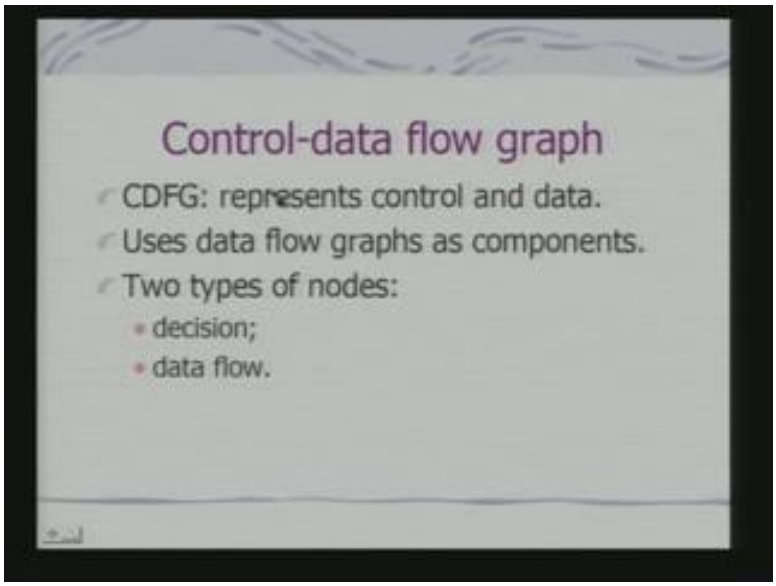
So, what we are talking about? We are talking about a task graph where each node represents a computation. And we can represent the computation to be done at each node in the task graph by a data flow graph. Data flow graph is one such model.

(Refer Slide Time: 19:43)



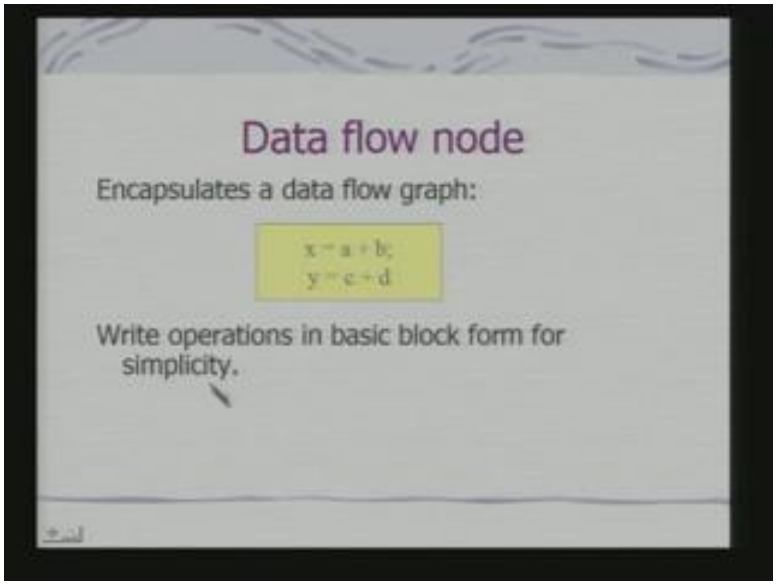
Now we can have more such models. So, here is the partial order that you have got. So, effectively what does it mean, can do pairs of operations in any order. So; that means, the instruction, the point I was telling the instruction scheduling can be done. And why I would like to do an instruction scheduling? May be, to optimally use my pipeline processor. So, another representation is control-data flow graph which is another model and CDFG represents control and data.

(Refer Slide Time: 20:10)



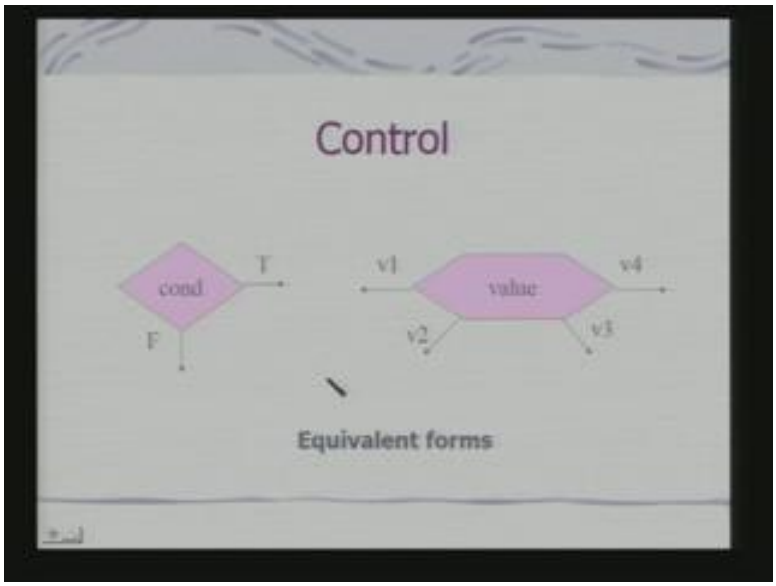
In fact any computation is truly can be modeled by control data flow graph. Because tasks themselves are just not data flows, there would be control decisions which will be involved in each task element. So, a task element would be naturally modeled by a control data flow graph. There are two types of nodes. Obviously, now in a control data flow graph one is a decision node the other is data flow node. Data flow nodes are nothing but actually your data flow graph being represented as a block.

(Refer Slide Time: 20:55)



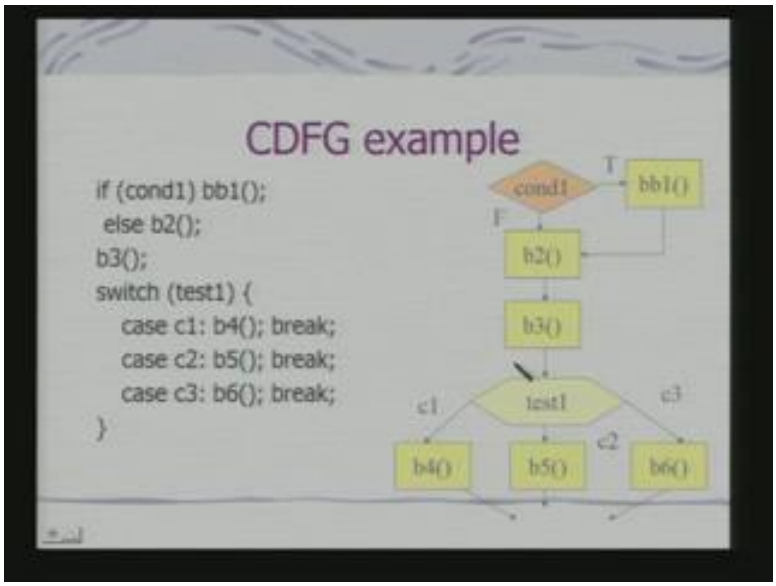
So, a data flow node encapsulates a data flow graph and write operations in basic block form for simplicity in the CDFG. Now, this can be again decomposed in to your data flow graph. The graph in terms of node where each node represents an operation. And I have input values with respect to that graphical representation.

(Refer Slide Time: 21:21)



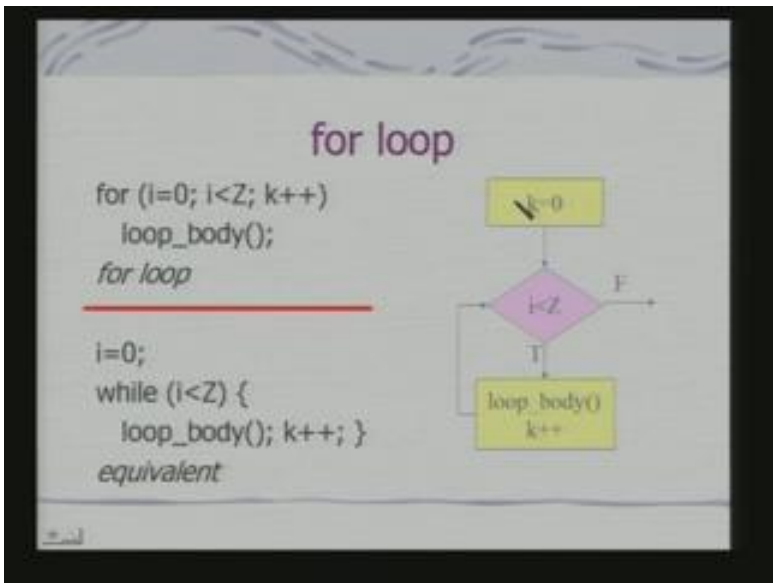
The control steps or the decision steps can be shown in these kind of a form. So, you can find out it is pretty similar conceptually that of your flow basically, your flow chart representation. So, you have got this condition false and true. This can be multi value conditions like case statements. They can be represented in any equivalent form, because a combination of these can be represented by this kind of a structure.

(Refer Slide Time: 21:54)



So, if take an example: This is an example of a case statement. If then else kind of case statement. So, I can represent these as a control data flow graph. Now each of this are nothing but functions. Now corresponding to the functions I can again have a CDFG. And that CDFG will replace this function; fine. And if I have a simple assignment kind of a thing then that assignment block can be modeled as the data flow graph.

(Refer Slide Time: 22:36)



This is an example of a for loop. So, I can represent a for loop or a while loop. You know their equivalently represented. And these will be their representation in a control data

flow graph. So, here this is a simple single assignment statement which is the data flow node. Now, what to do with these kind of representations? The basic idea is once we have these kind of representations. We can now try to look at various kinds of transformations, which can be applied on to this model to make your code of the computation task at the nodes more efficient. This is the basic objective of having this kind of modeling techniques to represent computation task at the nodes.

(Refer Slide Time: 23:31)

So; we say that we shall do some kind of high level transformation. It is not that, all this transformation will be applicable under all possible conditions. It may be that some of the transformations are applicable. Some of the transformations are not applicable. So, depending on the problem that you are considering, you have to look at this possible transformation of the computations scheduled to be done at each task node.

(Refer Slide Time: 24:08)

First we shall look at loop transformations. That is whenever there is a loop, whether I can change this loops. So, loop transformation the basic objective of the loop transformation is to reduce loop overhead, increase opportunity for pipelining and improve memory system performance. Because if I can reorganize the loops all this advantages can become operand. If these advantages become operands, what is the consequence? If I reduce loop overhead, the time required for execution of a loop at a node will be less with respect to an architecture mapping.

That means, after it being mapped on to a particular processor then, I can meet the design constraints or the timing constraints for the task. Otherwise the problem would have been that, I need to think about a special purpose hardware to implement that code such that the timing constraint is satisfied.

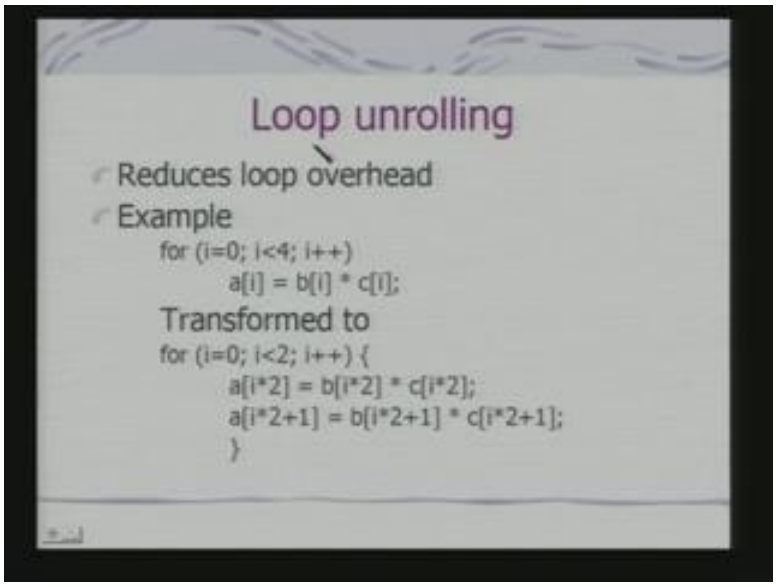
So, basically when understand that these transformations are critical. So that your hardware cost may be reduced and you can be able to meet your timing constraints. The timing constraint is not only the one issue, other issue should be energy. If you remember when we discussed cache memory we said that, cache misses would be a source of energy consumption.

So, I would like to maximize cache hits. So, is it that by this transformation can I maximize cache hits if I look at a software implementation. And if I maximize cache hits, I have savings both in times of time as well as that of the energy. So, if I can do that

and I can meet the constraints. I shall not map possibly the task on to a special purpose hardware. I shall do the task on my processor that I have chosen. So, that it gets mapped on to a software.

So, these transformations are critical for these purposes. So, let us look at the different kind of transformations that we can use. The simplest form of the transformation is loop unrolling.

(Refer Slide Time: 26:37)



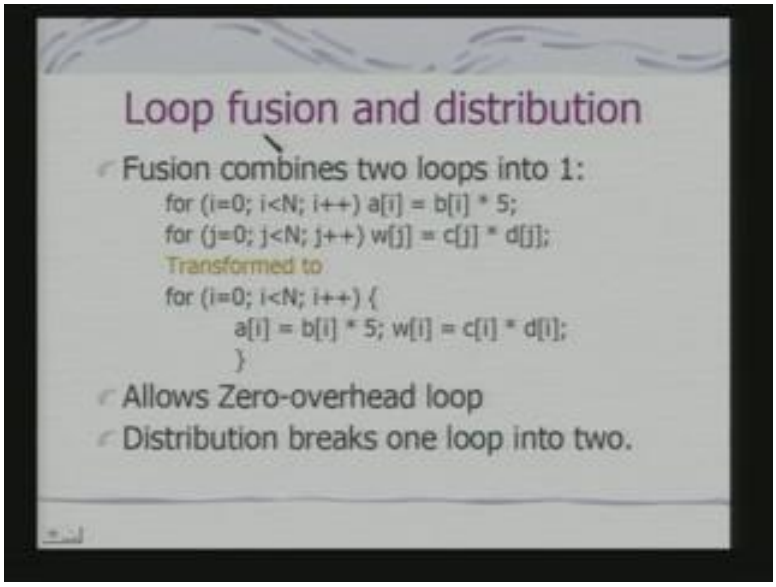
So, this is an example to explain the loop unrolling. The objective of loop unrolling is to reduce loop overhead. This is a very simple loop. Now, what is loop overhead? Loop overhead comes in corresponding to these statements because these statement will be translated to a set of machine level instructions. That instruction, those set of instructions should be executed each time the loop is executed. If I can reduce the number of those instructions or if I can reduce the number of times instruction being executed, and then I can reduce the loop overhead.

So, let us look at these possible solutions. So, what has been done here? I have reduced the number of iterations. In the process of doing that, what have I done? I have actually multiplied, used this array in this as can may different form. I multiplied by 2 and this is 2 plus 1.

So, effectively I can do the same computation using less number of loop iterations. So, the loop ever overhead goes down. Now, we can say that we have introduced this many multiplications. But actually if you remember our architecture, this is not really multiplication. So, they can be done as part of the same instruction itself, because of the barrel shifter.

So, I get a much better optimized code for loop. So this is the basic idea of loop unrolling, fine. Reducing the overhead of loop execution, because loops occupy a very large part of your execution time period. Because you actually executes codes in loops. If you profile, we will find that the time required for different loops vary, fine. And major part of your time for execution is pending loops. So, if you can really optimize the loops, then we can really get a advantage in terms of meeting the timing constraints of the design. Next operation or transformation that you can have is loop fusion.

(Refer Slide Time: 29:10)



So, if you are looking at the loop fusion, fusion means combination of the two loops.. So, here we have got example of two loops and; obviously, the advantage that I shall get is that these code for corresponding to the loop overhead will be executed once. And the interesting feature is if it is executed once then these loop overhead is being reduced to 0. These instructions had a loop overhead and this can be reduced to 0.

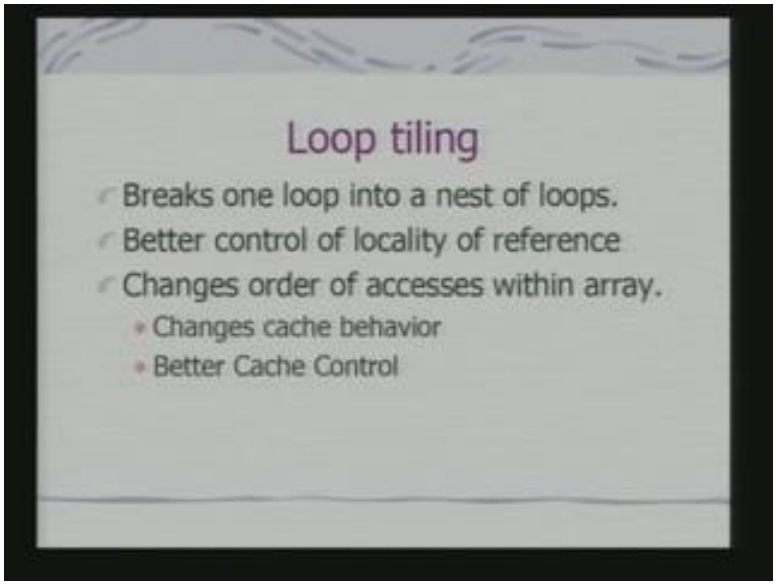
But please keep in mind that I have although given this kind of examples one after another. And it became obvious that we can combine this in to two loops, these two

loops in to one. But this is not always obvious in the court. By looking at your control data flow graph, you can find out whether there are dependents between such loops. If there are no such dependencies, you can decide to combine the loops. And get what is called a zero overhead loop for some of the cases.

In fact, the other interesting thing is that once we have done that literally, I can reduce to zero overhead. Because, if you remember many of the DSP processors have support for zero overhead loop. You have to initialize the corresponding counters or the registers with the value of the number of counts and in hardware itself the loop implementation is provided. So, once I can do this combination; I can therefore, further optimize the code depending on the architecture that I am using.

So, that is why this kind of analysis of the fusion becomes critically important. The other part of the fusion would be fission or breaking up the loop or distribution of loop in to two.

(Refer Slide Time: 31:25)



So, these one of this approaches which is used is called loop tiling. So, what happens in loop tilling? In loop tilling you break one loop in to a nest of loops. Now, what is the objective of doing it? You can say that if I am breaking one loop into a nest of loops. what I am doing? I am actually increasing the loop overhead.

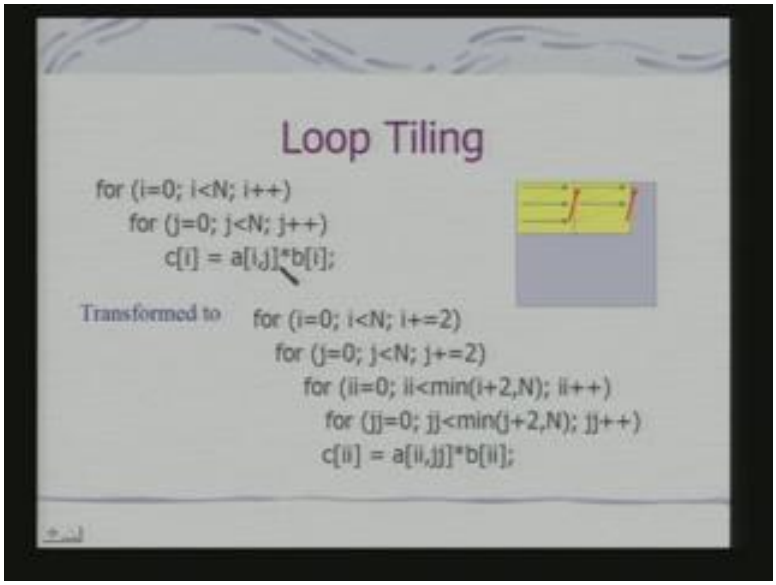
So, whatever I have discussed so far am I contradicting it? It is not so, it may be required to have better control of locality of reference. So, what is meant by, why locality of reference is important? Locality of references is important for cache access. Say for example, if I have a big array and I am doing some computation at an each element of that array. What happens?

After some set of updates there may be a cache conflict and cache has to be replaced. Again may be in the next statement I have to go back to the initial elements again a cache conflict to come. Because the by nature the single loop violates the condition of locality of reference.

So, I might like to split that loop in to multiple nested loops, such that in each nested loop I satisfy the locality of reference. That means, once that inner nest is completed then only the cache violation can come in. So, I am liking to use the cache; that means, I would like to restructure the code. So, that I can have a maximum use of the data, that I have loaded on to the cache.

So, it changes in that case it can changes order of access within an array. And it obviously, if we change the order of access, changes cache behavior. And if I am looking for change in a cache behavior I shall be interested in improving its behavior and a better cache control. We shall see an example now. Let us look at this.

(Refer Slide Time: 33:51)



This is a very simple operation. That is I am multiplying this is two elements of an array. This is a two dimensional array and this is a one dimension array right. Now, we can realize that if I am doing so what can happen? I can actually miss out on the cache because this array a i j is a 2 D array. So, I might be going depending on whether, it is organized on a column major or a row major form depending on that it might be so that there are cache conflicts.

So, while the loop is executed and I am processing. So, this is j is a inner loop when I am processing this there may be multiple times when the cache conflict occurs. Because I have to load the values corresponding to the jj index changes. And again I come in. So, there may be multiple conflicts here.

So, what I would like to restructure that array. By restructure the array what I am telling is I talk in terms of tiles. This is a tile what I am telling here is that I will not like to cover each and every element here. What is happening? If this j indicates the row. So, I am finishing off the row first, then I am loading the next column. But if I restructure the array, I can actually do this operation with respect to part of the row. Or I can do it for one row and not all columns, but part of the columns.

So, here the example is: this is a one complete row fine, what I am doing? I am doing part of the row, coming back doing the next row, then doing the next row. So, I am moving along the column before completing one complete row fine. And then, I am going back and again doing the computation.

So, I have created what we called tiles. And I have changed the order of access of the array. Now, how to do that? If we looking here, the way I have reorganized the indices. What it is providing for? It is providing for access of parts of it.

So, I am now if you see here, this I am incrementing i in steps of 2 and not 1. I am incrementing j in steps of 2. And here I am assigning this ii which is a new index and jj which is again a new index which I have brought in. And using this two indices I have introduce an inner nested loop. In fact in the inner nested loop, the ii for a particular value of j, if I assign a particular value of j and particular value of i, this goes from i plus 2, the minimum of i plus 2 to n.

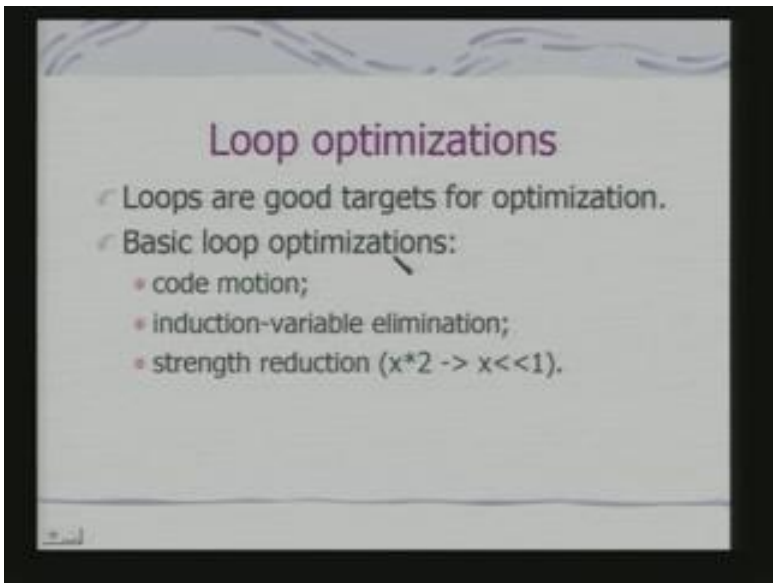
So, I have created tiles of size effectively from 2, i equal to 0, if I say i equal to 0 and j equal to 0. The beginning of the loop then ii is 0, jj is 0 and what will be the terminating value? Terminating value is 2. So, it will go to 0 1 0 1. So, I am basically looking at the tiles.

Tiles consisting of the two elements for row as well as that of column is this clear. So, I have actually created a tiles because, the array organization may be such that I can fit these data in to a cache. So, once I have completed this data, I won't like or I won't need to come back to use this tile again. This is the basic advantage.

So, what we have got in the process? We have increase the loop overhead but, I am reusing the data which is there in this tile. That means, I am completely doing the tasks that I needed to do with the data in this tile. Since I have finished that task, I can move over to the other tile and reduce the number of cache conflicts.

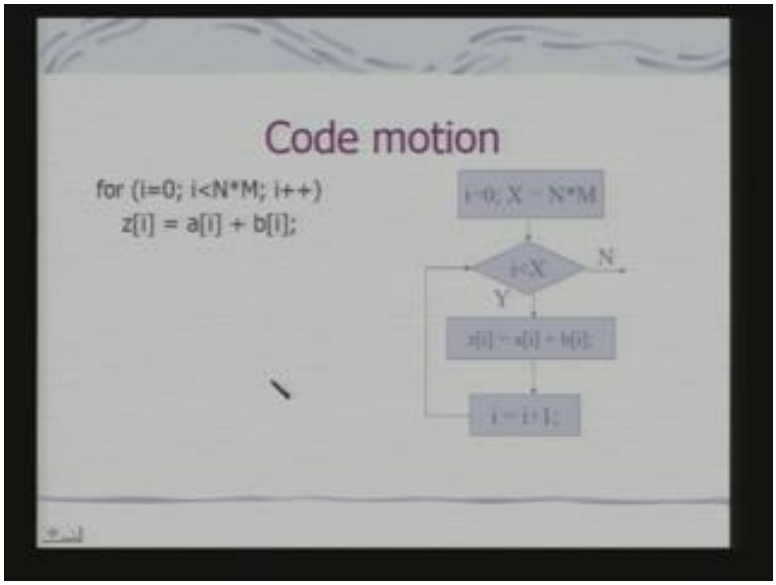
So, the point that you should keep in mind, that these loop transformations that we have talking about all of them cannot be blindly used. They have to be used depending on requirements. I advocated first the loop unrolling to reduce the loop overhead. Now, I am talking about splitting the loops increasing the loop overhead definitely, but it would actually lead to improvement in performance because cache conflict reduces. There are very another ways of loop optimization as well, because loops are good target for optimization.

(Refer Slide Time: 39:20)



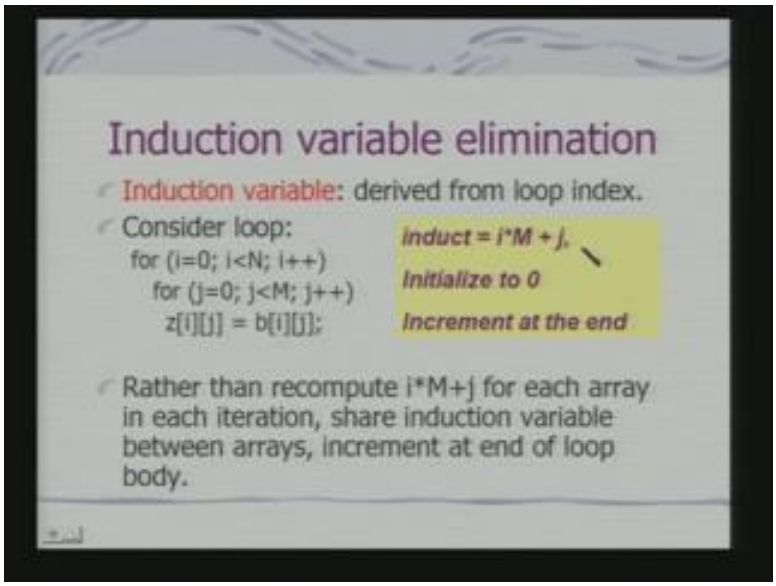
Because as I told you, a large number of time is large amount of time is actually used in the loops only. So, the loop optimization techniques which are followed are code motion, induction variable elimination, strength reduction. Now, many of these actually done as part of your compiler as well. So, might not need to do before the compiler comes in. The simplest thing is the code motion.

(Refer Slide Time: 39:49)



I did not do this evaluation every time as part of code execution; that means, a part of my loop overhead. So, I am removing from here all together and taking it outside. Okay so, this can be part of your initialization. So, I am reducing the number of multiplications that we need to do. The moment that I reduce the number of multiplication the time required would be less.

(Refer Slide Time: 40:21)



Then, what we call an induction variable elimination. An induction variable is typically derived from the loop index. Because, here you see that this is a two dimensional array z and b i j and I am doing an assignment using the same index pattern. So, for each assignment I need to compute, the address of this data element. The computation of an address would imply a multiplication. This is i into m plus j, if this is the organization of an array. So, for this address calculation I can, I need to compute this. So, this would be done as part of the code required for this assignment.

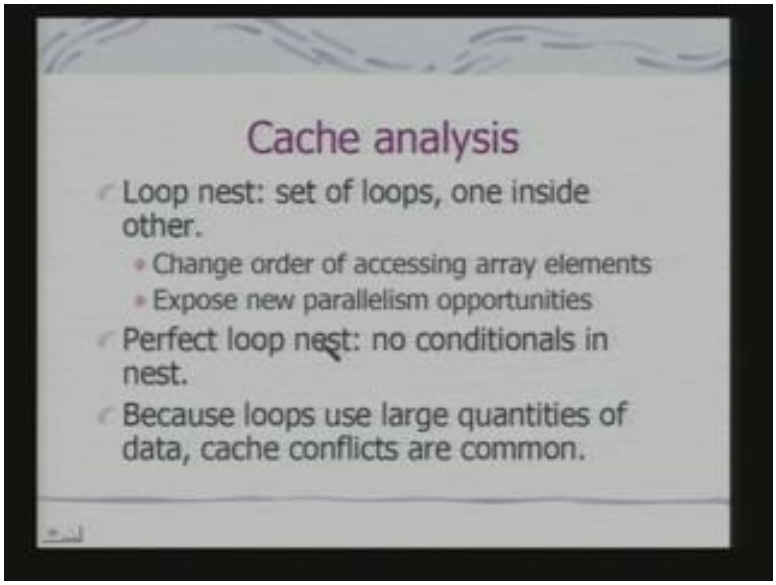
So, what I can do is I can use these expressions directly. Because this expression will be computed twice. Once for z, once for b, if I directly use this code for mapping on to the software, on to a processor. Now, I might like to avoid that because, that would be a redundant computation so, do that. So, you identify induct variable, initialize to zero and increment at the end.

So, if I do that then I need not do this computation for this assignment of every step. So, what we say rather than recompute i into m plus j for each array in each iteration. Share induction variable between arrays and increment at end of the loop.

So, this induction variable elimination can reduce this kind of timings. And in fact, you will find that these kind of a operations are not very uncommon. Because, if you remember the code that you write for, say your matrix multiplication. There you would

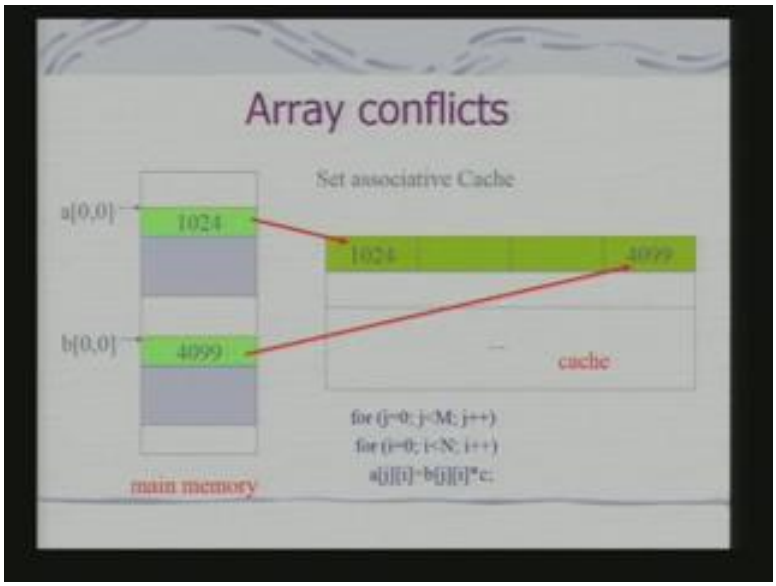
like to have similar kind of constructs and matrix multiplications a very common code for variety of operations.

(Refer Slide Time: 42:33)



Next thing is cache analysis. In fact the first part of the cache analysis we have discussed in the context of the loop nesting itself. That we would do nesting one inside the other and change order of accessing array elements. And expose new parallelism opportunities reduce the cache conflicts. And perfect loop nest is be the one there are no conditionals in the nest. If there are conditionals there may be data may not be properly be used. And because loops use large quantities of data, cache conflicts are common. And that is why this loop nesting is an important operation for dealing with this cache behavior.

(Refer Slide Time: 43:17)



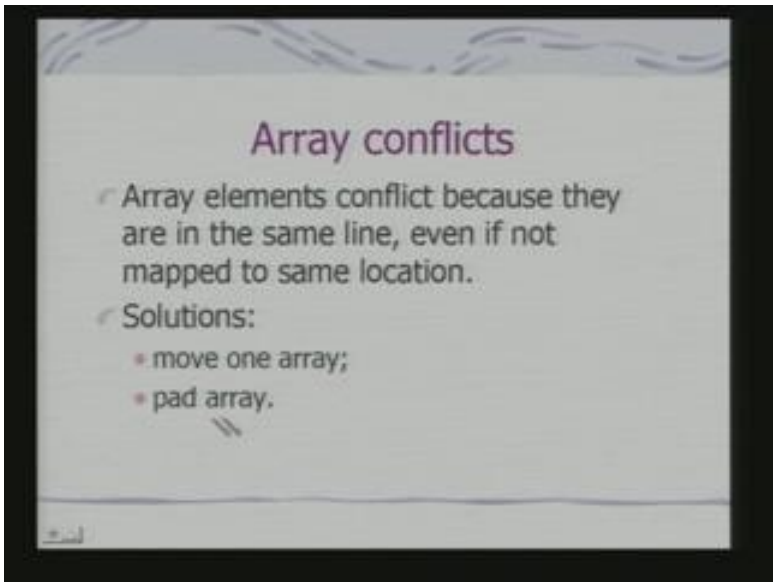
Let us look at another example. This is of an array conflict. We have all studied set associative cache. Let us consider that, I am dealing with two arrays a and b fine. If I am dealing with two arrays a and b and if it is a set associative cache. The mapping I am showing here is that this array a 0 0 starts a location 1024, b 0 0 starts at location 4099. And the mapping is such that the three elements 1024 to these.

In fact, this whole line the cache is mapped with respect to this; it is a set associative cache. So, a set gets mapped. So, this entire set gets mapped here and interestingly that b 0 0 gets mapped to this point; that means, the locations around b 0 0 would be mapped to this line as well.

So; that means, if I am accessing the array element a 0 0. I have to access actually this entire cache; that means, the data is loaded; this entire set gets loaded staring from 1024. when I am accessing b that same set in the cache gets loaded with entries at memory location before b 0 0 and ending in first location in b 0 0. So, when I am executing this loop what happens at each and every step, I shall have a array conflict. And the b's. So, which is not at all desirable? So, what is required, the solution could be reorganize the arrays, reorganize the data structure.

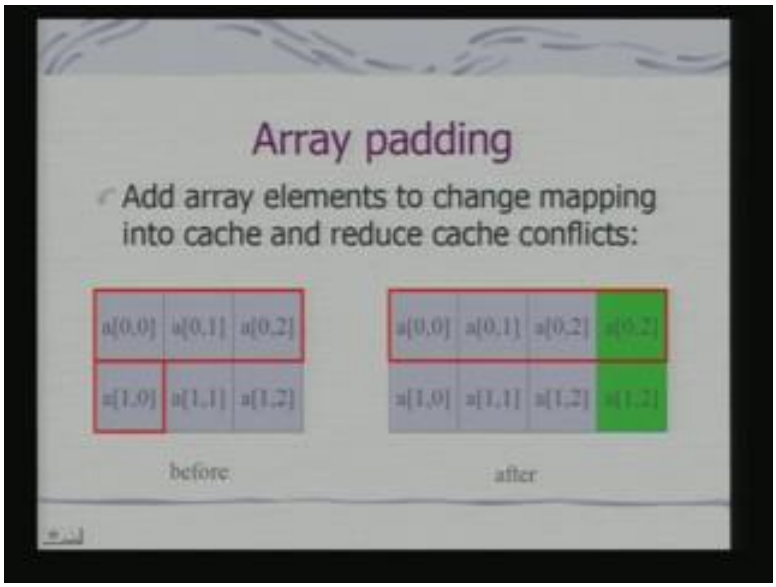
Now, that reorganization implies what? One possibility is change the address of b 0 0. Try to move around b 0 0. But that may not be always possible because of other arrays and other constraints. So, other option is, what you called padding.

(Refer Slide Time: 45:28)



You pad an array so; that means, you add additional elements an array. So, the solution of this kind of a conflict is move an array or pad an array. So, let us see what you really mean by padding an array.

(Refer Slide Time: 45:48)



This we shall care to that example of this two dimensional array a 0 0. Let us look at this array a 0 0. Now this has got three elements per row and therefore, if we look at this, I have got a 0 0, a 0 1, a 0 2 and a 1 0. So, if I have a mapping to the cache, the cache map is in this way. That means, these four elements gets mapped on one cache line.

Now, what is the problem can happen? If I am actually access in this row, there won't be a cache conflict. But for each access of the next row, there would be two cache loads. If I am loading this for each of this row there will be two cache rows or cache conflicts. Is it clear. But because you are actually having the mapping such that, on a cache line elements from two distinct rows of array is getting mapped.

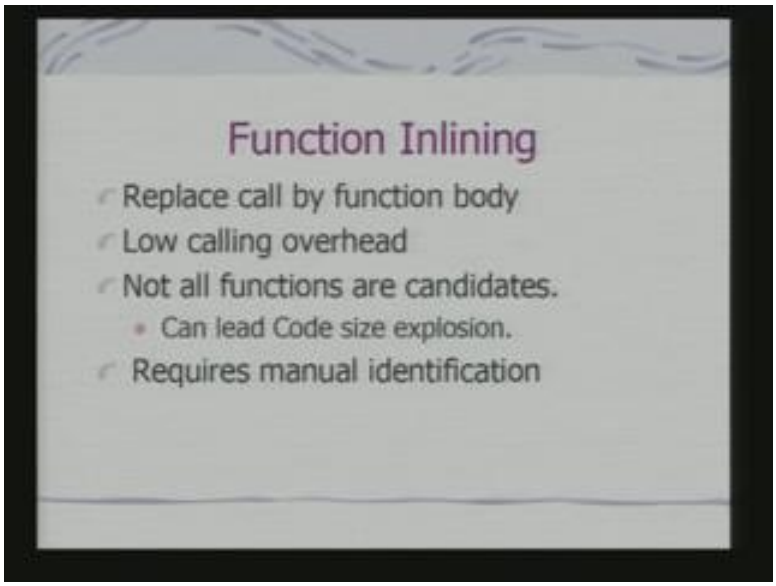
So, what is the solution? Solution is that you pad an element, that is you pad an element a 0 2 here. This is a dummy element; that means, you are changing this data structure itself. Why would you add this dummy element? It is very obvious; that means, when I am doing computation on the first row itself, I shall be doing the computation. Once I finish that and come to the second row, the second row as a whole it is loaded here. Otherwise, what would have happened? I would have got the data over here and then there would be a conflict and I need to come here to do the job.

So, for each row actually would have required accesses, fine. Which would lead to more number of cache conflicts. Now, since I am padding an array element, I am reducing the array access, cache conflicts corresponding to the array access.

So, add array elements to change mapping in to cache and reduce the cache conflicts. Is it clear. So; that means, when I am running a loop you can understand, when I am running a loop this first element. When I come to access this first element actual I might find out that it is in the array. When I come to the next element there would be an array conflict.

Now, if I am not doing anything of the first element, first row. And if I have do an operation of this element, I do a cache load once. When I am going in to the second element that of the first row again I have do a cache load. But if I do this padding I need to do it only once. So, array padding is another array way of array transformation to use an optimal, to have an optimal use of cache.

(Refer Slide Time: 49:09)

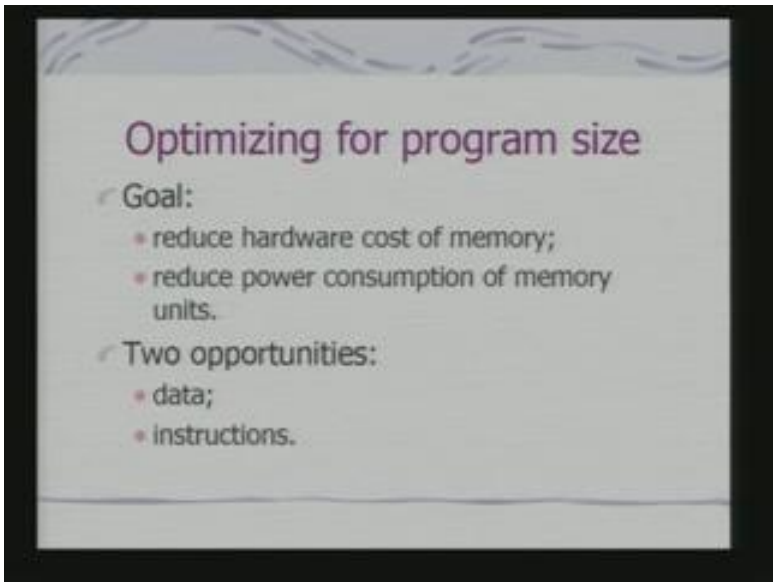


The other point is function inlining. Which is another transformation. So, you replace call by function body. If you have a call in your computation replace it by the function body. Obviously, it is a low calling overhead, the putting in to the registers on to the stack extra are avoided.

But; obviously, you can understand the not all functions are candidates. Because it can lead to code size explosion which you do not want and therefore, it requires a careful manual identification of which functions that can really be put on to the main code. In fact it has been found that if you can do judiciously function inlining. Your computation speed shows definite improvement, because time taken for this overhead is, why? Because you are access in the memory because, your major point is you are accessing the memory to save the register.

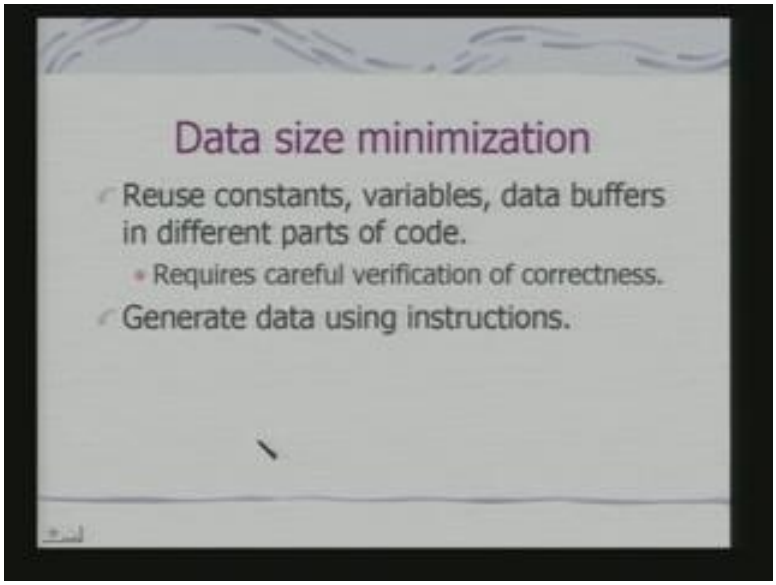
So, if you are avoiding the time taken to save the registers for small functions then, your time required for execution of your code can substantially go down. And that is the basic point or the motivation for looking at function inlining for optimization of computation. Related to this is the problem of optimizing for program size. If you looking at a node for computation, I need to optimize for the program size.

(Refer Slide Time: 50:47)



Now the other point comes in, just try to understand this tradeoffs. In the other previous case also we have looked at tradeoff with regard to the loop. Loop unrolling visa viz loop nesting. We are trying to look at function inlining visa viz optimization of the program. If I am doing function inlining, I cannot do optimization. So, but I need to do an optimization because, I would like to use minimum memory. Because that is also is a constraint for a design.

So, if I optimize the programs size, I reduce hardware cost for memory and reduce power consumption of memory units. Because I shall be using less number of memory units. Obviously, there are two opportunities to look at data as well as that of the instructions.
(Refer Slide Time: 51:34)



So, how do you reduce the data size? Here you use what we call the basic principle of reuse. Reuse constants, variables, data buffers in different parts of the code. Obviously, this violets again a basic software engineering dictate which says, that you use different work variables. Names of variables should be all distinct for all different applications.

But different variable should essentially mean what? Allocation for a space for those variables. If you can reuse variables then the space required reduces. So, variables data buffers you need to reuse different parts of the code. The other strategy which is followed in many of this embedded systems, you generate data using instructions. That means, you get a task node for generating data using instructions.

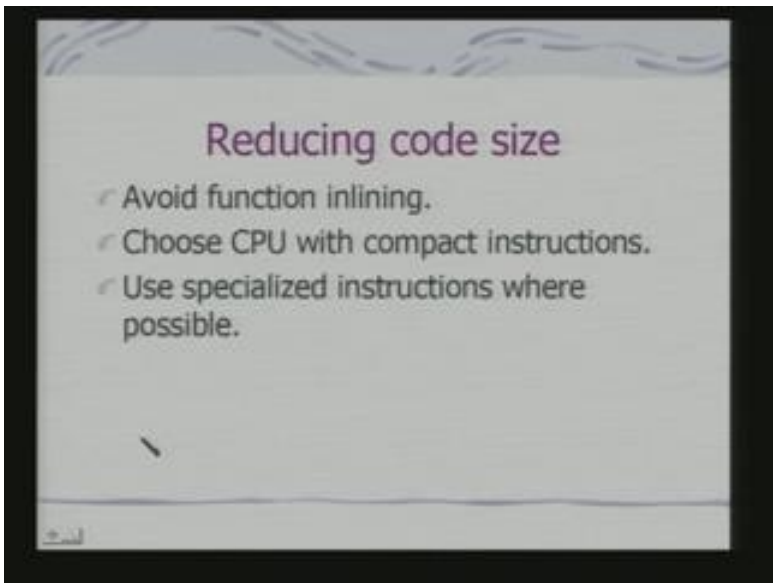
So, if you need some special kinds of instructions for an application, try to model the data using instructions. What is the advantage? The instruction would occupy less memory. Instruction goes in to say for example, flash.

So, the amount of flash memory required would be less, but when it is generating the data you can say that data size would be large fine. If the data size is large we can accommodate the data in a dRAM, but the point is we might not require the entire data for all computations.

So, I need not have the complete data stored. A priory write a program to generate the data depending on requirements. In fact, the very common tradeoff is if you look at a very simple example of a trigonometric function computation. I can use a table driven approach where I use a table and using a table I do a table look up and get the

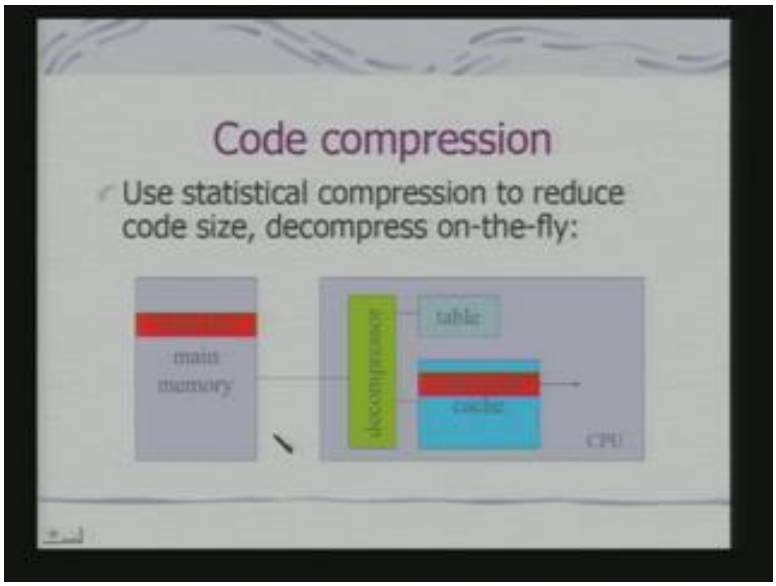
trigonometric value. This will be fast, but; obviously, it will be memory intensive. Because have to store the data in the memory. The other option could be I can use a code, special algorithms to evaluate trigonometric functions. The moment I do that I avoid the memory for storing the data. So, the reducing code size is, the basic strategy here is avoiding function in lining.

(Refer Slide Time: 53:59)



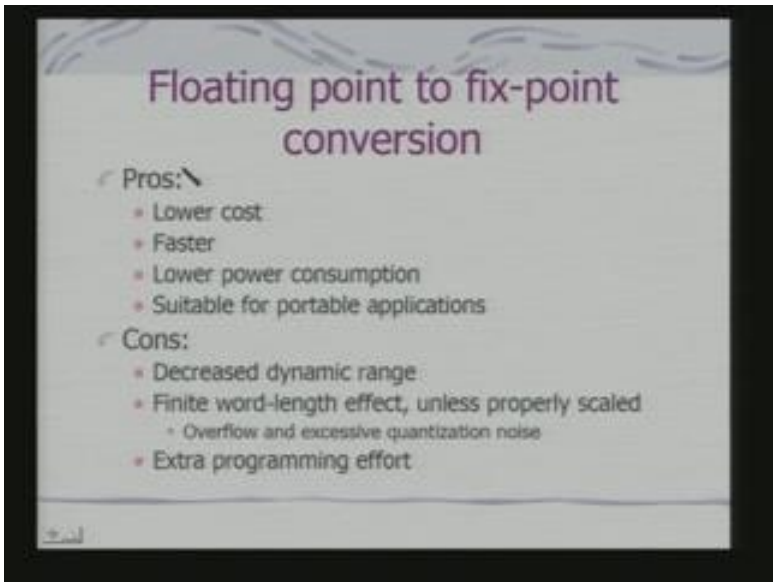
Choose CPU with compact instructions. In fact ARM shows an example and use specialized instruction where ever possible. So, if know that ARM from ARM example, say for we can do a judicious mix of thumb as well as ARM instructions to reduce the memory required for the code. The other approach is also used is called code compression.

(Refer Slide Time: 54:29)



Use statistical compression to reduce code size and decompress codes on-the-fly. So; that means, this is my instruction, this is the coded form of a instruction. So, these instructions from main memory passed on to the CPU. CPU itself can have a decompressor. And after decompression you get the actual instructions which is in the cache. The actual instruction can be a 32 bit but, my representation can be 8 bits. In fact, I can use any form a entropy coding, a variable length coding for instruction compression. In fact, very common example of these if you remember we have discussed use of java that J2ME and for J2ME use archival format of the code. And not the standard format where the binary code is actually compressed. So, that it can be loaded on to a small memory in the embedded system. So, the next part is other is of floating point to fix point conversion

(Refer Slide Time: 55:34)

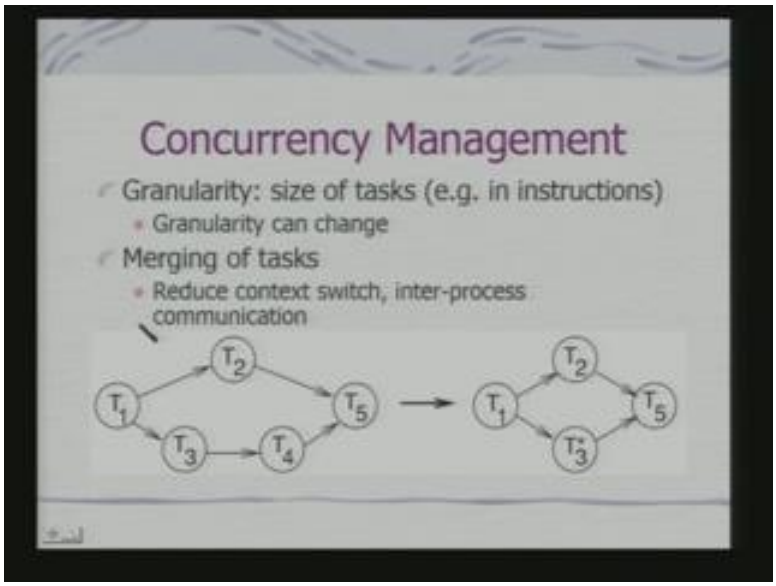


What is the floating point representation? You have a mantissa and you have exponent. Depending on the value of exponent can have variable precision. When you have a fix point that is not there. We have fix number of digits allocated for your fractional part. When you do that what are the advantage. Obviously, you have got lower cost, computational overhead is less in terms of hardware faster, lower power consumption. And that is what is suitable for portable applications. And obviously, the problem is if you have a finite word length effect, unless properly scaled, overflow and excessive quantization noise and it require extra programming effort.

But the fix point conversion, why it is done as part of computation units? If you remember many of this processor to have an support for fix point arithmetic. Even the DSPs have support for fix point arithmetic. And if you have to go for a higher cost DSPs you have to have a floating point support in that for higher cost DSPs.

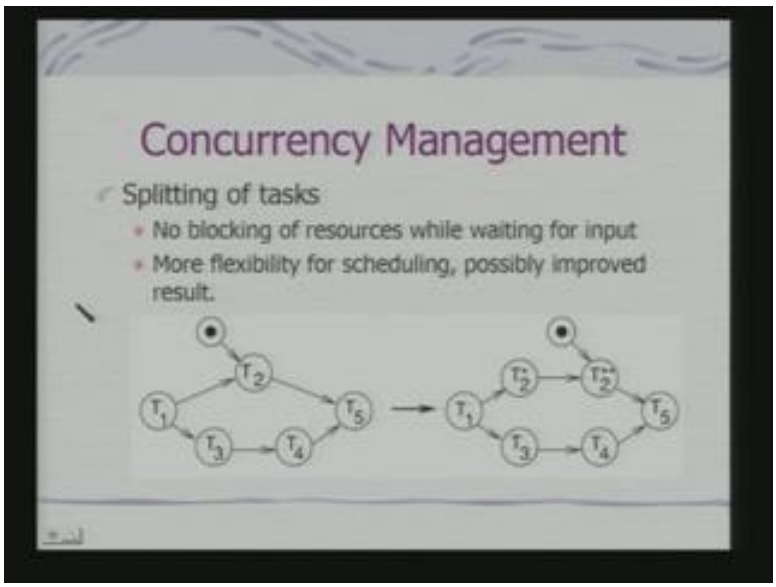
Now, if your precision is good enough to have that you require for an application is good enough to be done with fix point arithmetic, can expected to go for fix point arithmetic. The next thing is which comes up with a concurrency management

(Refer Slide Time: 56:55)



If you have a granularity of the tasks. So, these are the two tasks, we can merge the tasks. Now, we are again coming back and looking at a task level graph.

(Refer Slide Time: 57:06)

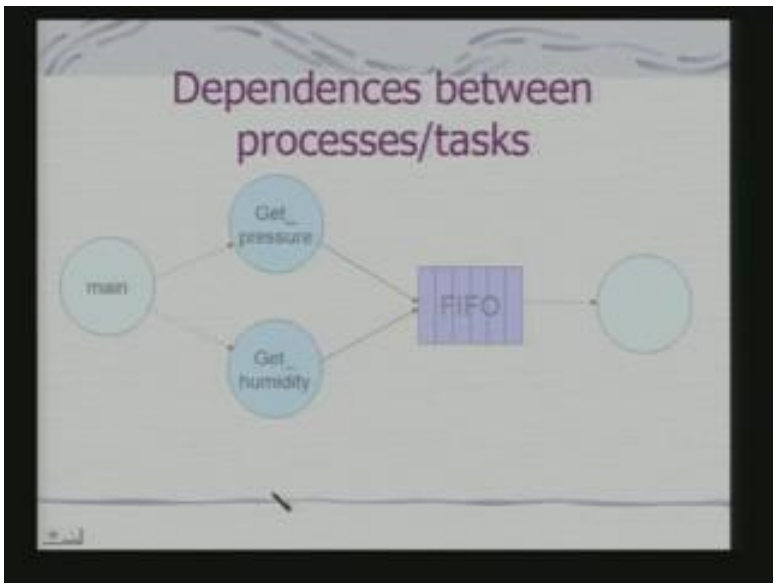


Again we can have a splitting of tasks, if I split the tasks, this is task T 2 was waiting for the input. But there we have isolated out one part of the task which does not depend on input. By doing it we have no blocking of resources while waiting for input. We have got more flexibility for scheduling and improved results

(Refer Slide Time: 57:27)

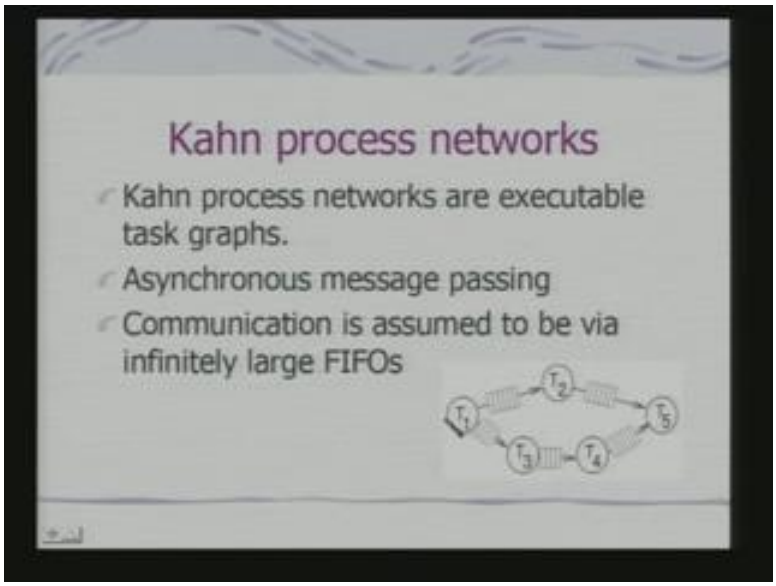
There are other representation tools, we have primarily look the task graph. The other representation tools which have been used. Process networks which is in this kind of a thing. We have the pressure sensor, humidity sensor. It is a multiplexer goes to the FIFO that is, the data representation in a set of communicating processes. In fact, task graph is a more detail representation of process networks in a way.

(Refer Slide Time: 57:53)



So, what is important? You get various dependencies indicated. So, this another representation tool.

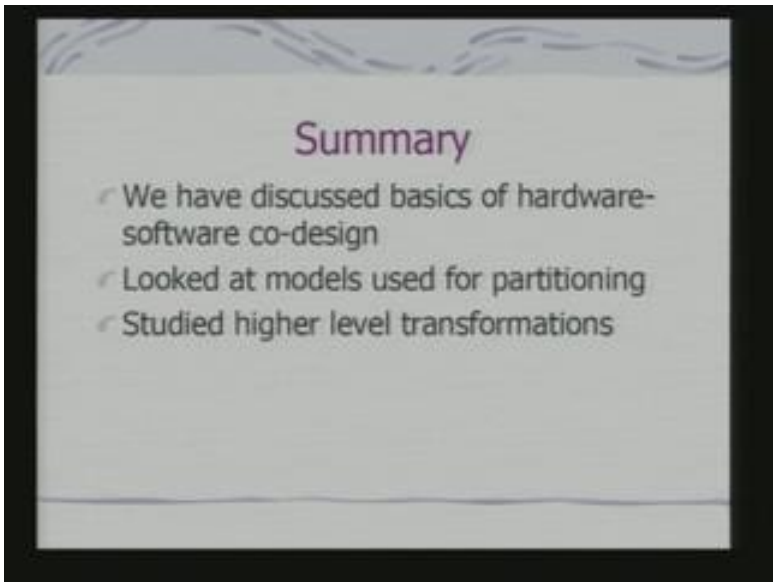
(Refer Slide Time: 58:03)



Associated with this is called Kahn process networks. In a Kahn process networks are executable task graphs and you have got asynchronous message passing here. And communication is assumed to be via infinitely large FIFOs. So, that means, the synchronization is taking care of to this FIFOs. These are all abstract models which are used. In fact, Kahn process network in an abstract model, which is also use for partitioning tasks.

But we have concentrated in a today's discussion primarily on the task graph and transformations with respect to the task graphs. Process networks and Kahn process networks are other alternative models.

(Refer Slide Time: 58:40)



So, today what we have discussed is basics of hardware and software design. Looked at models used for partitioning. And studied higher level transformations, which can be applied through the nodes. Further details we shall explore in the next class.