Embeded Systems - Design Verification and Test

Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 14
System Level Analysis

In the last few lectures we have been looking at various phases of Embedded Systems Design. We started with the situation in which the embedded systems designer had just a hazy and rough idea of the embedded application that he wants to design. Then he jots down this idea in English as accurately as possible, specifying the inputs to his system embedded computing system, the expected outputs from the system and describing as accurately as possible the functionality of his embedded system, the behaviour of his embedded system, how the embedded computing system transforms the input into outputs he writes that in English.

Now, given this English language description the designer embarks on to the stage of formal modelling of this behaviour, using tools such as the sequential program modelling in languages and then the tools such as the hierarchical and concurrent finite state machines, the program state machines etcetera. And after this he has a model the formal models describing his entire embedded systems behaviour, with possibly specifications also for performance constraints such as latency between two components may be area related constraints power related constraints etcetera.

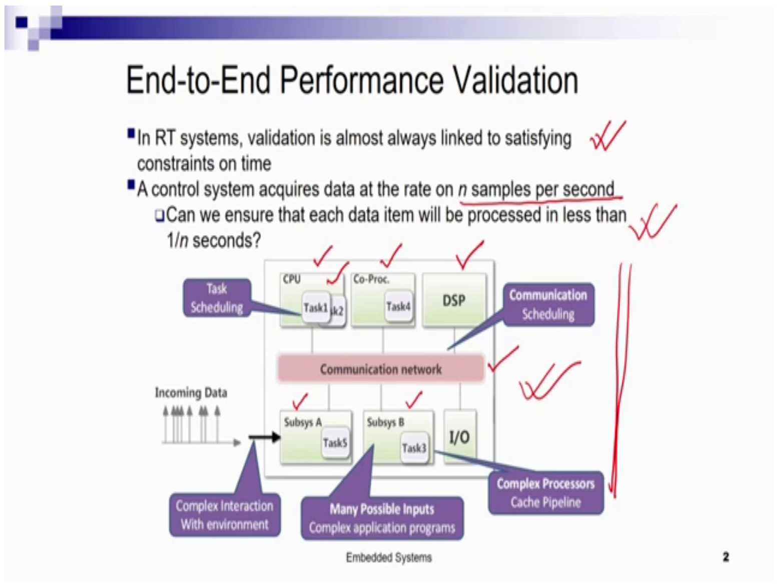
We also saw that there are a few modelling strategies such as discrete event systems, which allow direct synthesis of the controller taking into account the platform constraints as well. However, during that design we also told that for very complex embedded systems such accurate modelling of the behaviour of both resources, the specification and the system it may not be possible ok. For those such complex embedded systems when such accurate modelling, so that synthesis can be done directly is not possible we obtain from the model task graph like structures from the program state machine model say, we obtain structures like that a task graph corresponding to the embedded computing system that we want to design.

Given such a task graph like structure, we then embark on to the stage of hardware software partitioning, in which we decide which of the task loads in this task graph will be executed on general purpose processors say or using a C language high level C language and compiling it into its corresponding machine language or other high level languages, or and also which task will be designed in hardware ok. And you know hardware which essentially means that for that functionality we will build, we will build a single purpose processor a digital circuit which will be finally, fabricated on to a fabricated as a chip or many such functionalities combining together will be formed into a chip and placed onto the embedded system, in on the embedded system with other components.

Then at this stage we could also decide other implementation platforms, such as we can have instead of having these extremes of completely a general purpose processor or a completely customized hardware, we can have other implementation platforms as well such as specialized processors such as digital signal processors DSP's ESOP's application specific integrated, instruction set processors. So, application specific instruction set processors ESOP's or even say SIMD processor Single Instruction Multiple Data stream processors. And FPG as Field Programmable Gate erase various other computing platforms with various degrees of flexibility and performance into performance that it provides, can be chosen as our desired implementation platform further for a given functionality. And those functionality described as tasks in the task graph.

Then we saw how the architectural design of the hardware can be obtained ok. The so, from the functionality described in say a high level very log or VHDL code, we saw steps which give provides you provide which we have shown steps which ultimately derives the data path and the control path for the hardware that we want to design. Of course, there are subsequent phases before we can actually realize a package to chip and, we have discussed those phases, but we have not gone into those phases for example, we saw that there will be logic design then circuit design then physical design pack fabrication packaging etcetera ok. And those will be more a part of v l of cad for VLSI design course and not an embedded systems course and, we have not shown that. After we have understood got an overview as to how hardware is designed, we will now have to embark on system level analysis and design.

(Refer Slide Time: 06:19)



End-to-End Performance Validation

- In RT systems, validation is almost always linked to satisfying constraints on time
- A control system acquires data at the rate on *n* samples per second
  - Can we ensure that each data item will be processed in less than 1/*n* seconds?

Task Scheduling
CPU
Task1
k2
Co-Proc.
Task4
DSP
Communication Scheduling
Communication network
Incoming Data
Subsys A
Task5
Subsys B
Task3
I/O
Complex Interaction With environment
Many Possible Inputs Complex application programs
Complex Processors Cache Pipeline

Embedded Systems 2

So, after we have designed, we will have a special we will have a system in which different tasks in the task graph will be executed on general purpose processors on specialized co processors, on specialized digital signal processors, or we will have customized hardware such a subsystem A and subsystem B. And different tasks will be implemented on different such subsystems hardware’s and software’s.

And each of these components be it general purpose processor specialized processors or customized hardware, all these things need to be able to talk together talk among each other, because the tasks need to communicate information among each other. And this happens by means of an interconnection network, such interconnection networks could be buses could be a shared bus. And hence if it is a shared bus and multiple tasks are communicating among themselves, the shared bus needs to be appropriately scheduled. So, that there can be smooth seamless communication between tasks.

Similarly, if multiple tasks share the same implementation performed platform such as a general purpose processor, we need to schedule the tasks in on to them if we have multiple such general purpose processors, let us say platforms which on which multiple tasks are scheduled we need to be able to do multiprocessor scheduling as well. And we saw that there are um there are many things issues to consider, when we consider a system and how performance will be satisfied, whether performance will be satisfied by implementing this embedded system on this on this given implementation platform with multiple components.

Suppose this embedded system does some big computation as a stream ok, it takes input at the rate of n samples per second does a lot of computation through these components, through the through the all these components that we have here and produces data every one by n seconds. So, sorry what I am wanted to mean is that, let us say the data comes at the rate of every 1 by n seconds. And then the output also has to be provided at the rate of 1 by n seconds. So, in real time embedded systems validation of performance.

So, what we want to do here we want to validate whether the performance requirements that that are stipulated for my embedded system, will be met if I implement it on to such a platform, implementing some components in hardware some components in software connecting it with such an interconnection network ok. Applying certain scheduling principles on the buses and the processors, if I do all these things will I be able to meet the required performance that my embedded system needs to meet, that is that is what we need to validate ok.

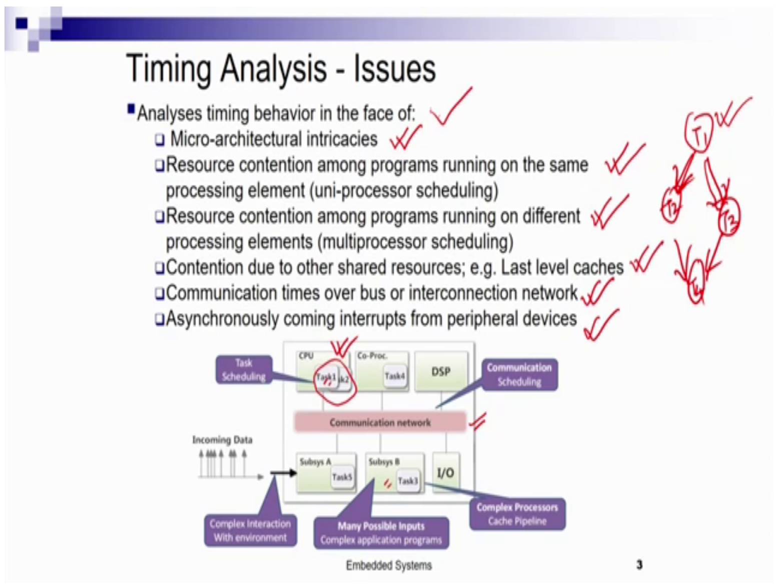
So, with respect to embedded systems validation is almost always linked to satisfying constraints on time. So, we will see this important how such this important constraint can be met. For example, let us say this embedded system is a control system which acquires data at the rate of n samples per second. And then we need to validate whether this can we ensure that each data items will be processed in less than 1 by n seconds.

So, what will happen if you are always able to process this data within less than 1 by n seconds. And if we can certify that it will never take more than 1 by n second, we will not require we will not require buffers to store unprocessed data ok. If we cannot ensure that we will require some finite size buffer to store unprocessed data. Suppose this embedded system the time taken is not always fixed, it is around one by n seconds on an average or a bit less than 1 by n seconds on an average, but sometimes the time required to process the input data could be large.

So, therefore, it may not always be the case that the data will be processed within less than 1 by n seconds. Hence we need buffers to store for the situations in which there are data items which could not be processed, because input samples will always be coming at 1 by n seconds.

Now, therefore, what we want to stress here is that, we need to correctly analyze, what will be the time taken by the embedded system to process each component in data, what is the upper bound on the time that will be required by this embedded system to process each data component and what we will see is that it is not very easy to do so, to obtain such estimates.

(Refer Slide Time: 11:17)



Timing Analysis - Issues

- Analyses timing behavior in the face of:
  - Micro-architectural intricacies
  - Resource contention among programs running on the same processing element (uni-processor scheduling)
  - Resource contention among programs running on different processing elements (multiprocessor scheduling)
  - Contention due to other shared resources; e.g. Last level caches
  - Communication times over bus or interconnection network
  - Asynchronously coming interrupts from peripheral devices

Embedded Systems 3

Why are obtaining such end to end estimates not that easy. Because analysis of timing behaviour must be done in the face of micro architectural intricacies for example, what is the pipeline structure, how many pipelines do we if I do not have a pipeline structure each data component, we need to say k time units, if I have a pipeline structure finally, I may require only one time unit on average to process each data component.

For example, we have to take care of the resource contention among programs running on the same processing element, that will be done through uni-processor scheduling we saw that there are multiple tasks onto the same processing element. And when will each task execute such that both tasks will be able to produce their outputs within given deadlines, that has to be correctly analyzed and scheduled and that will be done via uni-processor scheduling we will see uni-processor scheduling principles in the future classes. Starting possibly from the next class and we will also see that there can be resource contention among multiple programs running on different processing elements.

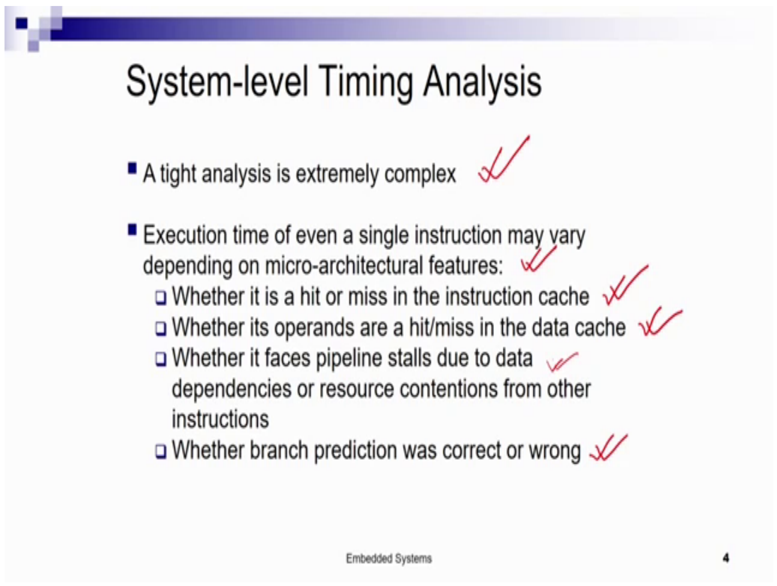
Let us say if this embedded system would have multiple general purpose processors onto which a set of n tasks have to be scheduled on a set of m processors, how do you do how do we do that seamlessly so, that deadlines of all the task instances all instances of all tasks will be met. So, we saw we said that each of these tasks will be fed data at a rate of some samples. And each instance for which this code runs will be called a job or an instance, and the task is this series of instances over which it receives data and produces output. So, there could be contention due to other shared resources like last level caches. So, we can have last level shared caches and when one task executes on a processor, other tasks data may be flushed and therefore, when the other tasks comes back to the processor it will not find the data in cash and it may incur a lot of cache misses and hence the time taken could be large.

So, the time depending on cache misses could vary a lot the time taken by a task to complete can vary a lot depending on whether there were cache misses ok. Similarly there could also be the case, depending on whether there were a lot of page faults or not the time taken by a certain task can vary time. The overall time taken could also may vary depending on the computation times over the bus or the interconnection network. As I told that this tasks multiple tasks in that this is a task graph structure where tasks, communicate data output of one task goes to the input of another task. So, I have task graph like structures like right. So, I have structures like this and the output from one task, say T 1 T 2 T 3 and T 4. So, output of T 1 goes to two outputs from T 1 goes to T 2 and T 3.

And therefore, such dependencies so, if T 1 is implemented on say the CPU and say T 2 is implemented on say the subsystem I need to be able to communicate the output of T 1 to T 2, through this communication network or through this bus. And there will be multiple tasks will which will be communicating their data through the bus to other tasks in other components. And this has to be seamlessly done to do this seamlessly in over the shared communication medium the shared bus, which is shared by multiple tasks multiple components you need to be able to properly schedule the bus.

We will see bus scheduling strategies as well, time could also time taken by the system, could also depend on asynchronously coming input interrupts from peripheral devices. Other peripheral devices interrupts and therefore, the tasks may have to wait until such interrupts are serviced and times. The overall time taken by the system could vary a lot depending on whether interrupts from peripheral devices were received ok. So, these were just a few issues to show that the time taken it is very difficult to tightly estimate what will be the time taken by an embedded system; however, tighter estimates do help us get estimates of how much resource that will be required. And hence we will not waste resources, because otherwise we have to keep a lot of resources thinking just of the worst case, worst case time that might be taken. And hence the design will not be that streamlined would not be that cost effective, hence tight estimate of the time taken is required and therefore, we do that.

(Refer Slide Time: 16:39)



System-level Timing Analysis

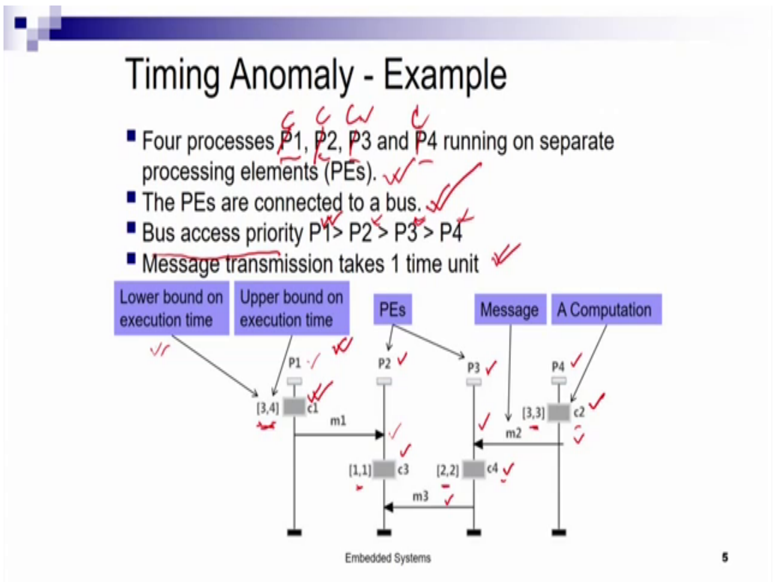
- A tight analysis is extremely complex ✓
- Execution time of even a single instruction may vary depending on micro-architectural features: ✓
  - Whether it is a hit or miss in the instruction cache ✓
  - Whether its operands are a hit/miss in the data cache ✓
  - Whether it faces pipeline stalls due to data dependencies or resource contentions from other instructions ✓
  - Whether branch prediction was correct or wrong ✓

Embedded Systems 4

So, we just understood that a tight analysis is extremely complex, execution time of even a single instruction may vary depending on micro architectural features as I told you what could be the different micro architectural features. For example, whether there was a hit or miss on the instruction cache, whether its operands are a hit or miss on the data cache. Whether it faces pipeline stalls due to data dependencies or resource constraints from other instructions.

So, whether do we have data hazards, read after write for example, such you if we have such data hazards, then we have to face pipeline stalls and time required could be more. Whether branch prediction was correct or wrong, so, depending on whether a branch at a particular point was predicted correctly the time taken, because if the branch prediction is correct, it already executes of few instructions within the branch within the predicted branch, if it is wrong all those instructions have to be flushed. And the other branch has to be taken. So, we save time if the prediction is correct on the other hand, we if the time required could be more if branch prediction is wrong.

(Refer Slide Time: 17:58)



Timing Anomaly - Example

- Four processes P1, P2, P3 and P4 running on separate processing elements (PEs).
- The PEs are connected to a bus.
- Bus access priority P1> P2 > P3 > P4
- Message transmission takes 1 time unit

Embedded Systems 5

Now, it is not only we will be told that it depends on the time required by the tasks and the scheduling within the processor, but it is also equally important about, what is the time taken on the bus. We told that, but we will see that with an example that that why system level analysis of both tasks as well as the bus scheduling has to be taken into account, if not done correctly there could be anomalies. And this in this through this example we will see that.

Let us say that we have four processes and let us say these processes are four processes, in an automatic system such as the anti lock braking system the cruise control system, and let us say the fuel injection system and let us say the brake controller. These are four processes and we also suppose that these four processes for the sake of simplicity that, these four processes are running on separate processing elements or processors. So, so each process is in a separate processor.

However, these four processing elements are connected by a single shared bus and, we also have a very simple scheduling strategy for the bus, we say that whenever you have multiple bus messages to send through the bus, how do you decide which message can be sent through the bus. Multiple messages from multiple processors are there to be sent through the, how do you decide which message will be sent through the bus. It will be decided through this simple bus access priority let us say, where priority of the processing element processing element or process P 1 is highest, followed by processing element P 2 followed by processing element, P 3 and followed by processing element P 4.

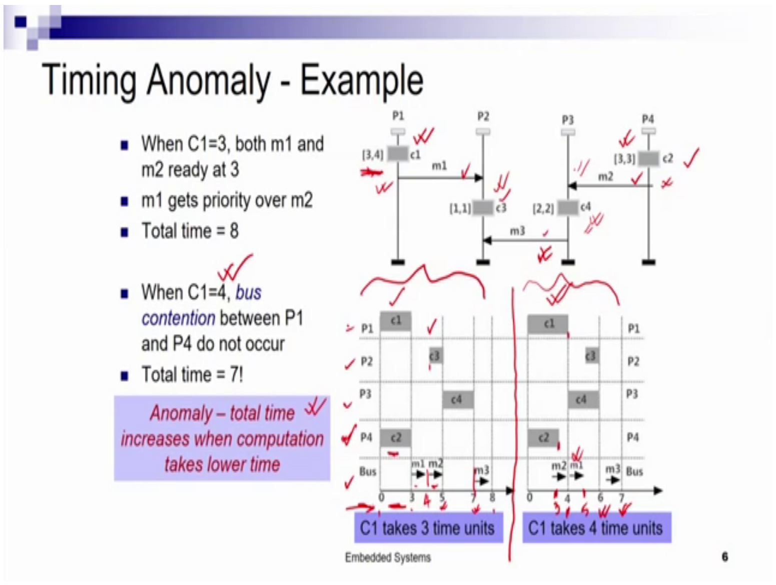
So, P 4 has the least priority to get the bus and P 1 has the highest priority to get the bus. And also for the simple sake of simplicity, we assume that all message transmissions only take one unit time. So, just one piece of data which takes one unit of time we consider that. And each task is marked so, these are the tasks C 1. So, these are the processing elements P 1 and the processes actually are C 1 C 2 and C 3 and C 4 these are the processing elements. So, the sorry these are the processes or the tasks.

So, C 1 is one task C 2 is another task C 3 is another task C 4 is the other task. So, C 1 is mapped onto processing element P 1 C 2 is mapped onto processing element P 4 C 3 is mapped onto processing element P 2 and C 4 is mapped onto processing element P 3 ok. And with beside each task C 1 C 2 C 3 C 4 we have kept the lower bound on execution time and upper bound on execution time.

Let us say we have somehow come to know what will be the what will overall data that this is this over all the possible data, the which C 1 can take as input through its arguments or two from outside, the different control paths that it follows for example, this task is C 1 over all such control paths over all such data the least execution time, that it takes is 3 units and the highest execution time that it can take it 4 units. So, over all data that it can ever receive C 1 takes 3 unit of time at least and 4 unit of time at most. For all other computations or tasks the we have kept the lower bound on execution time and upper bound on execution time to be same. For all other tasks the lower bound and upper bound is same.

And we say that after m C 1 completes its execution message m 1 from C 1 goes to C 3. Similarly message m 2 from C 2 goes to C 4 and C 3 and after receiving message m 1 C 3 can execute after receiving message m 2 C 4 can execute. After C 4 does its execution C 4 sends message m 3 to processing element P 2, which is possibly processing element P 2 and then this process this computation component ends.

(Refer Slide Time: 22:57)



Timing Anomaly - Example

- When C1=3, both m1 and m2 ready at 3
- m1 gets priority over m2
- Total time = 8
- When C1=4, bus contention between P1 and P4 do not occur
- Total time = 7!

Anomaly – total time increases when computation takes lower time

Now, let us consider two cases one in which the task one C 1 takes 3 units of time. And the other case in which task one that is C 1 takes 4 units of time. So, when task one takes 3 units. So, this is the case in which task one takes 3 units of time this is the progression of time on the x axis, we have progression of time these are the four processing elements and this is the bus. This one is the bus these ones are the four processing elements.

So, when C 1 takes 3 units of time, the computation on P 1 task T 1 executes in the interval 0 to 3 and completes at time 3 C 2 on the other hand execute in parallel on P 4 on P 4 and, it also takes 3 units of time and completes at time 3 it takes 0 to 3. Now, at time 3 both messages m 1 and m 2 are ready and they are sent one after another. So, let us say because m 1 and m 2 are ready and both are ready and because m 1 has a higher priority m 1 is from m 1 is from P 1 processing element P 1 and therefore, m 1 has higher priority than m 2 which is on processing element P 4.

So, although both are ready at 3 m 1 sends its message first between 3 and 4 and then m 2 sends its message between time at 4 and between 4 and 5 at this slot m 2 sends this message before m 1. And after m 1 sends its message C 3 starts at time 4 itself at time 4 C 3 C 3 starts at time 4 and as we have we have seeing C 3 takes only one unit of time. So, therefore, at time 5 C C 3 completes, at time 5 C 3 completes and m 2 also arrives at processing element P 3 at time 5.

So, now C 4 can start C 4 starts at time intervals at time 5 and it requires 2 units of time and therefore, it ends at time 7 after C 4 completes the final message m 3 that is the output can be sent at time 7. So, at times 7 after C 4 completes the message m 3 can be transmitted. So, total time to end this whole computation processes 8 time units from 0 to 8 from 0 from 0 to 8.

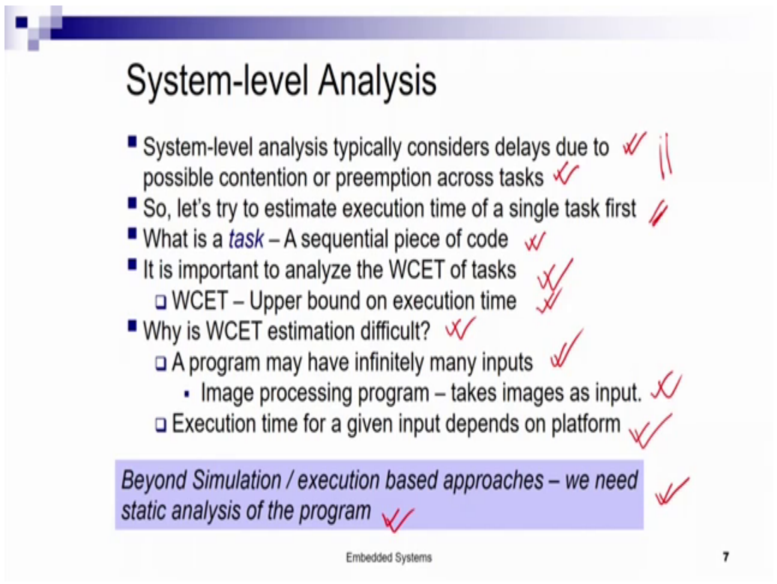
Now, when C 1 takes fours time units to execute so, it is more so, in the first case C 1 takes 3 time units to execute. Now, in the second case C 1 takes more time to execute, now so both tasks on P 1 C 1 takes 4 time units and completes at time 4 and on P P 4 C 2; however, takes only 3 time units, because its upper bound and lower bound are both 3 time units and it executes on P 4 and completes at time 3. Now, because C 2 C 2 completes at time 3 and at that time C 1 is not C 1 has not finished. So, message m 1 is not ready at time 3 so, at time 3 message m 1 is not ready.

So, the only ready message is m 2 and that can be transmitted over the bus and it is it is received at P 3 at time 4 at time 4 ok. When this one is received at time 4 so, two things happen at time 4 C 4 starts, because m 2 has already been received C 4 can start at time 4. And C 1 also completes a time 4 so, message m 1 can be transmitted at time 4 over the bus. So, m 1 is transmitted and completes its transmission at time 5 at which time C 3 can start, C 3 takes only one unit of time and both of them complete at time 6. So, C 4 which takes 2 units of time also complete at time 6 C 3, which takes only one unit of time starting at time 5 it completes at time 6. Now, at time so, after C 4 completes at time 6 message m 3 can be transmitted and the whole transfer this message final message m 3 completes its transmission at times 7.

So, now we see that we have a case in which when the computation takes less time, the whole process this whole process takes more time to complete 8 time units. So, when C 1 takes 3 time units to execute the overall time required by the embedded system to complete this process is 8 time units; however, when C 1 takes more time so, sorry when C 1 takes less time to execute the whole process takes 8 time minutes, which is more amount of time when C 1 takes when C 1 takes more time, then the entire process takes less time to complete.

So, when C 1 takes 3 time units the overall process completes in time 8 time units, when C 1 takes 4 minutes the overall process taken is only 7 time units ok. So, this is an anomaly why because the total time increases when computation takes lower amount of time. Now, this shows that we need to carefully do the scheduling of both the tasks and the processing elements together, to correctly analyse whether the given embedded system will meet its performance estimates for example, time estimates.

(Refer Slide Time: 29:07)



System-level Analysis

- System-level analysis typically considers delays due to possible contention or preemption across tasks
- So, let's try to estimate execution time of a single task first
- What is a *task* – A sequential piece of code
- It is important to analyze the WCET of tasks
  - WCET – Upper bound on execution time
- Why is WCET estimation difficult?
  - A program may have infinitely many inputs
    - Image processing program – takes images as input.
  - Execution time for a given input depends on platform

Beyond Simulation / execution based approaches – we need static analysis of the program

Embedded Systems 7

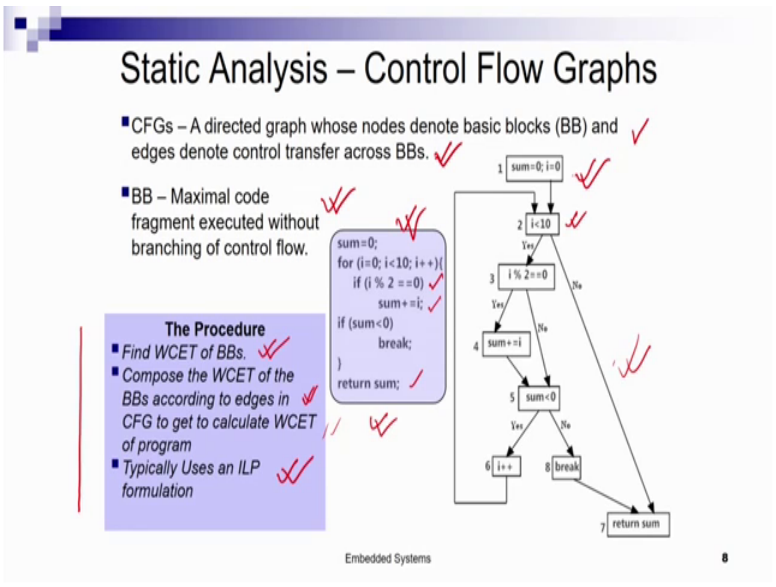
Now, we are seeing that in order to do system level analysis, we have to consider delays due to possible contention and pre-emption across tasks. So, we have to take the case of contention on the bus for messages, contention on the processors for the execution of tasks. Now, in order to do all these analysis on contention we need to have an idea of what amount of time a task is going to take. Now, this is not very easy to do again. So, therefore, we have to first try to estimate the execution time that will be taken by a single task.

Now, what is the task a task is a sequential piece of code. So, it is important to be able to analyze the worst case execution time of task, which is its upper bound on the execution time, why is such worst case execution time estimation difficult, because the program may have infinitely many inputs. For example, let us say an image processing program, it can take any image as input and it will process that image. So, the input data could be infinite and depending on the infinite in depending on the input the program can take different control path, within it is control flow graph we have already studied control flow graph.

Let us say flow chart it can take different control paths in the flow chart of the program. So, therefore, and each such control path may end up taking a different amount of time. So, therefore, the program can take different amounts of time to complete, depending on which control path is followed by it and which control path is followed by it depends on the data that it takes. The execution time of the program also depends on the platform, as we just saw. And the platform when we do the execution time analysis the final platform may not already have been decided.

Therefore execution time estimation is something beyond simulation or execution based approaches, we need to do static timing analysis or static analysis of a program. So, in a system level or execution based approach we run the program over various types of data and try to analyze, what will be the upper bound on the execution time that this program can have. However, what we try to what we try to emphasize here is that such simulation and execution based simulation or execution based approach may not be complete may not be accurate, what we need is static analysis which is followed in a lot of cases. So, we will see how static analysis for worst case execution time estimation of a program is done.

(Refer Slide Time: 32:01)



Static Analysis – Control Flow Graphs

- CFGs – A directed graph whose nodes denote basic blocks (BB) and edges denote control transfer across BBs.
- BB – Maximal code fragment executed without branching of control flow.

The Procedure

- *Find WCET of BBs.*
- *Compose the WCET of the BBs according to edges in CFG to get to calculate WCET of program*
- *Typically Uses an ILP formulation*

```
sum=0;
for (i=0; i<10; i++){
    if (i % 2 ==0)
        sum+=i;
if (sum<0)
        break;
}
return sum;
```

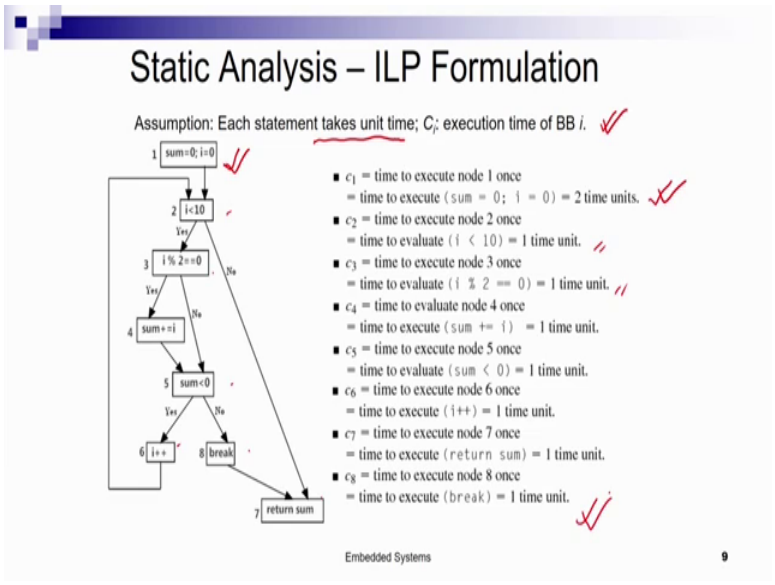
Embedded Systems 8

In order to conduct static analysis, we have to first generate the control and data flow graph, or the control flow graph of the program. We already have seen how this is done in software we will just going to reiterate. Let us say we have a program sum equals to 0 for i equals to 0 i less than 10 i plus plus, if I percentage 2 equals to equals to 0, then sum equals to sum plus i, if some less than 0 break return. Now, the control flow graph of this program will be as follows the first two statements. So, again what was a control flow graph, a control flow graph is a directed graph whose nodes are basic blocks and edges denote control transferred across basic blocks. We have already seen this and what is a basic block, it is a maximal code fragment executed without branching or of control flow. And how do we obtain the control flow graph it is here so, control flow we already know this for this program the control flow graph is this.

The first two statements are in the same basic block, because it is a single sequential execution, the there cannot be any branch in control, it has one entry point and the next the next node has two entry points. So, this goes into a separate basic block and likewise we see. So, this is the control flow graph corresponding to the program that we have here. Now, what is the essential idea how does the static analysis how is static analysis done, static analysis which is the worst case execution time estimation of programs here for our purpose here, how is that done.

We first find the worst case execution time of basic blocks, after we have done that we compose the worst case execution time of the basic blocks according to the edges of the CFG to get to calculate the worst case execution time of the overall program. And this uses an ILP based formulation this typically uses an ILP base formulation which we discuss next.

(Refer Slide Time: 34:12)


1 sum=0; i=0
2 i<10
Yes
No
3 i % 2==0
Yes
No
4 sum+=i
5 sum<0
Yes
No
6 i++
8 break
7 return sum

Static Analysis – ILP Formulation

Assumption: Each statement takes unit time; C_i: execution time of BB *i*.

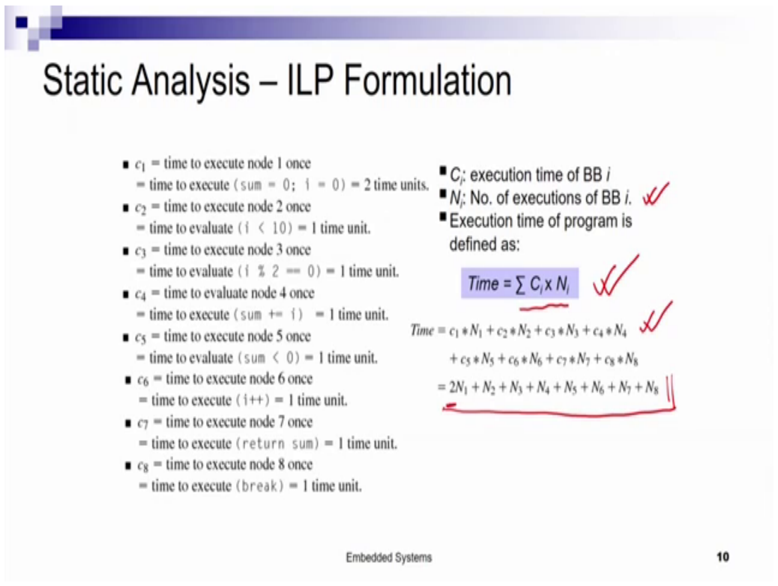
- c_1 = time to execute node 1 once
  = time to execute (`sum = 0; i = 0`) = 2 time units.
- c_2 = time to execute node 2 once
  = time to evaluate (`i < 10`) = 1 time unit.
- c_3 = time to execute node 3 once
  = time to evaluate (`i % 2 == 0`) = 1 time unit.
- c_4 = time to evaluate node 4 once
  = time to execute (`sum += i`) = 1 time unit.
- c_5 = time to execute node 5 once
  = time to evaluate (`sum < 0`) = 1 time unit.
- c_6 = time to execute node 6 once
  = time to execute (`i++`) = 1 time unit.
- c_7 = time to execute node 7 once
  = time to execute (`return sum`) = 1 time unit.
- c_8 = time to execute node 8 once
  = time to execute (`break`) = 1 time unit.

Embedded Systems 9

So, in an ILP based formulation we start with the control flow graph, we first analyse the execution time of each basic block. In calculating the execution time of each placing block we assume that each statement takes one unit of time. So, 1 unit time for each statement, although this assumption may not be very accurately true for, but for our purposes when we are estimating big computation times, each statement taking one unit of time is a reasonable estimate. So, we will assume that each instruction or each statement in the program takes one unit of time and let C i B the execution time of each basic block.

Now, for our case here we see that basic block one has two statements and therefore, its execution takes two time units. If we see all the other basic blocks all the other basic blocks, each of them has one statement each and therefore, all the other basic blocks take 1 1 1 time unit. So, C 1 takes 2 time units C 2 through C 8 all take 1 time unit.

(Refer Slide Time: 35:24)



Static Analysis – ILP Formulation

- c_1 = time to execute node 1 once
  = time to execute `(sum = 0; i = 0)` = 2 time units.
- c_2 = time to execute node 2 once
  = time to evaluate `(i < 10)` = 1 time unit.
- c_3 = time to execute node 3 once
  = time to evaluate `(i % 2 == 0)` = 1 time unit.
- c_4 = time to evaluate node 4 once
  = time to execute `(sum += i)` = 1 time unit.
- c_5 = time to execute node 5 once
  = time to evaluate `(sum < 0)` = 1 time unit.
- c_6 = time to execute node 6 once
  = time to execute `(i++)` = 1 time unit.
- c_7 = time to execute node 7 once
  = time to execute `(return sum)` = 1 time unit.
- c_8 = time to execute node 8 once
  = time to execute `(break)` = 1 time unit.

- C_i: execution time of BB i
- N_i: No. of executions of BB i. ✓
- Execution time of program is defined as:

$$\underline{Time = \sum C_i \times N_i} \quad ✓$$

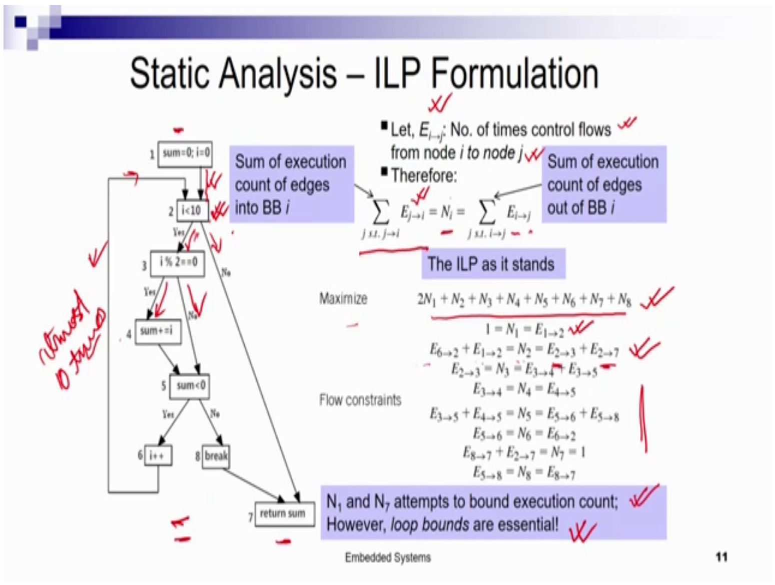
$$Time = c_1 * N_1 + c_2 * N_2 + c_3 * N_3 + c_4 * N_4 \quad ✓$$
$$+ c_5 * N_5 + c_6 * N_6 + c_7 * N_7 + c_8 * N_8$$
$$\underline{= 2N_1 + N_2 + N_3 + N_4 + N_5 + N_6 + N_7 + N_8} \;||$$

Embedded Systems 10

And now given the individual so, give so now what we have a situation where we know the execution time taken by each basic block. So, given the execution time of each basic block, we just need to find the number of executions of each basic blocks, how many times does each basic block execute in the worst case. So, if we know that and we also know that time taken when each time it runs we can find out the total execution time of the program.

So, the total execution time of the program it will be given by summation over C i into N i, where N i is the number of executions of basic block i and C i is the time taken for each execution of basic block i. So, summation C i N i is the total execution time of the program. So, time is given by this and here we know that for our specific case for our CFG C 1 takes 2 units of time so, C 1 is two and all other C 1s are 1. So, this expression essentially gives us the total time that will be taken by the program.

(Refer Slide Time: 36:39)



1 sum=0; i=0
2 i<10
3 i % 2==0
4 sum+=i
5 sum<0
6 i++
8 break
7 return sum
Yes
No

Static Analysis – ILP Formulation

- Let, $E_{i \to j}$: No. of times control flows from node *i to node j*
- Therefore:

Sum of execution count of edges into BB *i*

$$\sum_{j \text{ s.t. } j \to i} E_{j \to i} = N_i = \sum_{j \text{ s.t. } i \to j} E_{i \to j}$$

Sum of execution count of edges out of BB *i*

The ILP as it stands

Maximize $2N_1 + N_2 + N_3 + N_4 + N_5 + N_6 + N_7 + N_8$

Flow constraints

$$1 = N_1 = E_{1 \to 2}$$
$$E_{6 \to 2} + E_{1 \to 2} = N_2 = E_{2 \to 3} + E_{2 \to 7}$$
$$E_{2 \to 3} = N_3 = E_{3 \to 4} + E_{3 \to 5}$$
$$E_{3 \to 4} = N_4 = E_{4 \to 5}$$
$$E_{3 \to 5} + E_{4 \to 5} = N_5 = E_{5 \to 6} + E_{5 \to 8}$$
$$E_{5 \to 6} = N_6 = E_{6 \to 2}$$
$$E_{8 \to 7} + E_{2 \to 7} = N_7 = 1$$
$$E_{5 \to 8} = N_8 = E_{8 \to 7}$$

N_1 and N_7 attempts to bound execution count; However, *loop bounds* are essential!

Embedded Systems 11

And this becomes the objective function in our ILP so, our ILP is this one. And the worst case execution time for the program will be the one which maximizes this value. So, we maximize to N 1 plus N 2 plus N 3 up to N 8 this value the maximum value that is possible. So, we have to find valuations possible valuations for N 1 N 2 N 3 N 4 and for that value of of these decision variables N 1 through N 8 for which the value of so, N 1 through N 8 the number of times each basic block executes here are the decision variables.

And we have to find valuations of those decision variables, for which the time taken by the program is maximum. And after getting the decision variables we need to have constraints on these decision variables, we need to have bounds on each of these N 1 N 2 up to N 8 to obtain these bounds, we will see we will we will see look at the edges. And in particular let E i j is the number of time control flows from node i to node j. So, E i j is the number of time control flows from node i to node j. For example, E 1 2 will be 1, because control flows from N 1 to N 2 sorry from basic block 1 2 basic block 2 only once.

Now, we can say that what will be the number of times that a certain basic block will be executed. So, basic block i will be executed N i times what is N i the number of times that control flows into this basic block ok. So, this is given by this expression here so, E j i tells me that how many times, how many times control flows from basic block j to basic block i. So, for all such js for all js for which you have an edge j for which you have an edge from j to i, if you sum that up you will get the total number of times control flows into this basic block. For example, how many times the does control flow into basic block two it is E 1 2 plus E 6 2.

So, E 1 2 is this one E 1 2 plus E 6 2 so, number of times control flows from 1 to 2 and number of times control flows from 6 to 2, these two together tell me what is the number of times that control can flow into basic block 2. And similarly it will be the same as the number of times that control flows out of the basic block. So, the number of times the basic block is executed also equal to the number of time control flows out of the basic block.

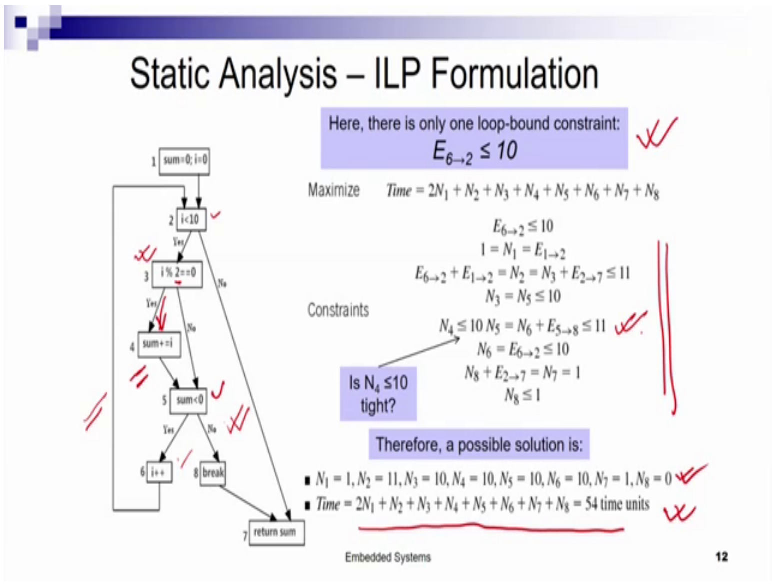
And essentially this property will be used to generate the constraints for example, for our case the number of times control flows into this basic block 2 is equal to the number of times these basic block executes and is equal to the number of times these two branches are taken the ok. So, E i j where i is basic block i and j could be all other edges from i to j so, there are 2 edges from so, you have E 2 3 and E 2 7. So, so here we can say that N 1 is executed once and E 1 2 is once, how many times is N 2 executed E 6 2 plus E 1 2 equals to N 2 equals to E 2 3 plus E 2 7 as we told just here for this case.

Similarly, we have for other cases as well for example, for N 3 if we see what do we have for N 3 control comes to this block from 2 so, E 2 3 equals to N 3 equals to E 3 4, there are two outgoing edges from this basic block. And therefore, it is N 3 equals to E 3 4 plus E 3 5. Similarly, the other constraints so, we can obtain all the other constraints, but can we get any constraints from it although we know that this nodes 1 and to node 7 node 1 and node seven will execute only once. And these are the only two ways in which we can constrain the number of times these nodes execute; however, we have loops inside it if you do not give loop bounds ultimately the worst case execution time is going to become infinity. So, therefore, it is essentially put essential to put bounds on the loop to have practical execution time estimates.

Now, N 1 and N 7 attempts to bound the execution count attempts to bound the execution count; however, loop bounds are essential. Now, how can we obtain loop bounds if we look at this we will see that if i is less than 10 we take this one or if otherwise we go outside the loop. So, we get inside the loop if only when i is less than 10 and i starts with

0. So, therefore, this loop can be executed at most 10 times this loop will be executed at most 10 times hence, we can bound we can say that that at most four nodes N 3 N 4 N 5 N 6 all these nodes will be executed at most N at most 10 times can we do better is the is a question.

(Refer Slide Time: 42:50)



1 sum=0; i=0
2 i<10
Yes
No
3 i % 2==0
Yes
No
4 sum+=i
5 sum<0
Yes
No
6 i++
8 break
7 return sum

Static Analysis – ILP Formulation

Here, there is only one loop-bound constraint:
$E_{6\to 2} \le 10$

Maximize $Time = 2N_1 + N_2 + N_3 + N_4 + N_5 + N_6 + N_7 + N_8$

Constraints

$$E_{6\to 2} \le 10$$
$$1 = N_1 = E_{1\to 2}$$
$$E_{6\to 2} + E_{1\to 2} = N_2 = N_3 + E_{2\to 7} \le 11$$
$$N_3 = N_5 \le 10$$
$$N_4 \le 10 \; N_5 = N_6 + E_{5\to 8} \le 11$$
$$N_6 = E_{6\to 2} \le 10$$
$$N_8 + E_{2\to 7} = N_7 = 1$$
$$N_8 \le 1$$

Is $N_4 \le 10$ tight?

Therefore, a possible solution is:

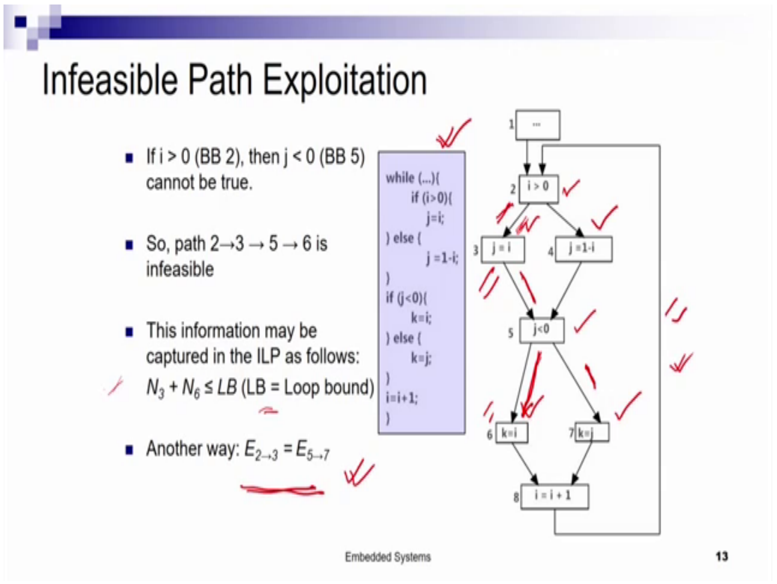
- $N_1 = 1, N_2 = 11, N_3 = 10, N_4 = 10, N_5 = 10, N_6 = 10, N_7 = 1, N_8 = 0$
- $Time = 2N_1 + N_2 + N_3 + N_4 + N_5 + N_6 + N_7 + N_8 = 54$ time units

Embedded Systems 12

So, therefore, here there is only one loop bound constraint E 6 2 E less than equal to 10 E 6 2 is less than equal to 10. So, we can say that N 4 will be so, N 4 is less than equal to 10 here and N 5 equals to N 6 N 5 equals to N 6 N 6 plus 5 8 it is less than equals to 11 so, n 5 N 5 equals to N 6 plus E 5 8 N 5 equals to N 6 plus E 5 8 ok. And that is less than equals to 11, if sum is less than equals to 0 once it will go out. Hence this is the constraint, but so, therefore, when we get these constraints, when we get these more tighter constraints it is possible to get execution time estimates bounds on execution time estimate. And one possible solution one possible feasible solution to this is 54 time units by putting these values on the decision variables.

However tighter the execution time tighter the bounds tighter will be the estimation on the maximum so, lower will the maximum become tighter the constraints that we put. Now for example, how can we do such tighter constraints by infeasible path determination are there paths which are in feasible for example, in this case we can see that yes whenever i is less than 10 you come here to this one; however, I have i percentage two equals to equals to 0. So, i percentage two equals to equals to 0 if i is of i executes 10 times this can be taken at most 5 times it cannot be taken more than 5 times, because i percentage 2 equals to equals to 0 will not become true more than 5 times in a span of 10. So, we can say that a bound on N 4 is actually 5 and not 10 ok.

(Refer Slide Time: 45:10)



1 ...
2 i > 0
3 j = i
4 j = 1-i
5 j<0
6 k=i
7 k=j
8 i = i + 1

Infeasible Path Exploitation

- If i > 0 (BB 2), then j < 0 (BB 5) cannot be true.
- So, path 2→3 → 5 → 6 is infeasible
- This information may be captured in the ILP as follows: $N_3 + N_6 \leq LB$ (LB = Loop bound)
- Another way: $E_{2\rightarrow3} = E_{5\rightarrow7}$

```
while (...){
    if (i>0){
        j=i;
} else {
        j =1-i;
}
if (j<0){
    k=i;
} else {
    k=j;
}
i=i+1;
}
```

Embedded Systems 13

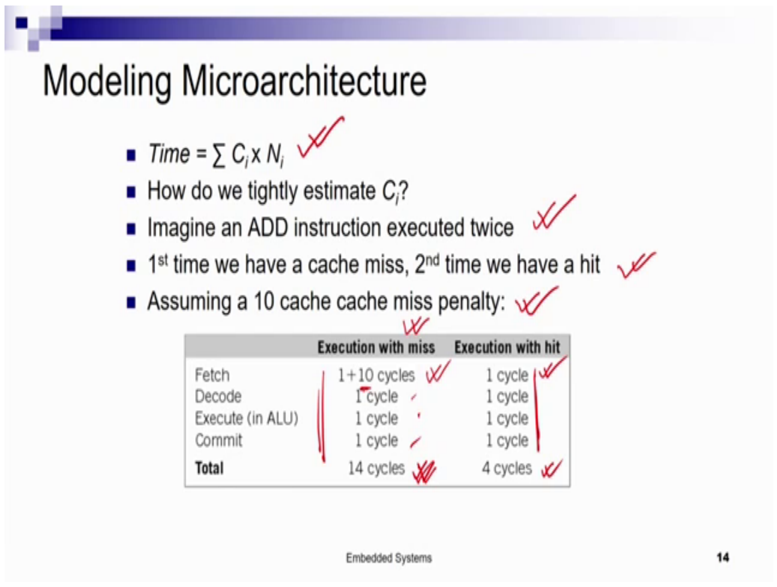
So, by doing this we can have even tighter estimates through infeasible path exploitation to take another example on infeasible path exploitation. Let us say that we have this program. So, while something if I greater than 0 then j equals to i else j equals to 1 minus i. So, when i greater than 0 j becomes equals to i. Now, if j less than 0 then k equals to i and j j equals to k on the other side i equals i plus 1 this it enters this loop.

So, we see that path 2 3 5 and 6 this is infeasible, why is this path infeasible this cannot happen, why cannot this path happen, because when i is greater than j we say j equals to i. So, j is also greater than 0 so, if j is greater than 0 the path for j less than 0 equals to true that cannot happen. So, this path can never be taken. So, we can say that this path 2 3 5 6 this is infeasible. This information may be captured in the ILP as follows, suppose we can say that N 3 plus N 5 is less than equal the loop bound, why because N 3 if N 3 executes N 6, if basic block 3 executes basic block 6, does not execute if basic block 6 execute basic block 3 does not execute.

So, therefore the if you have if you have a loop bound here. So, N 3 plus N 6 number of times basic block 3 execute plus the number of times basic block 6 execute this summation will be less than the loop bound. Another way to look at it is that through the flow constraints here that E E 2 3 E 2 3 equals to E 5 7. So, whenever this path is taken that i greater than 0 is true i greater than 0 is true you make j equals to i. So, therefore, j becomes greater than 0. So, whenever 2-3 is taken edge 5 7 will always be taken.

So, E 2 3 or the number of times edge 2 3 is taken is equal to the number of times edge 5 7 is taken. So, this is how we can take in feasible path into consideration and make tighter constraints. So, we have seen that how to obtain execution time estimates of tasks; however, if we do not take the micro-architecture or the platform into consideration such estimates may not become very realistic.

(Refer Slide Time: 48:06)



Modeling Microarchitecture

- Time = $\sum C_i \times N_i$ ✓
- How do we tightly estimate C_i?
- Imagine an ADD instruction executed twice ✓
- 1st time we have a cache miss, 2nd time we have a hit ✓
- Assuming a 10 cache cache miss penalty: ✓

	Execution with miss	Execution with hit
Fetch	1+10 cycles	1 cycle
Decode	1 cycle	1 cycle
Execute (in ALU)	1 cycle	1 cycle
Commit	1 cycle	1 cycle
Total	14 cycles	4 cycles

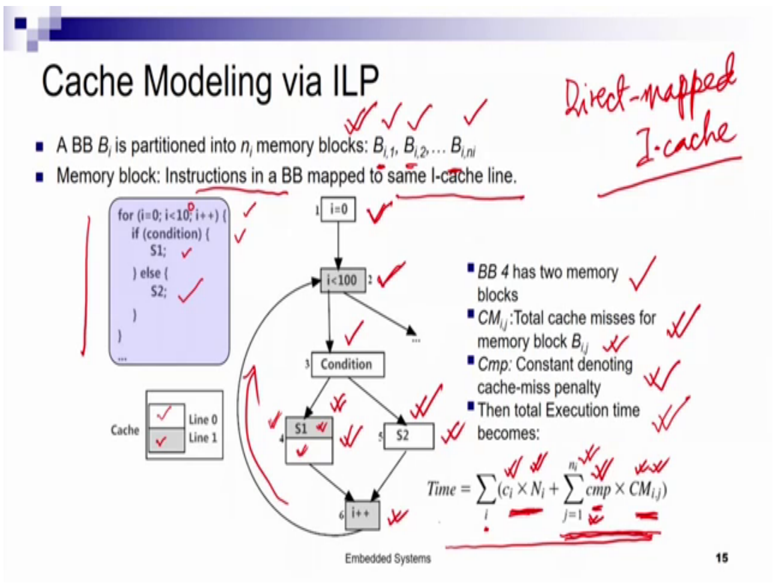
Embedded Systems 14

Now, how can we take micro-architecture and what is the effect if we do not take micro-architecture into consideration, let us take a very simple example. So, till now we have seen that the time taken by the program is summation C i into N i. Now, let us say that in this program there are two add instructions. And this instruction this same add instruction executes twice, the first time this add instruction is executed there is a cache miss that time it is brought into cache. And the second time again when this add instruction is executed, we have a cache hit. Now, assume that the cache miss penalty say the level one cache miss penalty is 10 cycles.

Now, for the fetch you need 11 cycles decode execute and commit you, if you have these stages of the pipeline if you have such a pipeline set of pipeline stages, then for that case in which there is a cache miss for the add instruction the total time taken will be 14 cycles 1 plus 10 cycles for the cache miss. So, 11 cycles for fetching the data for fetching sorry the instruction we are considering an instruction cache for simplicity here. And then it takes 11 11 cycles, but 3 cycles for the other three pipeline stages and the total time taken is 14 cycles.

Now, if we have a cache hit the same fetch takes only 1 cycle, because there is no cache miss and the overall instruction can be completed within 4 cycles. So, we see that for even a simple instruction, if we do not take micro-architecture into consideration we may have to consider an average for this cache miss penalty, or worst case and the execution time estimates have to be much more conservative. However, we if we can accurately model micro architecture although not always possible, we can we can have the opportunity of tighter estimates.

(Refer Slide Time: 50:06)



Cache Modeling via ILP

Direct-mapped I-cache

- A BB B_i is partitioned into n_i memory blocks: $B_{i,1}, B_{i,2}, \ldots B_{i,ni}$
- Memory block: Instructions in a BB mapped to same I-cache line.

```
for (i=0; i<100; i++) {
    if (condition) {
        S1;
    } else {
        S2;
    }
}
...
```

- *BB 4* has two memory blocks
- $CM_{i,j}$:Total cache misses for memory block $B_{i,j}$
- *Cmp:* Constant denoting cache-miss penalty
- Then total Execution time becomes:

$$Time = \sum_{i}(c_i \times N_i + \sum_{j=1}^{n_i} cmp \times CM_{i,j})$$

Embedded Systems 15

So, let us see how within our model we can within our ILP based strategy for execution time estimation how we can incorporate cache behaviour ok. So, let us say that a basic block B i is partitioned into N i memory blocks. So, we are considering here a simple instruction direct mapped instruction cache, we are considering direct mapped instruction cache yes. So, this we are doing for our simple case so, in a direct mapped instruction cache each memory each block in memory can be stored only in a single cache line. So, corresponding to each block it can be stored in all your single cache line as opposed to set associative caches and fully associative caches, where there can be multiple options for putting a block in memory into a cache line.

So, basic block B i is partitioned into N i memory blocks B i 1 B i 2 up to B i N i. So, these are parts of the basic block right so, a basic block has N i memory blocks. And a memory block is what instructions within the basic block that is mapped to the same eye cache line. So, therefore, I will require so, B i 1 B i 2 B i N i are memory block and each of them will be mapped to a single cache line.

Let us consider this simple program here for i equals to 0 i less than 10 high plus plus, if condition then a set of statements s one is executed else, another set of statements S 2 is executed. And for that our basic block is something like this i equals to 0 i less than i less than 100 or i less than 10 or something let us say this is 100 i less than 100, I have a condition if this condition is true, I go to basic block 4. Otherwise I go to basic block 5 we come to basic block 6 and this loop continues. So, basic block one equals to 0 i less than 10 i i less than 100 basic block 2 this condition is basic block 3 ok. And let us say we have only 2 cache lines the gray cache line 1 and the white cache line 0.

So, these are the two cache lines we have we also see that basic block 1 is mapped on to cache line 0. Basic block 2 is mapped on to cache line 1 basic block 3 is mapped on to cache line 0 basic block 4 has two parts. The first it has two memory blocks rather basic block 4, it is the only basic block here which has more than 1 memory block. So, memory block the first memory block of basic block 4 is in cache line 1. The second memory block of basic block 4 is in memory block is the second memory block of basic block 4 is in cache line 0 ok.

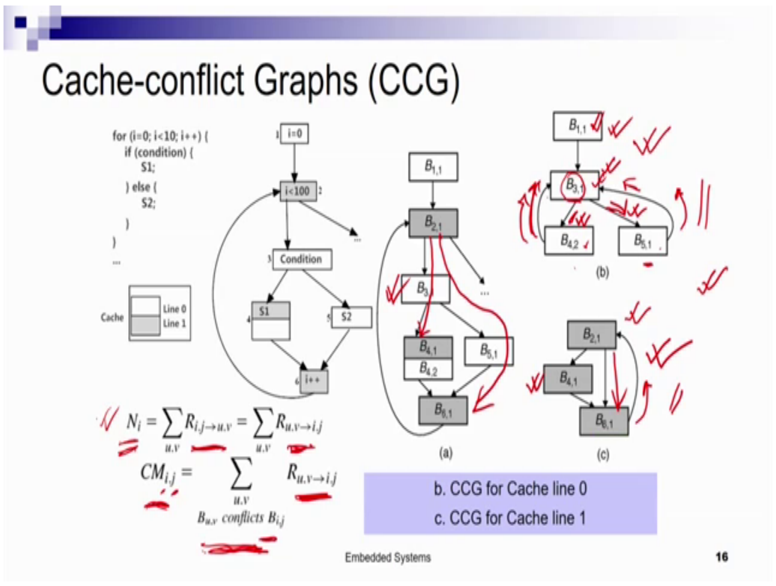
Again cache the basic block 5 is mapped only to cache line 0 and basic block 6 again is mapped only to cache line 1. So, we see here that basic block 4 has two memory blocks let CM i j be the total cache misses for memory block B i j ok, for memory block B i j CM i j is the total number of cache misses. What is CM i j, then CM i j is the number of cache misses for the jth memory block of basic block i of for B is B i j is the jth memory block of basic block i. So, CM i j is a cache misses with respect to the jth memory block of basic block i cmp.

Let cmp be a constant for simplicity Cmp be a constant denoting cache miss penalty. So, we are considering average cache miss penalty and for each such a cache miss the average penalty is CMP, then the total execution time now becomes what so, I had C i into N i previously, what is what was C i into N i what is i here i is basic block i for the ith basic block I have C i is the execution time of the basic block N i is the number of times that basic block is executed.

Now, each time the so, what is the then for this basic block, what will be the cache misses, the cache misses will be corresponding to each memory block within this basic block. So, this basic block i has N i memory blocks and for each such memory block, you have CM i j is the total number of cache misses CM i j is the total number of cache misses ok. Over all the times this basic block executes CM i is a total number of cache misses and for each cache miss the overhead in terms of time is cmp.

So, now the total time taken by the program will be given by this part which we have already seen. And this part due to cache misses for each basic block, for the i n basic block i go to j j goes from one to N i all the all the memory blocks, for each memory block how many cache misses CM i j is the total number of cache misses for each such cache miss the penalty is cmp and the total overhead for each memory block is cmp into CM i j. For the entire basic block its summation over j cmp into CM i j and that gets added to the time. Now, here we see that CMP is a constant the and therefore, CM i j becomes a variable. And therefore, this has to be constrained we need to get constraints bounds on CM i j.

(Refer Slide Time: 56:43)



Cache-conflict Graphs (CCG)

```
for (i=0; i<10; i++) {
   if (condition) {
      S1;
   } else {
      S2;
   }
}
...
```

$$N_i = \sum_{u,v} R_{i,j \to u,v} = \sum_{u,v} R_{u,v \to i,j}$$

$$CM_{i,j} = \sum_{\substack{u,v \\ B_{u,v} \text{ conflicts } B_{i,j}}} R_{u,v \to i,j}$$

b. CCG for Cache line 0

c. CCG for Cache line 1

Embedded Systems 16

So, this is done here this is done as follows, what we have to do to obtain a constraint on CM i j is to generate the cache conflict graphs, for each cache line ok. As the name says cache conflict graph says that what are the conflicting memory blocks for this cache, and this will be dependent on for a given cache line from where the so, which memory block replaces which block on this cache line, what is the what is the different who uses this who uses this cache line, what are the sequence of memory blocks which uses it this cache line to get that we need the cache conflict graph.

So, here we see that basic block 1 1 so, memory block 1, 1 from memory block 1, 1 on cache line 0, we are considering cache line 0, for cache line 0 first memory block 1 1 executes in cache line 0 after that memory block 3 1 executes in cache line 0. Then depending on control memory block 4 2 or memory block 5 1 can execute after execution of 5 1 control goes back to 3 1 so, so basic block 3 1 can be replaced so, basic block 5 1 can be replaced by basic block 3 1 on the cache line.

Similarly, basic block 4 2 can be replaced by memory block sorry basic block 4 2 sorry memory block 4 2 in cache line 0 can be replaced by memory block 3 1 on cache line 0. This is for the cache conflict graph corresponding to cache line 0. Similarly for cache line one what do we see firstly, memory block 2 1 is in cache line 1 ok. After that either basic block 4 1 will remain in cache line in cache line 1, if this path is taken if this path is taken or basic, or if this path is taken then cache then memory block 6 1 will replace basic will replace memory block 2 1. After basic block 6 1 completes execution then it will go back to basic block 2 1.

So, if the loop will go back and then on cache line 0 if currently basic block 6 1 is in cache line 0 that may be replaced by sorry, if memory block 6 1 is in cache line 0. After this loop goes back then basic sorry memory block 2 1 can replace memory block 6 1. So, this is what this cache conflict graph tells me. Now, what it also tells me that what is the number of times a certain basic block executes a certain basic block execute. So, N i tells me the number of times the basic block executes. So, number of times a basic block executes is equal to the number of times, you have control flows into the summation of the number of time control flows into a memory block in that basic block.

Suppose if you consider let us say basic block 3 and we are saying that basic block 3 has 1 a memory block B 3 1. So, basic block 3 has 1 memory block basic block 3 1. And we are saying that how many times this basic block can execute the number of times, there is a transfer of control on this cache conflict graph from this i j from a memory block of basic block i to somebody else. So, what we are saying is that what R i j tells me R i j tells me how many times does a memory block of basic block i go out to somewhere else for example, for this basic block 3 1 there are two edges from 3 1.

So, one is basic block sorry one is memory block 4 2, then the other is memory block 5 1. So, summation of the number of times these two edges are taken is equal to the number of times this basic block is executed. So, R i j 2u v tells me that for here R 3 1 2 R 4 2 and R 3 1 2 R 5 1. So, these are the two outgoing edges in this cache conflict graph, if you take the summation of this you will get the number of times this basic block executes ok. And similarly, it is also equal to the number of times that flow comes into this basic block. So, from other basic blocks say from B 5 1 and B 4 2 and B 1 1 the control flows into B 3 1. So, R u v u v in this case is 1 1 4 2 or 5 1 and R i j is B 3 1.

So, number of times there is an inflow into this memory block is equal to the number of time this basic block is executed. Now, when we want to find CM i j or the cache miss related to a certain memory block of a basic block. So, what is the number of times there will be a cache miss for memory block j of basic block i, it will be the number of times there will be a conflict, there will be a conflict between B u v and B i j. So, is equal to the number of times R u v 2 R i j occurs R u v 2 R i j occurs. And the number of times that this R R u g 2 R R u g 2 i j so, number of times this inflow is conflicting meaning that what do we mean by conflicting, that the contents of u v will be replaced by the contents of i j in the cache line.

So, for cache line 0 let us say we are saying that B 4 2 is conflicting with B 3 1, if B 3 1 replaces the contents of B i j, or rather the opposite B 4 2 replaces the contents of B 3 1. So, why do we want to say this the we want to say this, because whenever control comes back to be 3 1, if you have no trace of the previous contents of B 3 1, then there will be a lot of cache misses ok. So, when will that happen when B 4 2 or B 5 1 will replace the contents previous contents of B 3 1. So, again reiterating let us say we are considering cache line 0 in cache line 0 content currently you have contents of memory block B 3 1.

Now, after your control goes to B 4 2 and B 5 1 let us say the contents of cache line 0 is evicted by the contents of B 4 2 and B 5 1. So, the previous contents of B 3 1 no more remains in cache line 0. Now, what happens when I go back to B 3 1 the previous contents of B 3 1 are not there and hence you have a set of cache misses.

And therefore, summation of the number of times when B u v conflict B i j, the summation the number of times this edge is taken for the conflict you have CM i j, by using this we can have a bound on the cache misses that can happen ok. So, this is how we have given an overview as to how cache behaviour the can be modelled and incorporated within the ILP for estimating the worst case execution time of the program ok.

With this we come to the end of this lecture.