Compiler Design

Prof. Y. N. Srikant

Department of Computer Science and Automation

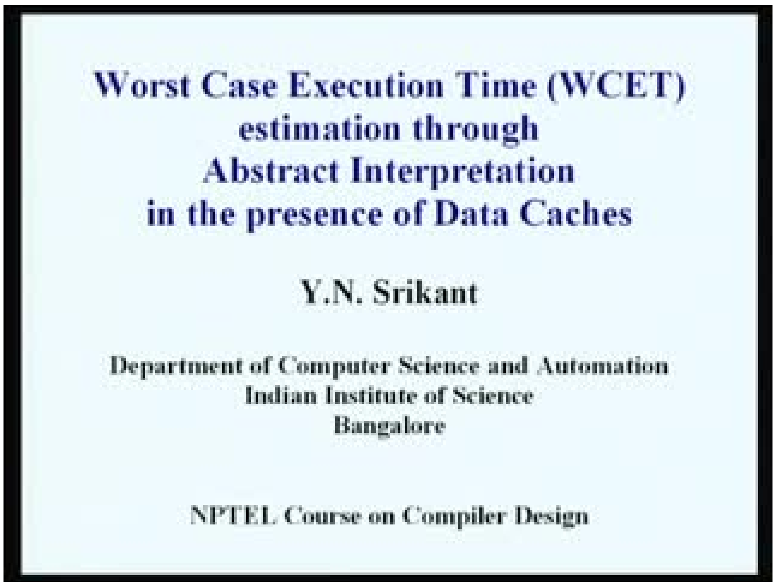
Indian Institute of Science, Bangalore

Module No. # 21

Lecture No. # 40

Worst Case Execution Time-Part 2

(Refer Slide Time: 00:18)



Worst Case Execution Time (WCET) estimation through Abstract Interpretation in the presence of Data Caches

Y.N. Srikant

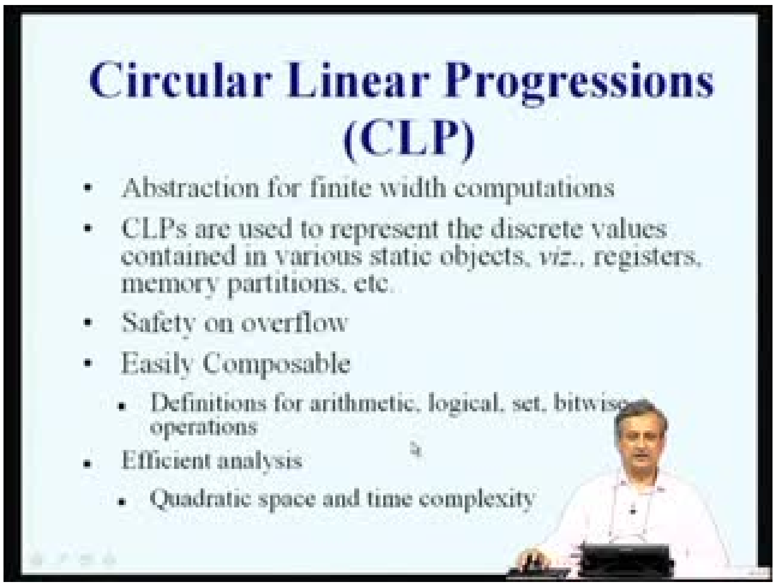
Department of Computer Science and Automation
Indian Institute of Science
Bangalore

NPTEL Course on Compiler Design

Welcome to part 2 of the lecture on Worst Case Execution Time estimation through Abstract Interpretation in the presence of Data Caches. Last time we discussed why WCET estimation is important and then went through some of the abstract interpretation fundamentals.

Today we are going continue discussion of the representation of various abstract objects and things like that and continue with WCET as well.

(Refer Slide Time: 00:55)



Circular Linear Progressions (CLP)

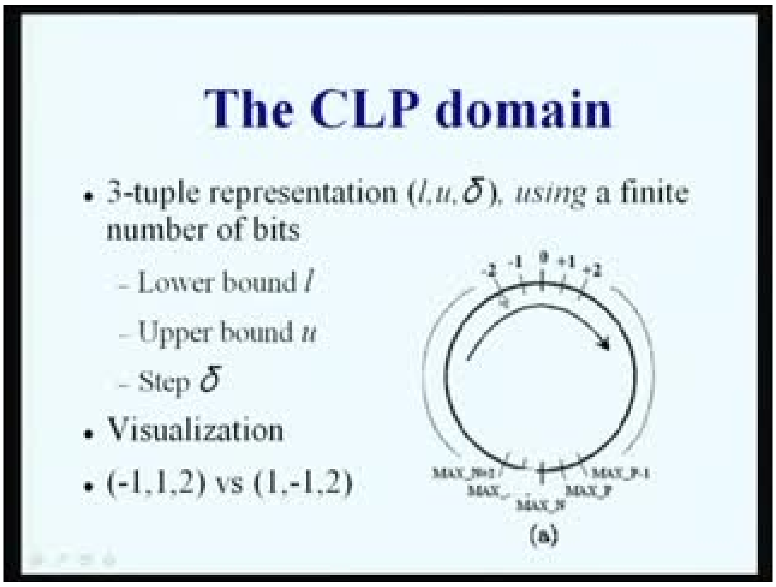
- Abstraction for finite width computations
- CLPs are used to represent the discrete values contained in various static objects, *viz.*, registers, memory partitions, etc.
- Safety on overflow
- Easily Composable
  - Definitions for arithmetic, logical, set, bitwise operations
- Efficient analysis
  - Quadratic space and time complexity

Circular linear progressions are one way of representing finite width computations; for example, when we create abstractions of these partitions or the computations which happens in the registers memory locations etcetera.

The sets of discreet values, which are actually encountered will have to be represented concisely and so that a precision also maintained. So, CLPs are one type of such abstractions.

They provide safety and overflow they are easily composable definitions for arithmetic logical set bitwise operations etcetera, are all provided and efficient analysis is possible.

(Refer Slide Time: 01:54)



The CLP domain

- 3-tuple representation (l, u, δ), *using* a finite number of bits
  - Lower bound l
  - Upper bound u
  - Step δ
- Visualization
- (-1,1,2) vs (1,-1,2)

-2
-1
0
+1
+2
MAX_N+1
MAX_
MAX_N
MAX_P
MAX_P-1
(a)

So, to visualize the CLP domain it can be visualized as a circle and the elements of this CLP domain can be represented along the periphery of this circle as shown here. So, we always traverse in the clockwise direction.

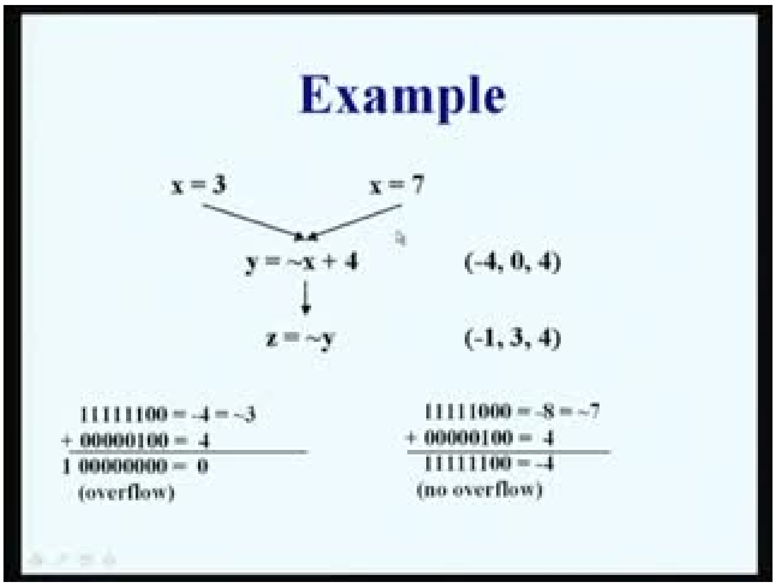
The CLP domain has the elements have three parts: one is l, the second is u and third is delta. So l corresponds to the lower bond of the set of elements that the CLP domain represents and u represents - u is the upper bond and the step is delta.

For example, if we take minus 1, 1 and 2; so you are really starting at minus 1, you are really ending at plus 1 and then step is 2; so, that means, in 1 step you go from minus 1 to 1 so that means you really represent only two elements.

Whereas, 1 minus 1 and 2, you start from 1 and in steps of 2 go to minus 1; so, there are a large number of elements in this domain so in this set.

That is the difference between the set and this CLP representation. In a set minus 1 comma 1 and 1 coma minus 1 are equivalent. Whereas, in CLP they are not equivalent the elements are all going to be three tuples and the semantics is as i just now described.

(Refer Slide Time: 03:30)



Example

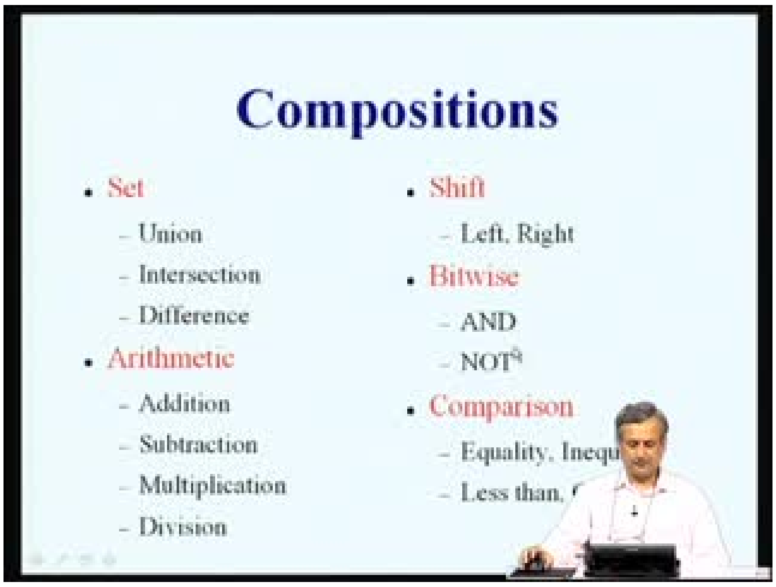
$$\begin{tikzcd}
x = 3 \arrow[dr] & & x = 7 \arrow[dl] \\
& y = \sim x + 4 \arrow[d] & & (-4, 0, 4) \\
& z = \sim y & & (-1, 3, 4)
\end{tikzcd}$$

```
  11111100 = -4 = ~3          11111000 = -8 = ~7
+ 00000100 =  4             + 00000100 =  4
1 00000000 =  0               11111100 = -4
  (overflow)                 (no overflow)
```

Here is the example that we discussed last time - x equal to 3 causes overflow in y equal to complement x plus 4; whereas, x equal 7 does not cause any overflow. And in both cases in the case of overflow and no overflow the CLP represents the values properly. Here the value is 0; so you so really have minus 4 and 0 are the 2 values.

Here we have minus 4, 0 and 4, really gives you 2 values from minus 4 to 0. So, in both in the minus 4 corresponds to no overflow case and 0 corresponds to the overflow case. Similarly, the value of z is also appropriately represented as minus 1 comma 2 comma 4.

(Refer Slide Time: 04:29)

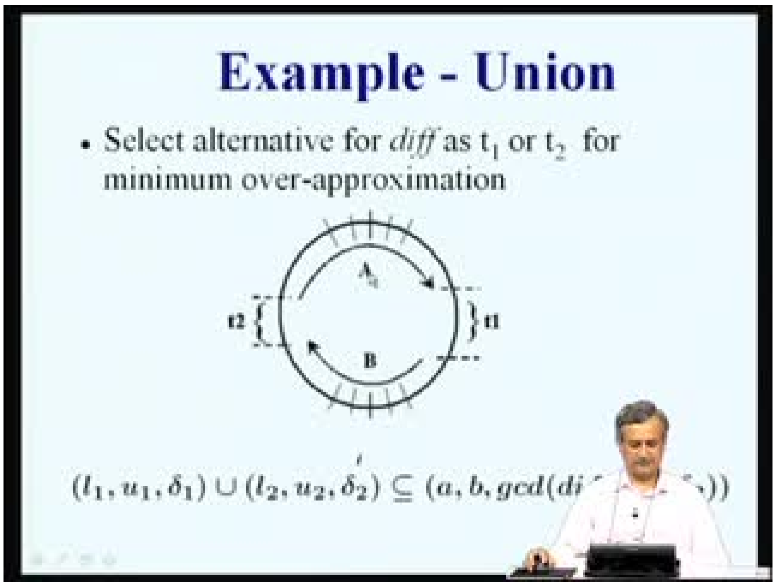


Compositions

- Set
  - Union
  - Intersection
  - Difference
- Arithmetic
  - Addition
  - Subtraction
  - Multiplication
  - Division
- Shift
  - Left, Right
- Bitwise
  - AND
  - NOT
- Comparison
  - Equality, Inequ
  - Less than,

Now let us look at the operations so there are set operations such as union intersection and difference possible on the CLPs. Arithmetic operations such as addition subtraction multiplication division, then shift left and right shifts are possible then bitwise and not operations are possible, comparisons of equality inequality less than greater than are all possible. I will give you one example of a how exactly union is a carried out.

(Refer Slide Time: 05:00)



Example - Union

- Select alternative for *diff* as t_1 or t_2 for minimum over-approximation

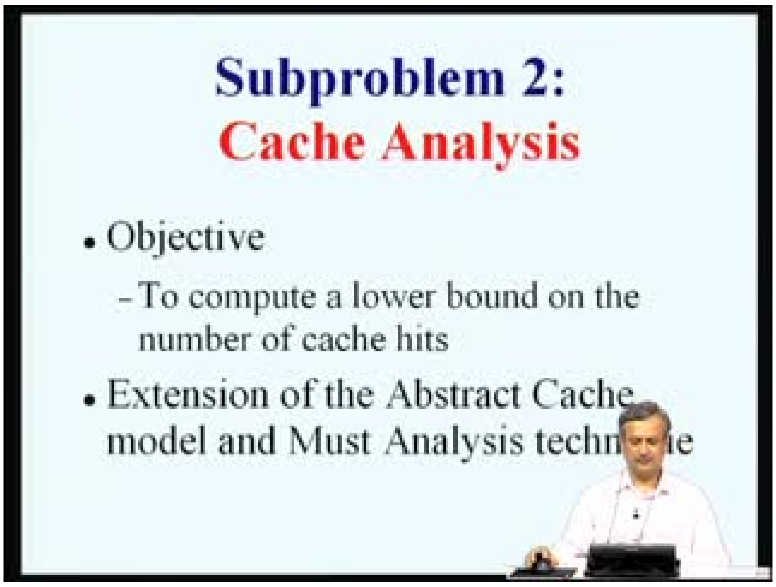
$(l_1, u_1, \delta_1) \cup (l_2, u_2, \delta_2) \subseteq (a, b, gcd(di$

So, let us say there are two CLPs A and B. So, A starts here and ends here, B starts here and ends here. So, if we consider the union of these two that CLPs A and B. There are two possibilities obviously, we cannot maintain the discrete nature of the two CLPs this particular gap in between here and here cannot be maintained as it is. We have a possibility of starting from this point, go all the way include this gap as well and then go all the way up to this B.

You will have A union B along with some more elements here. The other possibility is you start here, go up to this point then include the elements in this gap as well and then go all the way up to this point. So, here you are going to include these extra elements into the union A and B of course, all the elements will be included, but here you are going to include a few more elements.

In such a case, we want to minimize the over approximation in other words the number of elements which are included here or here whichever is smaller is chosen for inclusion.

(Refer Slide Time: 06:38)



Subproblem 2: Cache Analysis

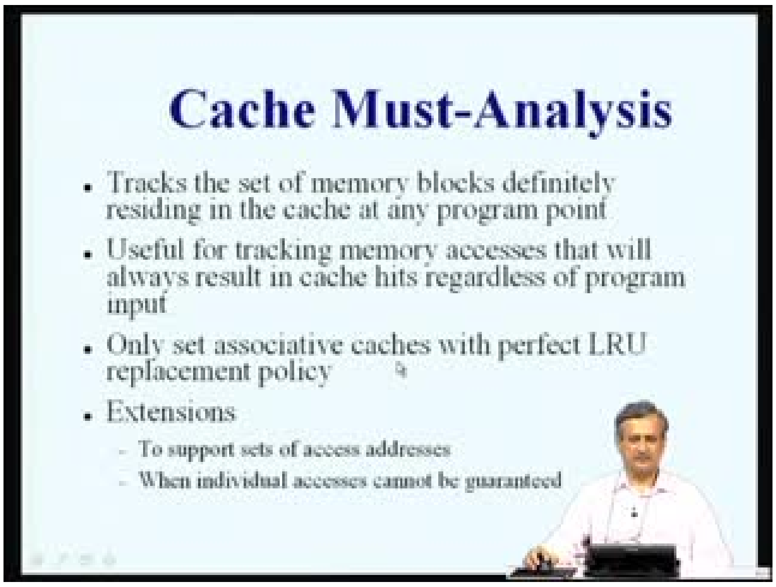
- Objective
  - To compute a lower bound on the number of cache hits
- Extension of the Abstract Cache model and Must Analysis techn ie

If we choose t1 for inclusion then we start from here and go all the way up to this point; whereas, if we choose t2 for inclusion we start from here and go all the way up to this point. So, that is how we represent unions so similar representations are possible for other operations as well. So, that was about representation of the various partitions and computations using CLPs which are a very efficient representation.

Now the second sub phase of this WCET analysis is the Cache Analysis. So the objective of cache analysis is to a compute lower bond on the number cache hits. So, we do not simulate cache here, we are actually going to make a model of cache an mathematical model and then update the cache appropriately whenever we have a memory access.

This model of cache and analysis of cache is an extension of the abstract cache model and what is known as a must analysis technique for caches, which has been there in literature for some time.

(Refer Slide Time: 07:41)


Cache Must-Analysis

- Tracks the set of memory blocks definitely residing in the cache at any program point
- Useful for tracking memory accesses that will always result in cache hits regardless of program input
- Only set associative caches with perfect LRU replacement policy
- Extensions
  - To support sets of access addresses
  - When individual accesses cannot be guaranteed

We have actually extended the available abstract cache model. So, what exactly is cache must analysis? So, a must analysis tracks the set of cache must analysis tracks the set of memory blocks definitely residing in the cache at any program point.

Because it says must that means we must guarantee something so what are guaranteeing, we are guaranteeing the set of memory blocks definitely residing in the cache at any program point. This is useful top for tracking memory accesses that will always result in cache hits regardless of program input. So, that is the point so it gives you some kind of a lower bound on the hits there may be other hits as well, but we cannot guarantee them that is what this cache analysis says.

Based on this we would actually tag some of the memory accesses in the binary executable, as cache hits and some other as non-cache hits or cache misses and appropriately calculate the access time those memory references. So that is the idea of cache analysis.

For each instruction there will be memory references. We must determine whether that memory reference results in a memory cache hit or a cache miss so that is done by our cache model.

So, only set associative caches with perfect LRU replacement policy are modeled here. Others are not so easy to model and LRU of course, captures the essence of available replacement policies and it is probably widely used.

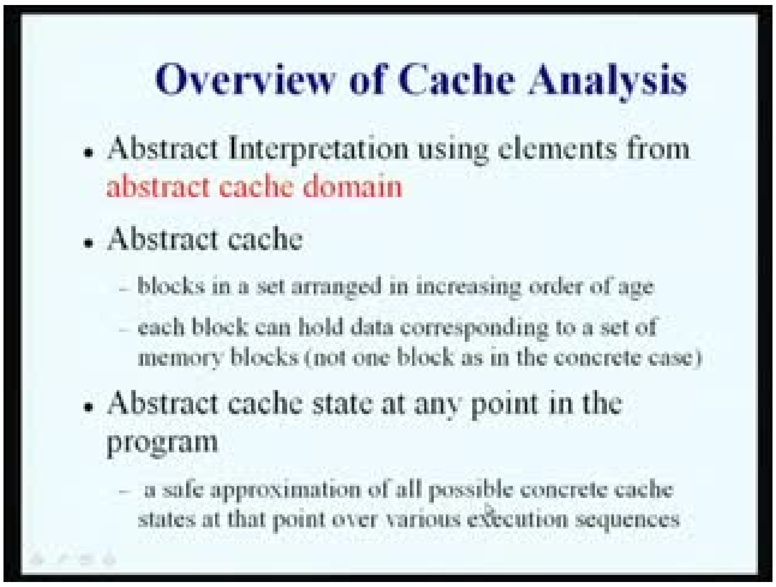
There are extensions that we have made, one is to support sets of access addresses and when individual accesses cannot be guaranteed, in other words suppose we are accessing some array element let in a particular partition. We do not know which element of the array will be accessed the reason is the entire partition is going to be tracked as one unit so any element of this particular partition can be accessed during that memory that reference. Otherwise we would have already divided that reference in two or more parts.

So, all the elements in a particular partition can be accessed with that reference. So there are really going to be many elements possible so, that is what is represented by a CLP.

The previous model of cache supported only one singleton address and we support now multiple support sets of access multiple addresses.

We cannot really guarantee individual accesses here, because no one particular access can be guaranteed every one of them is equal probable.

(Refer Slide Time: 10:40)



Overview of Cache Analysis

- Abstract Interpretation using elements from abstract cache domain
- Abstract cache
  - blocks in a set arranged in increasing order of age
  - each block can hold data corresponding to a set of memory blocks (not one block as in the concrete case)
- Abstract cache state at any point in the program
  - a safe approximation of all possible concrete cache states at that point over various execution sequences

It act the cache analysis happens as abstract interpretation using elements from an abstract cache domain. So what exactly is an abstract cache?

So, abstract cache consists of blocks in set arranged increasing order of age. So obviously, we have to model age as well because LRU requires this age information for replacement policy.

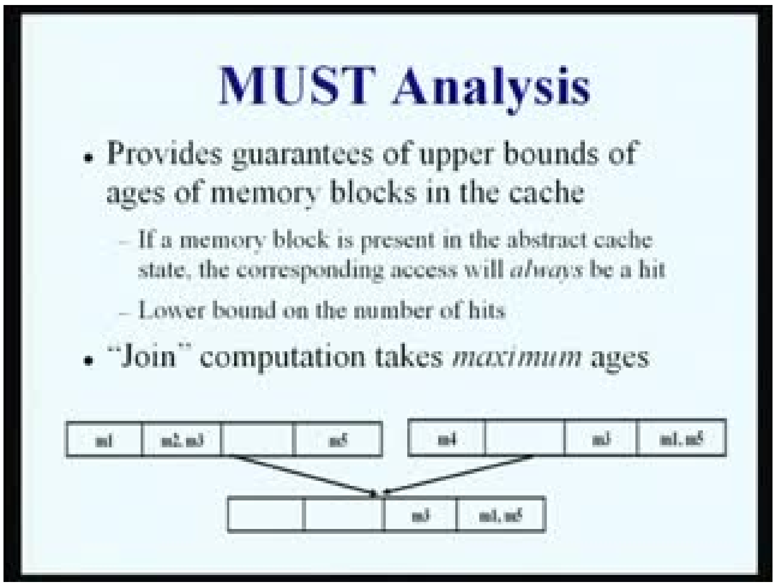
Since we are modeling age as since we are modeling LRU policy we must also model the age. So there are going to be many blocks and depending on how much of history we want to maintain the number of blocks will be as many.

In a cell the blocks in a set arranged in the increasing order of age and each block can hold data corresponding to a set of memory blocks so, not one block as in the concrete cache case.

In a concrete cache at any point in time only each block can hold only one memory block. Whereas, in an abstract cache each block can hold many memory blocks so, that why it is an abstract cache, just like in abstract domain of integers and signs etcetera.

What exactly is an abstract cache state at any point in the program? So, a safe approximation of all possible concrete cache states at the point over various execution sequences. This is a safe approximation and we will have to see how to make this safe approximation. So, at any point in time as abstract cache has to represent many states so that is the requirement. And even in the above point, when we say many blocks correspond to a single block of abstract cache; it does not mean all those will be present physically, one of them will be present, but which one we do not know so, we have track all of them.

(Refer Slide Time: 13:08)



MUST Analysis

- Provides guarantees of upper bounds of ages of memory blocks in the cache
  - If a memory block is present in the abstract cache state, the corresponding access will *always* be a hit
  - Lower bound on the number of hits
- "Join" computation takes *maximum* ages

m1
m2, m3
m5
m4
m3
m1, m5
m3
m1, m5

So, what exactly is must analysis? So it provides guarantees of upper bounds of ages of memory blocks in the cache. So if a memory block is present in the abstract cache state the corresponding access will always be a hit.

So, you are guaranteed to have a hit if the memory block is present in the abstract cache state. But if the memory block is not present, you cannot say that it will always be miss in the physical world. it could be a miss; it could be a hit, but since it is a must analysis and we are not able to guarantee that it will be a hit, we will say that it is a miss.

So, it provides a lower bound on the number on hits so that is important it does not give you the exact number of hits, it does not give you an upper bond on the number of hits but, it gives you a lower bound on the number on hits.

That is because of the above statement. See if a block is present in the abstract cache then we guarantee a hit, but if it is not present it can be a hit or it can even be a miss.

Because of this you only the guarantee a lower bound on the number of hits. So this information as I said is used to compute the execution time of various instructions.

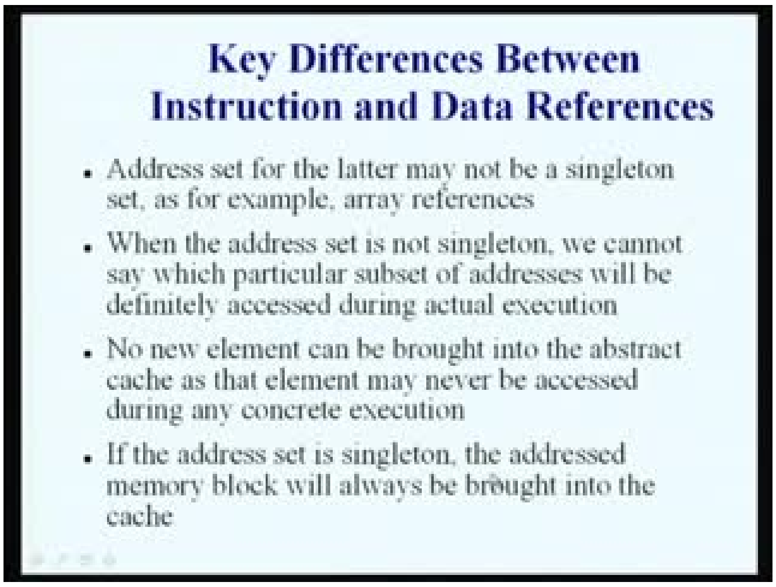
So, the control flow graph is going to have a join statement so, what is a joint computation? It really takes the maximum ages. So, let us consider these two cache states, one coming from this direction and the other coming from this direction.

So, this has m1 as the most recently used m2, m3 as the next recently used and m5 as the last least used. So, here we have m4 and then we have m3 and m1, m5.

When we take a join of these two we always take the maximum ages so look at m1 so, m1 is present here in the last one so, we are actually going that use that as the age of m1 in the join. Then m2 is not present in the second one so we are not going to consider it at all, both of them must be having the same block otherwise we do not consider it.

m3 is here m3 is here so m3 is going to be present here, m5 is here and m5 is here so m5 is going to be present here so this is the cache state corresponding to join.

(Refer Slide Time: 15:47)



Key Differences Between Instruction and Data References

- Address set for the latter may not be a singleton set, as for example, array references
- When the address set is not singleton, we cannot say which particular subset of addresses will be definitely accessed during actual execution
- No new element can be brought into the abstract cache as that element may never be accessed during any concrete execution
- If the address set is singleton, the addressed memory block will always be brought into the cache

What are the key different between instruction data references? So, address set for the latter that is the data references may not be a singleton set as for example, array references.

Whereas in the instruction reference it is going to be exactly a singleton the instruction will be present in exactly one place, one address not in many possible addresses so, any particular instruction will have a fixed address so, and it is a singleton reference.

Whereas, for data as we have seen there will be a partition and if it is an array reference we do not know, which element of the array will be accessed until runtime? So, because it could be accessed with a variable such as I whose value is computed at time.

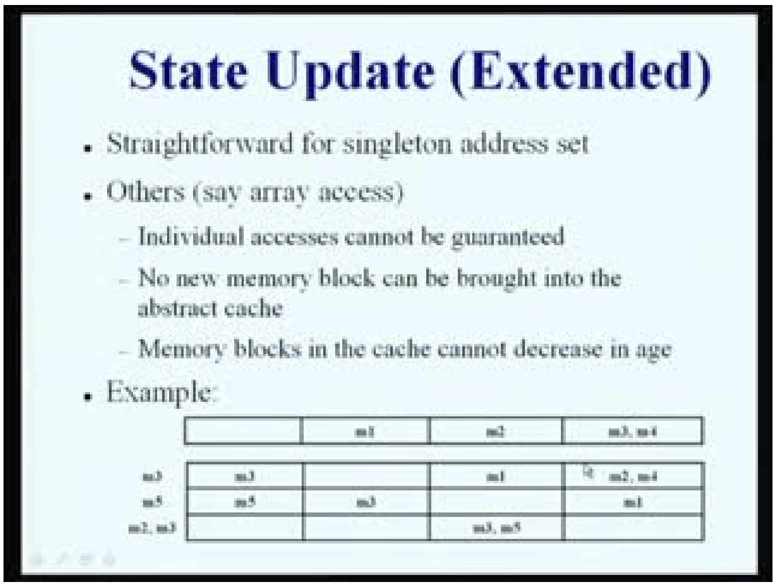
We will really have a set of possibilities corresponding to that partition any address in that partition is possible so that is a CLP.

When the address set is not singleton. We cannot say which particular subset of addresses will be definitely accessed during actual execution any one of them this possible or more of them one subset is possible.

No new element can be brought into the abstract cache as that element may never be accessed during any concrete execution. In such a case when there is a set of possibilities bringing if one of them not presenting in the cache, bringing it into the cache may be wrong because that particular element never be accessed during run time. So we are never going bring any new element into the cache when it is set of references. If the address set is a singleton the address memory block will always be brought into the cache.

If it is singleton reference and it is not in the cache block we are going bringing into the cache block. That is not an issue only when there is a set of addresses we cannot bring the element which is not present a new element in to the cache.

(Refer Slide Time: 18:03)



State Update (Extended)

- Straightforward for singleton address set
- Others (say array access)
  - Individual accesses cannot be guaranteed
  - No new memory block can be brought into the abstract cache
  - Memory blocks in the cache cannot decrease in age
- Example:

	m1	m2	m3, m4

m3	m3		m1	m2, m4
m5	m5	m3		m1
m2, m3			m3, m5	

What exactly is cache update? This is the extension, straightforward for singleton address set so that is very easy so we will see in this example what exactly happens. So here we have a cache state with m1, m2 and m3, m4. Now there is an access for m3; m3 is already present here so this is the least recently used and this is the most recently use so we are bring m3 to the recently used block.

And then m1 actually increases in age by 1 so it comes to this block, m2 increases in age by 1 so it goes to the last block. Each one of these blocks can hold a maximum of two entries so we have put m2 also into this slot.

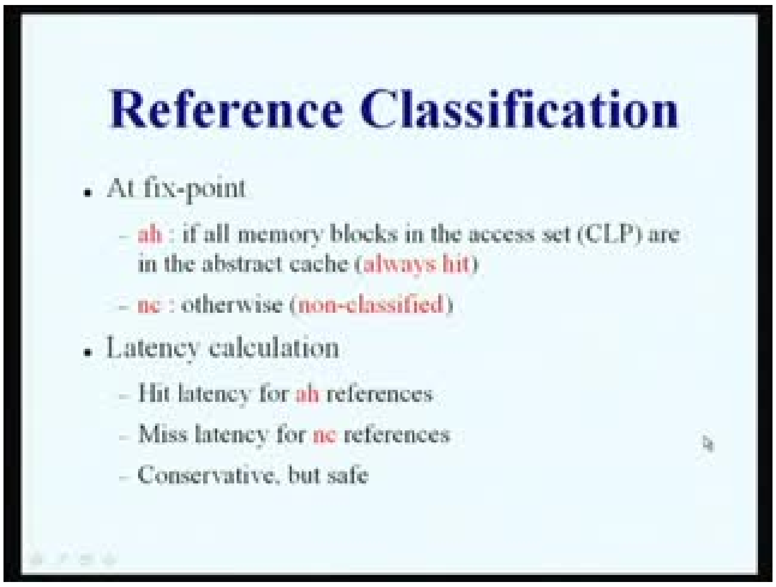
Similarly m5, m5 a singleton. So we have m3, m1, m2, m4, m5 is a new element but, it is a singleton so it is brought into the most recently accessed block. So m5 is going to here, m3 will be shifted by 1 so it comes here, m1 will be shifted into this block since this is already full the elements here will be thrown out and m1 occupies that particular slot.

But when there is an array access any number of elements are possible so individual accesses cannot be guaranteed. No new memory block can be brought into the abstract cache and memory blocks in the cache cannot decrease in age.

This update function is actually quite complex and it is difficult to explain in this short lecture. So m2, m3 is a set of addresses here so what we really compute is the maximum age that has to be provided by a suitable update function so it so happens that the function computes m2 is a new element which is not presents in this cache before so it cannot bought into the cache. Whereas, m3 is an element which is already present in the cache, but neither of them guaranteed to actually happen, neither m2 nor m3 actually may be accessed at runtime or any one of them maybe access.

So, m3 is actually a the maximum age that is possible would be according to that formula that as i said is complex will shift it to this particular slot and m5 will also be shifted into this slot m1 will go out. This is the cache update that can happen for a set of addresses.

(Refer Slide Time: 20:52)



Reference Classification

- At fix-point
  - ah : if all memory blocks in the access set (CLP) are in the abstract cache (always hit)
  - nc : otherwise (non-classified)
- Latency calculation
  - Hit latency for ah references
  - Miss latency for nc references
  - Conservative, but safe

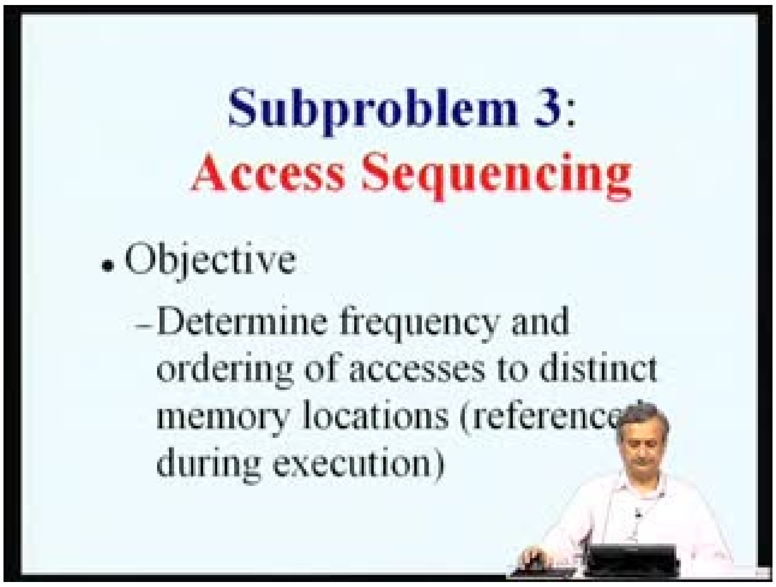
We are going to classify cache accesses into two categories. So at a fix point we are going to see what this means. We are going to have an iteration over the statement so that the fix point is reached just like data flow analysis.

So, ah if all memory blocks in the access set or in the abstract cache, this is always it. So CLP corresponds to a set of addresses and if all those addresses are guaranteed to be in the abstract cache so of course, CPL may have just a singleton address as well or it can have more than one.

If all of them are guaranteed to be in the abstract cache then we say it is always a hit otherwise we say non-classified so that corresponds to a miss. So latency calculation will be hit latency for ah references and miss latency for nc references this is conservative but safe.

That is why we really get the lower bound on the number of hits so our actual number hits maybe more than that. So that is why our w cet value will be somewhat more than the actual w cet value.

(Refer Slide Time: 22:09)



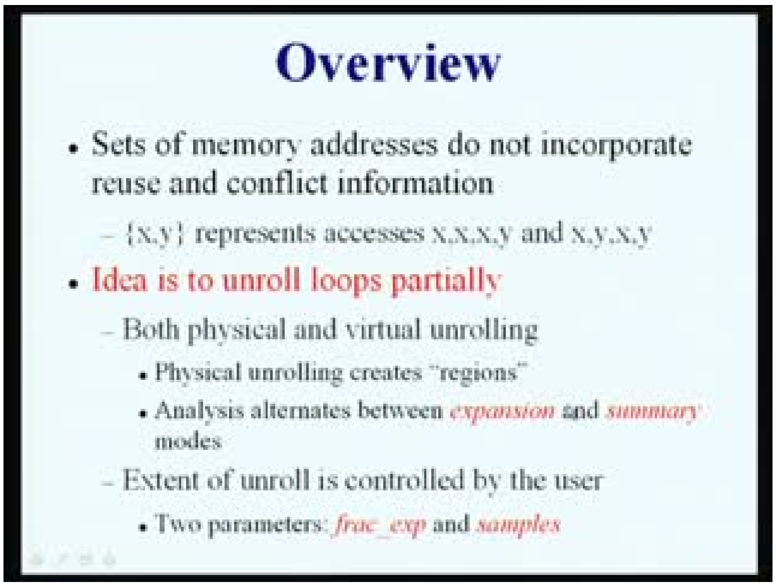
Subproblem 3: Access Sequencing

- Objective
  - Determine frequency and ordering of accesses to distinct memory locations (references) during execution)

After classifying the various instructions as ah or nc we know how much time is needed for each instructions you know how to the compute the situation time of instructions.

Now we need to wary about access sequencing in the program. So, each instruction we have already taken care of now to compute actually to this access sequencing and that calcification of reference happens together. So we need to see how go through the program, navigate through the program and compute those references or rather calcified those references. So, determine frequency and ordering of access to distinct memory location reference during execution so then we will know where its hit or a miss on the cache model and we can compute the execution time of the instruction.

(Refer Slide Time: 23:09)



Overview

- Sets of memory addresses do not incorporate reuse and conflict information
  - {x,y} represents accesses x,x,x,y and x,y,x,y
- Idea is to unroll loops partially
  - Both physical and virtual unrolling
    - Physical unrolling creates "regions"
    - Analysis alternates between *expansion* and *summary* modes
  - Extent of unroll is controlled by the user
    - Two parameters: *frac_exp* and *samples*

Here is an over view sets of memory address do not incorporate reuse and conflict information so simple set of memory address will not do. For example, altering of address is important.

If we say x y are two access. They possibly there would be four access x, x, x and y in that order or there are going to be four access in this order x, y, x, y. So in the first case when x is access let us say the x is not in the cache it will be brought in to the cache then the other two x's are going to be hits and then y is going to be a miss.

Whereas in this case first time x is brought in to the cache, it was a miss so it was brought in to the cache. But then y displace it and then x again displace y and y displace x again so all four of them would be misses assuming that x and y map to the same map.

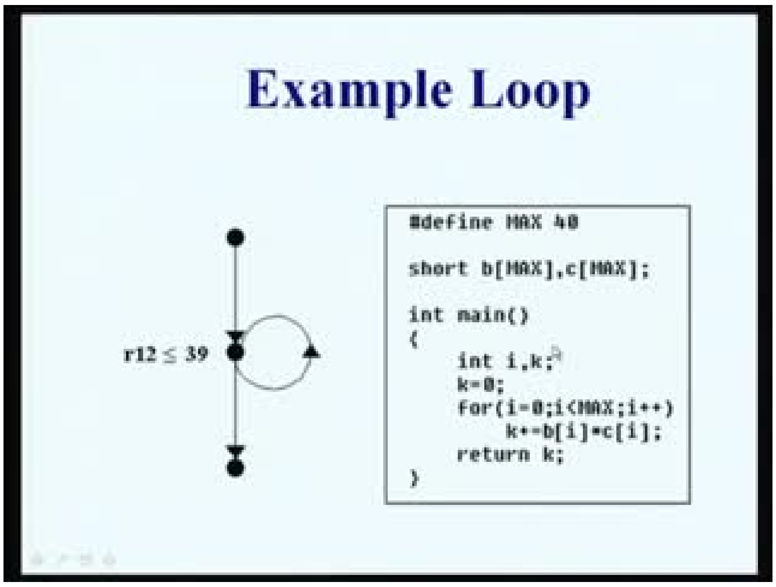
So, representing the access as a set will not help, we need to have a some kind of a sequencing mechanism. The idea is to unroll loops partially so we use both physically and virtual unrolling.

Physical unrolling is basically to create regions so, we know what is unrolling of loops so the body of the loop actually including inside the body and the number of hydration will reduce the appropriately that is unrolling of the loop so, this is physical unrolling.

The virtual unrolling implice analysis goes through the loop more than once, but there is no physical unrolling of the loop itself.

So, analysis alternates between expansion mode and summary mode this will become clear very soon. And extend of unroll is control by the user two parameters faction of expand and samples will control the amount of the unroll.

(Refer Slide Time: 25:25)



Example Loop

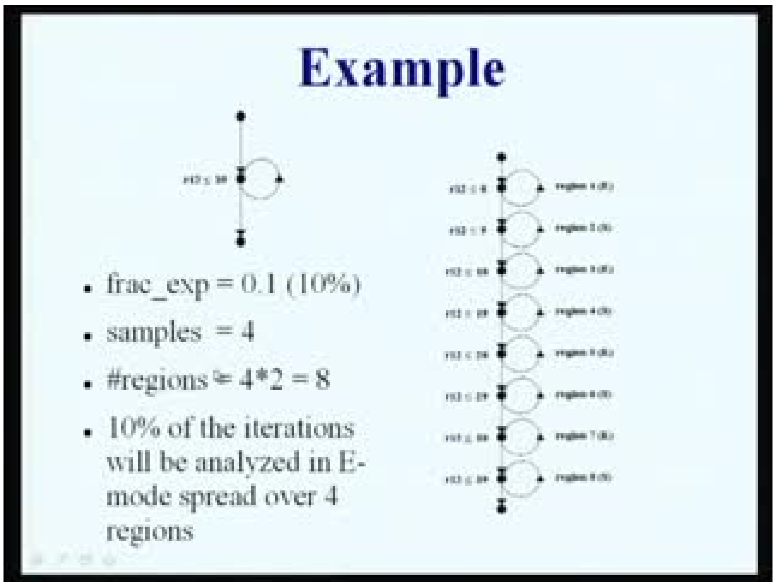
```
#define MAX 40

short b[MAX],c[MAX];

int main()
{
    int i,k;
    k=0;
    for(i=0;i<MAX;i++)
        k+=b[i]*c[i];
    return k;
}
```

Let me give you an example. So here is a program this max is defined as 40 and then there is a variable b, another variable c and this main has int i, k as local variables, k is set to 0 then we have a for loop which goes from 0 to max minus 1. So there is some computation inside and then return k.

(Refer Slide Time: 26:00)



Example

- frac_exp = 0.1 (10%)
- samples = 4
- #regions = 4*2 = 8
- 10% of the iterations will be analyzed in E-mode spread over 4 regions

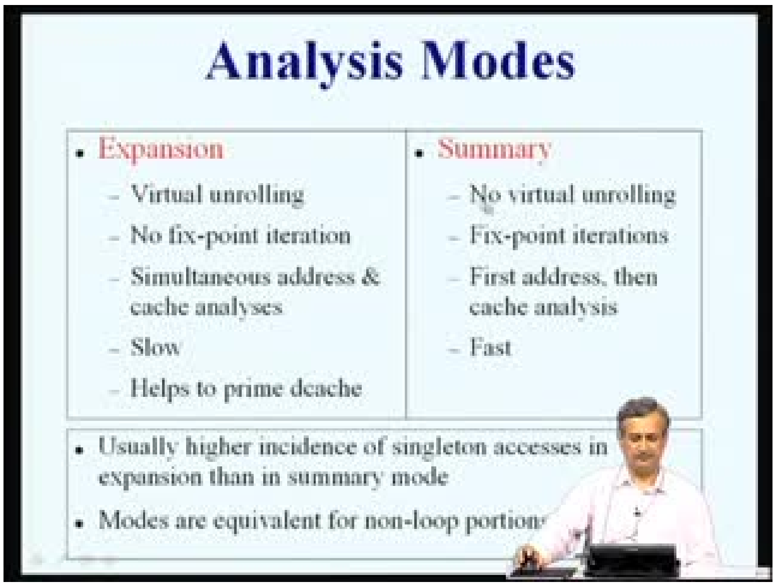
The structure of the sing is there are these are the blocks, nodes in the control flow graphs. Then there is a loop here and let us say the loop counter is r12 it is less than equal 39. This is the unrolled version of the loop. Let us say frac_exp is 0.1 so that is 10 percent so samples is 4 so then the number of regions is computed as samples in to this 2 that is 8.

So, the implication is 10 percent of the iterations will be analyzed in expansion mode spared over 4 regions. So, the 10 percent of iterations emplace there are 48 rations so really 4 so these 4 will be analyzed in expansion mode. What exactly is expansion mode? We will see a soon.

And they are spread over 4 regions so there are 8 regions here. Four of these reason have been marked as E, the other four marked as S so E and S they are alternating. S is the summery mode and E is the expansion mode.

So, as the name may suggest summery mode will do less accurate analysis and expansion mode will do more accurate analysis and there is a definite purpose behind such analysis.

(Refer Slide Time: 27:15)



Analysis Modes

• Expansion – Virtual unrolling – No fix-point iteration – Simultaneous address & cache analyses – Slow – Helps to prime dcache	• Summary – No virtual unrolling – Fix-point iterations – First address, then cache analysis – Fast

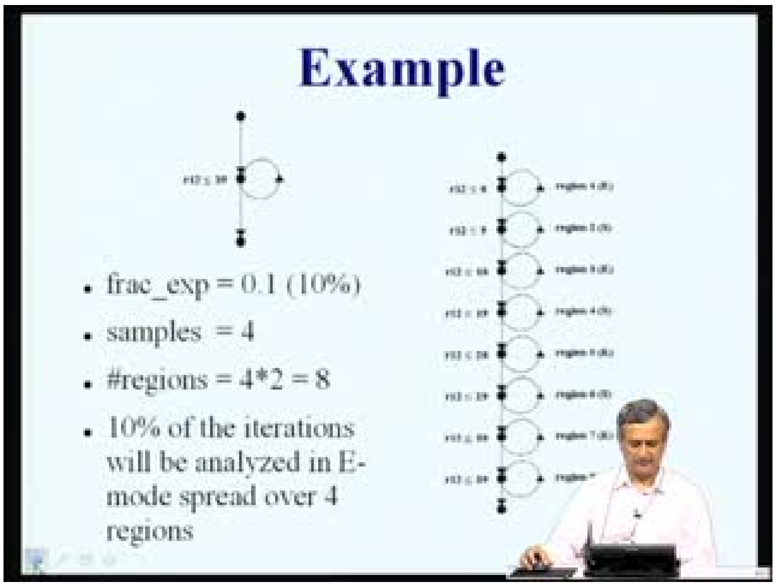
- Usually higher incidence of singleton accesses in expansion than in summary mode
- Modes are equivalent for non-loop portions

The expansion mode actually this is virtual unrolling so in other words we are not going to physically unrolled again, but we go through the loop in a very systematic and thorough manner. There is no fix point iterations in other words we go through once and then that is all there is to it.

But there is simultaneous address and cache analysis in other words we are going to look at each instruction, do an address analysis find out what the address access by that instruction are. We also then do the cache analysis for that instructions then go to the next instruction in the sequence.

So, this takes a lot of time because we are doing address analysis and cache analysis simultaneously. The basic idea is to prime the data cache here. See the expansion mode usually will have just one iteration.

(Refer Slide Time: 28:26)



Example

- frac_exp = 0.1 (10%)
- samples = 4
- #regions = 4*2 = 8
- 10% of the iterations will be analyzed in E-mode spread over 4 regions

So, that is what we had in the previous example also. If you look at this E mode it is a Sartwell less than or equal 0 so we really start from 0 and this is only one iteration. So this is 3, region 3 Sartwell less than is equal to 10 the other 1 went up to 9 summery mode went up to 9 so, less than equal to 10 implies 1 iteration so that is the point. (Refer Slide Time: 28:47) So usually, higher incidence of singleton access in the expansion mode than in summery mode.

If we have single singleton access in the expunction mode then the cache update will be accurate. The cache priming aculeate some extant will happen during this expansion mode.

We had only one iteration so possibly we would be accessing only one elements there that is the basic idea. And in the summery mode there is no virtual unrolling and there are fix point iterations so, we go through same court many time until stabilization happen.

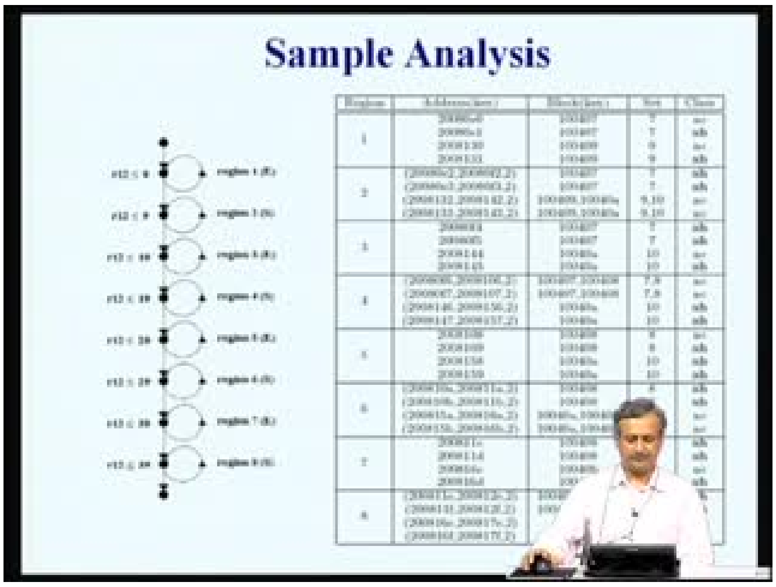
So, we do the address analysis first, get all the address patrician computed then do the cache analysis resuming that the address analysis over. We do not do the address analysis again and again in an instruction by instruction manor as in the expansion mode. This is very fast, but since we have separated the address analysis and cache analysis actually the procession will be lower in the summery mode as the name indicates corresponding to compare to the expansion mode. Summery mode will have many more iterations compare to the expansion node.

And the modes are equivalent for non-loop i portions so, that is obvious because non-loop portions implies going through the whole thing only once so that is the same for both summery mode and the expansion mode.

Why are we doing summery mode and expansion mode separately and why all these, why not just do summery mode or why not just expansion mode do.

In the case of summery mode it is fast, but precession is lower, cache updates are going to be safe, but may not be very accurate. Whereas in the expansion mode since a singleton access are more, it is very accurate and it helps to prime the data cache, but it is very slow. Since we are doing both analysis simultaneously.

(Refer Slide Time: 31:16)



Sample Analysis

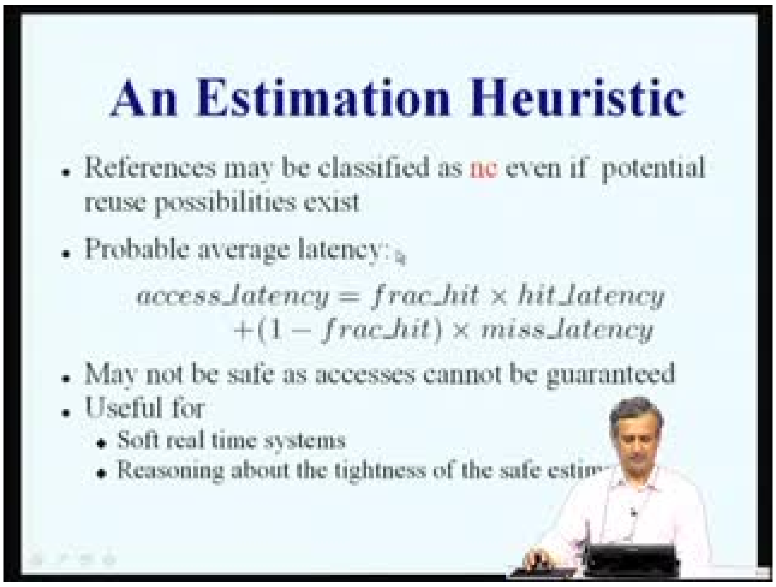
Region	Address(es)	Block(s)	Set	Class
1	[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]	[illegible]
3	[illegible]	[illegible]	[illegible]	[illegible]
4	[illegible]	[illegible]	[illegible]	[illegible]
5	[illegible]	[illegible]	[illegible]	[illegible]
6	[illegible]	[illegible]	[illegible]	[illegible]
7	[illegible]	[illegible]	[illegible]	[illegible]
8	[illegible]	[illegible]	[illegible]	[illegible]

Here is an example of how we classify instruction and do all these analysis. So, these are the various reasons which will be actually analyzed in summery mode and expansion mode.

Here are the various address that were generated this is a CLP for example, whereas these are analyzed in expansion mode so, region 1 is in expansion mode so there are singleton address, region 2 is in summery mode there are CLP is here again region 3 has singleton address and so on.

So each of these reference has been classified as nc or ah depending on how the cache update has happened. So, this is what we get at the end of this type of analysis.

(Refer Slide Time: 32:03)



An Estimation Heuristic

- References may be classified as nc even if potential reuse possibilities exist
- Probable average latency:

$$access\_latency = frac\_hit \times hit\_latency + (1 - frac\_hit) \times miss\_latency$$

- May not be safe as accesses cannot be guaranteed
- Useful for
  - Soft real time systems
  - Reasoning about the tightness of the safe estim

An estimation heuristic so reference may be classified as nc even if potential reuse possibility exist. As I told you we actually guarantee only lower bond on the number of its we do not really give you exact number of upper bond.

Because of that there may be references which are classified as miss and at run time actually possibly would have been a hit so such possibilities exist. Now because of this our worst case execution time estimation is much more than what it really should be. So to avoid this and make it little more accurate WCET estimation more accurate.

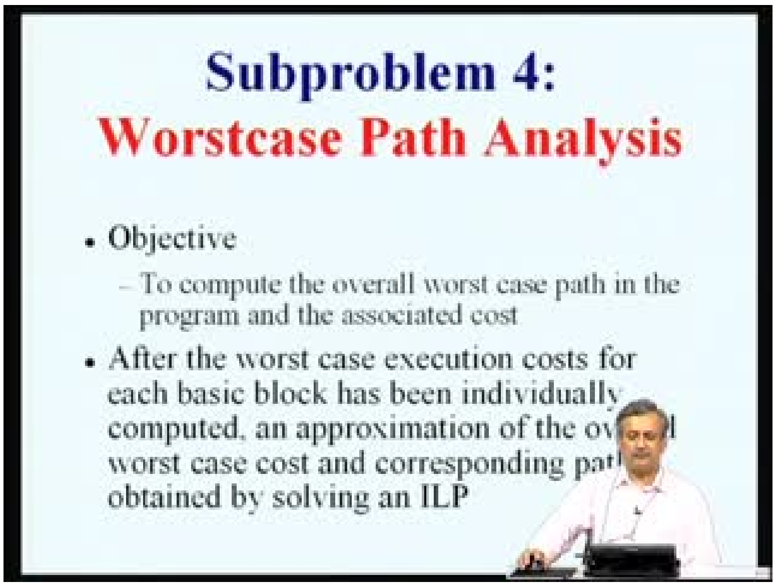
What we say is, we will see the fraction of hits and fraction of misses and then access latency will be fraction hit into hit latency plus 1 minus fraction hit into miss latency.

Instead of classifying every one of them as hit ah are nc here, we actually take the average latency, access latency and use it.

So, in such a case, for those which are nc, we are going to use this access latency and if the fraction of hits is much more; then, we have a access latency which is close to the hit latency. If the fraction of hits is very less and the number of misses is large, then, the average access latency would be close to the miss latency. So, it may not be safe as accesses are guaranteed, but it is useful for soft real time systems and reasoning about the tightness of the safe estimate.

So, each access latency will be taken as this. This is the average latency so; we are going to just use this. So, this gives a soft estimate; it is not safe. So, but it may be useful to look at the soft estimate, which is useful in soft real time systems in which the WCET estimate closer to the real value is more useful. Hopefully, such a soft estimate will be closer to the real value and not far away from it.

(Refer Slide Time: 34:50)



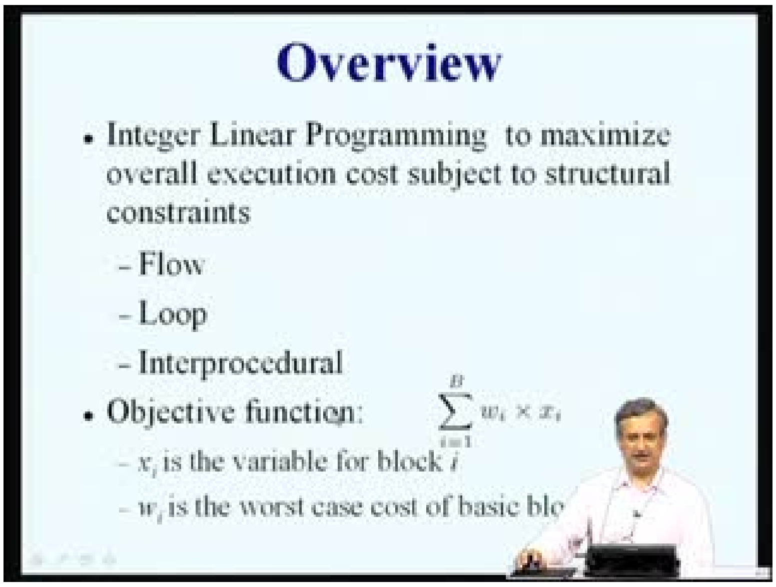
Subproblem 4:
Worstcase Path Analysis

- Objective
  - To compute the overall worst case path in the program and the associated cost
- After the worst case execution costs for each basic block has been individually computed, an approximation of the ov l worst case cost and corresponding pat obtained by solving an ILP

The once we have analyzed the cache model and so on and so forth and each one of the instructions has been tagged as ah or nc we are now ready to compute the worst case path and find out what is the worst case execution time.

So, objective of worst case path analysis is to compute the overall worst case path in the program and the associated cast that would give you the WCET. After the worst case execution costs for each basic block has been individually computed, an approximation of the overall worst case cost and corresponding path is obtained by solving an integer linear program. So, we for each basic block we can now compute the cost we know the various instructions in basic block we have tagged the memory references of each one these instructions as ah or nc. So now, we can compute the time for one of the memory references and there by the time for the entire instruction as well. When we are computing the time the hit or miss etcetera will all be taken care of.

(Refer Slide Time: 36:17)



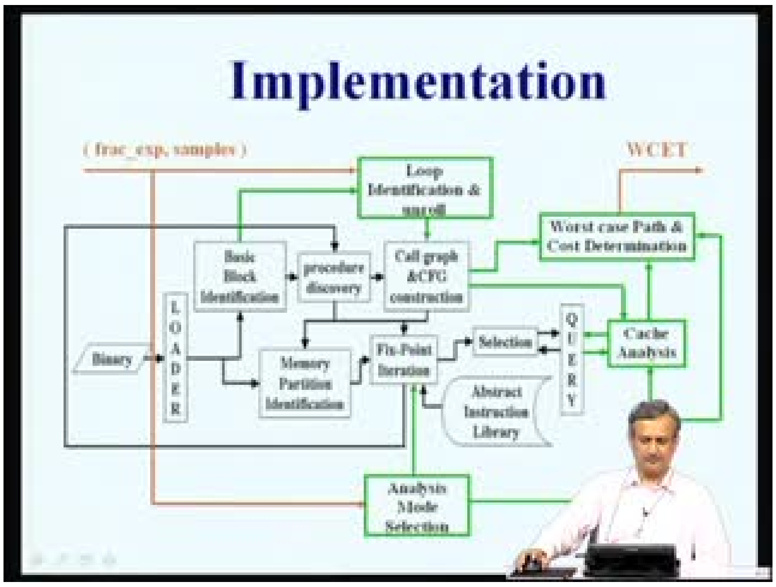
Overview

- Integer Linear Programming to maximize overall execution cost subject to structural constraints
  - Flow
  - Loop
  - Interprocedural
- Objective function: $\sum_{i=1}^{B} w_i \times x_i$
  - x_i is the variable for block i
  - w_i is the worst case cost of basic blo

So now, once we do this we are ready to compute the overall execution cost for the worst case. So integer linear programming to maximize overall execution cost subject to structural constraints.

There are flow constraints, there are loop constraints, and there are interprocedural constraints. But it is actually a bit complex to going to details of each of this constraints in this lecture. So, it suffices to know that there are many of such constraints and the objectives function is sigma equal to 1 to b that is, a number of basic blocks w i cross x i; w i is the x i is the variable for block i. So, in other words, if x i is 1 then the block is counted for the worst case and if x i is 0 the block is not counted for the worst case so, that is the path, it is in the path of worst case execution or it is not in path of worst case execution.

(Refer Slide Time: 37:22)



Implementation

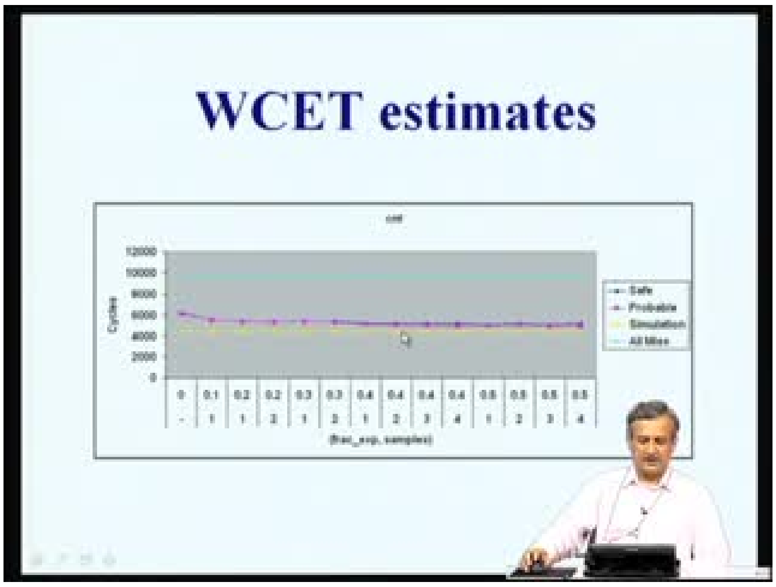
(frac_exp, samples)
WCET
Loop Identification & unroll
Worst case Path & Cost Determination
Basic Block Identification
procedure discovery
Call graph &CFG construction
LOADER
Binary
Memory Partition Identification
Fix-Point Iteration
Selection
QUERY
Cache Analysis
Abstract Instruction Library
Analysis Mode Selection

And w i is the worst case of execution basic block i, which we have computed by looking at each of instructions in the basic block. The implementation is here; the block diagram is here; the binary is fed to the loader. Then the basic block identification is carried out here. Memory partition identification is carried out here. Procedure discovery happens here because, each of this is not trivial we are not analyzing the binary. Loop identification and unrolling happens here call graph and control flow graph construction happens.

Then the fix point iteration analysis mode selection whether, it is summary mode or it is really expansion mode etcetera is chosen here. Then fix point iteration happens abstract instruction library is present here. So, that is useful selection then, we actually query the cache analysis whether, it is a hit or a miss etcetera.

Worst case and based on this information is passed to the worst case path and cost determination unit, which in turn gives you the WCET. So, there are many block here each of which contributes to the analysis phase of the WCET.

(Refer Slide Time: 38:46)


WCET estimates

So, here is the set of results for various types of a bench mark WCET bench mark. So, the top line which is in light blue corresponds to the all miss situation. So, assuming that all the memory references are misses no cache at all so, it obviously this will be a flat line.

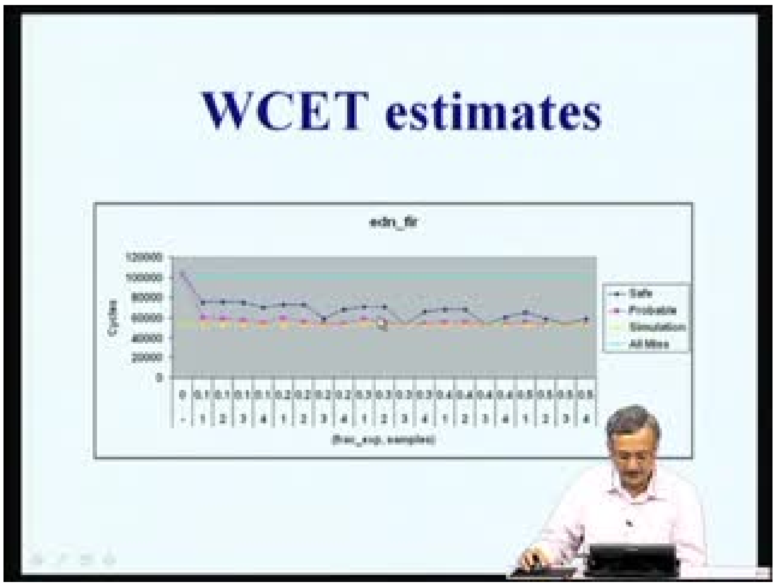
So, here we are really actually doing it to for various frack expend samples so, parameter these 2. So, either 0.1 and 1 , 0.2 and 1, 0.2 and 2 etcetera.

Then the simulation that is, trying to run the program as many times as possible for various types of inputs; and trying to determine over a very large number of inputs what is the worst case execution time. So, that gives this yellow line. So, that in some senses the very accurate possible WCET.

So, the pink one is what we have actually computed as a soft estimate and then, this black one is the safe estimate according to ah r nc classification.

So, the other one uses average latency this probable estimate uses average latency whereas, safe estimate always uses lower bound on the hits and therefore, in this example of course, the probable one and the safe estimates are very close.

(Refer Slide Time: 40:34)

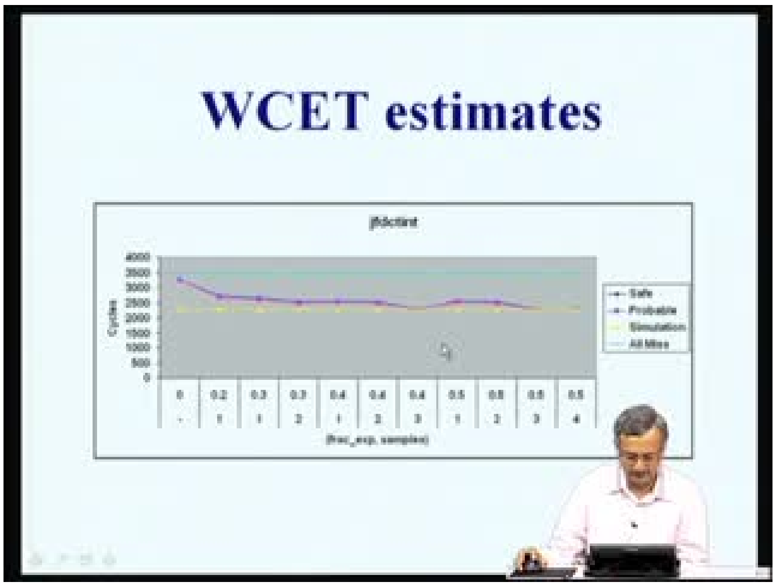


WCET estimates

So, our estimate technic has been very good, but if look at this particular bench mark the safe estimate actually is like that, the probable estimate is much closer to the simulation or the accurate actual one.

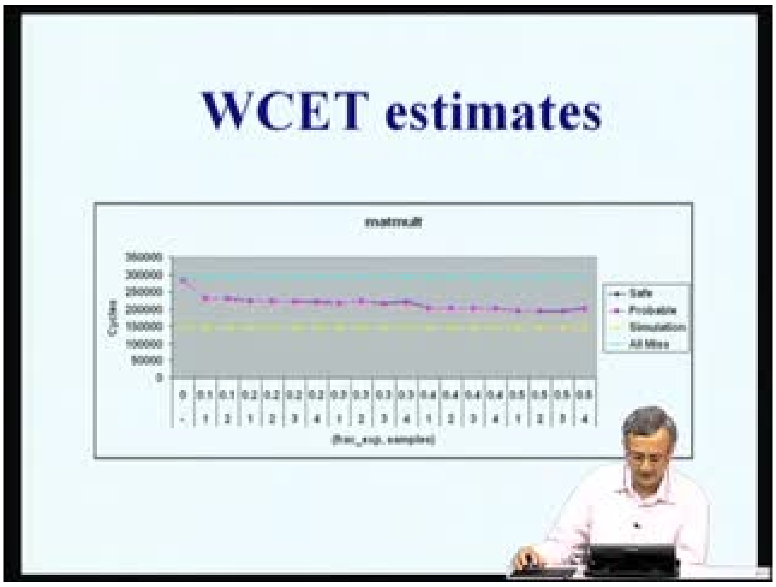
The yellow is closer to actual one and this pink one which is soft real time as WCET estimate is closer to the actual WCET whereas, our safe estimate is much higher than the actual WCET.

(Refer Slide Time: 41:06)



WCET estimates

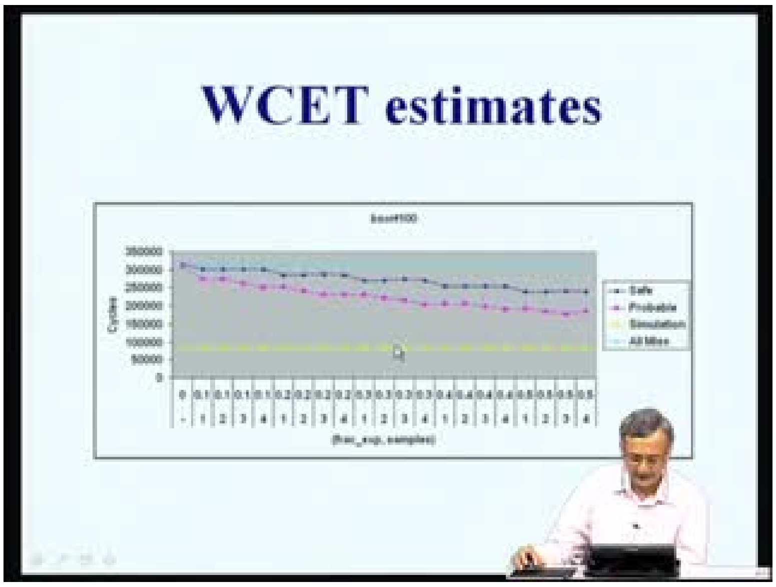
(Refer Slide Time: 41:28)



WCET estimates

Here again both probable and safe estimates are almost close to each other and you remember we seem always to be above the yellow line that is even our probable estimate seems to be fairly safe in this case but, it may not happen all the time. Even here our safe estimate and probable estimates are very close, but both of them are much higher than the actual value.

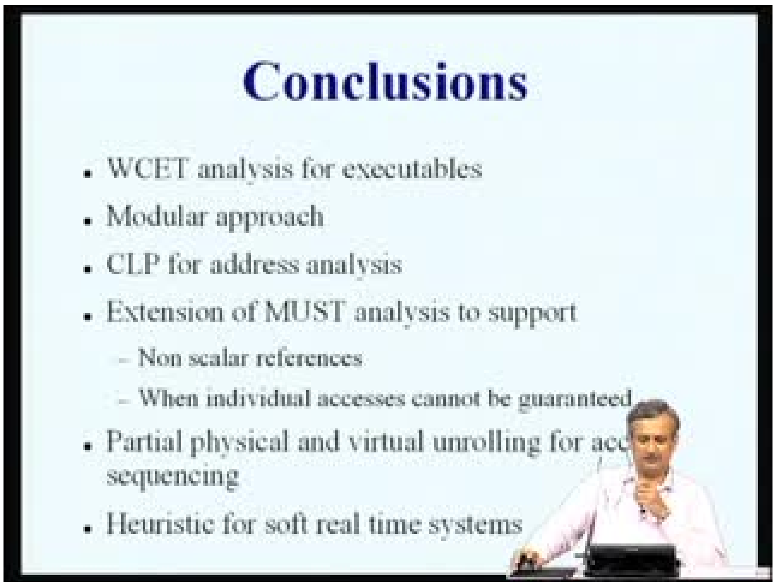
(Refer Slide Time: 41:39)



WCET estimates

In the bubble sort example the it’s really bad so, here is the actual value this is the probable value and which itself is much higher, our safe estimate is even higher all misses is above. So, this goes to show that our estimates are actually safe, but may actually much more than what exactly happens practice. The actual WCET may be much lower than the value that we estimate during WCET analysis. Since we need a safe estimate then we always going to be little higher.

(Refer Slide Time: 42:27)



Conclusions

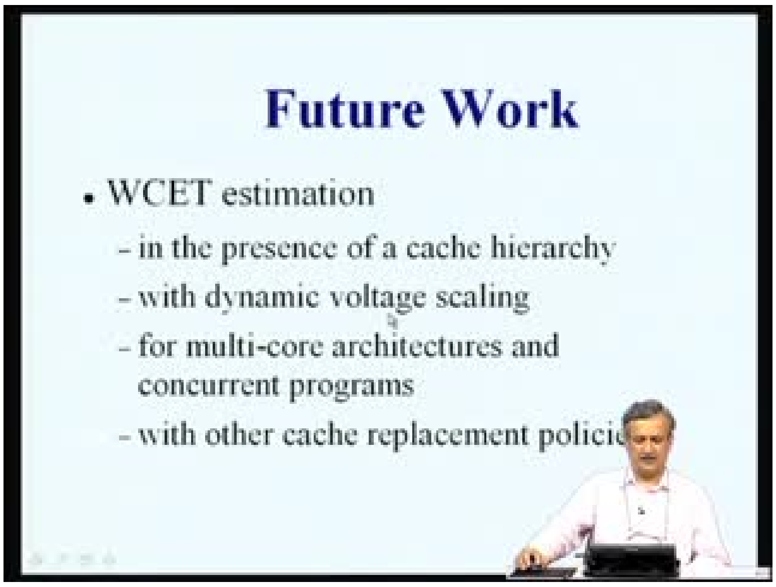
- WCET analysis for executables
- Modular approach
- CLP for address analysis
- Extension of MUST analysis to support
  - Non scalar references
  - When individual accesses cannot be guaranteed
- Partial physical and virtual unrolling for acc sequencing
- Heuristic for soft real time systems

The general research direction in WCET has always been to bring that safe estimate as close to the actual value as possible. That is not so easy because our cache model etcetera very conservative. How to actually create metro cache model etcetera is not understood as of now.

Concluding WCET analysis for executables is what we have attempted and it is a modular approach, we have use circular linear protraction for address analysis, extension of its an extension of must analysis to support non scalar references and when individual accesses cannot be guaranteed for example, array references etcetera.

But a partial, physical and virtual unrolling for access sequencing has been used and we have use heuristics for soft real time systems, which uses lightly better estimate of the WCET value, closer to the actual value, but it may not be safe most of the time.

(Refer Slide Time: 43:40)



Future Work

- WCET estimation
  - in the presence of a cache hierarchy
  - with dynamic voltage scaling
  - for multi-core architectures and concurrent programs
  - with other cache replacement polici

In future WCET estimation in the presents of a cache hierarchy, with dynamic voltage scaling, multi-core architectures, concurrent programs and other cache replacement polices etcetera is in store. So, we see a bright future for a research WCET estimation. Thank you very much this is the end of lecture.