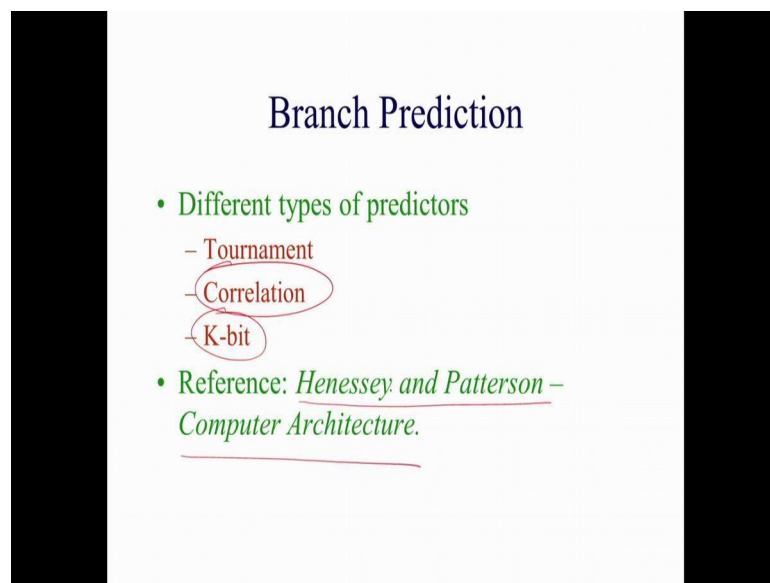


**Computer Organization and Architecture**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 25**  
**Structural Hazard, Architectural Enhancements**

(Refer Slide Time: 00:24)



**Branch Prediction**

- Different types of predictors
  - Tournament
  - Correlation
  - K-bit
- Reference: Henessey and Patterson – Computer Architecture.

So, we have seen different types of branch predictors, the tournament predictor the global predictor or what we now call it as a correlation predictor. It is also called a correlation predictor because we are trying to correlate the behaviour of one branch with some other branch and of course, you also saw the k bit predictor, which of was 1 bit, 2 bit, 3 bit and then we also mentioned that a k bit predictor k greater than 2 is as good as a two bit predictor. And so many of these predictors still they Byblos Hennessey and Patterson computer architecture book, and you will see this plus many more predictors you can search you will get close to half a million hit when you say branch predictors in google ok.

So, with this I will sign off on this and this is what I believe that you should know in some amount of detail of course, when you go to a advance course on computer architecture, you will learn how a tournament predictor can be implemented right. As we

see a tournament predictor is one which will move from depending on the behaviour of a branch and move from one to next to next to next and so I the branch predictor actually starts becoming intelligent right. So, that is a machine intelligence that is built there which will intelligently predict branches. So, there are very fundamental questions can and a tool which will enhance the intelligence of a branch predictor have branches inside it and so on. So, very interesting questions that we can try and handle. So, with this there is a scope for lot of intelligence to be built, lot more scope for writing papers. So, that is there we stop here now we move to the next type of hazards. So, we have seen data hazards we have seen control hazards, we have some remedies for each one of them correct and importantly do remember this zero cycle instruction. So, this is very very important now let us go to the last type of hazard in a pipeline, which we call as a structural hazard.

(Refer Slide Time: 02:29)

**Other Hazards**

- **Structural Hazards**
  - Non availability of a functional unit
  - Say would like to schedule the seventh instruction in our example
  - The new instruction has to wait.
  - Separate Integer, FPU and Load Store units are made available
- **Load-Store Architecture – What is it?**

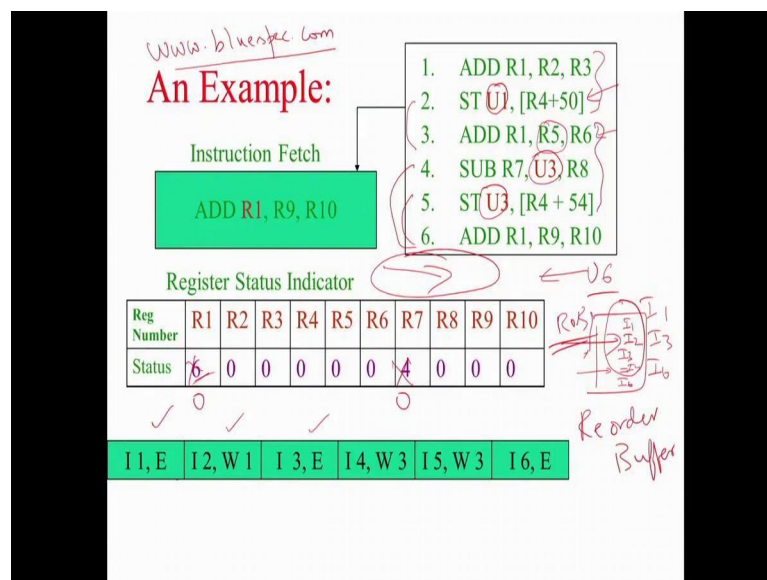
See the structural hazard is something like see we have some multiple functional unit, each of these units will not do everything right. There is no point in keeping say for example, there is no point in having a load store functionality inside an adder unit right. So, I cannot have one unit in which has both load store and adder or load store add and floating point right because if I have such type of a unit then I will be wasting lot of space and we can all we

cannot also selectingly go and disable parts of that unit in some

sense right there will be lot of dependencies.

So, what will happen is, I will not doing any floating point operations [FL] I will be wasting power I will be hanging around. So, I will put lot of area and also I will be wasting power there. Because if I have all the three in one execution unit and you have multiple such execution unit, we will say these are all symmetric execution units right then what will happen is that either I will be doing some floating point operation or I will be doing load store or I will be doing integer arithmetic not all the three in same unit. So, if when we look at multiple execution unit which forms the basis of course, the scalar architecture there will be some execution units will do integer arithmetic, some execution integer and logic arithmetic bit wise and bit wise and or sort of things, some will be dedicated to do floating point, some will be dedicated to do load and some will be dedicated to do store. Now if I get suppose I am getting those 6 instructions as we saw interrupt the previous example right.

(Refer Slide Time: 04:05)



We have seen this 6 instructions that we saw here. The 6 instructions are such that all are store or all are load for example, the worst case and I have only two load units the remaining 4 instructions have to wait.



This waiting is not because of a control issue, it is not because of any data issue it is because of the non availability of the functional unit. It is because the structure of the architecture does not have enough units to do this computation, and that is why we call this as the structural hazard. So, what it means to build this architecture there are lot of questions here to answer; how many such integer instructions I will get at any point of time? If I am going to get always say 5 to 6 integer instructions which can execute in parallel, then to actually get the biggest speed up I need to I will put 6 integer execution units right. If I am only going to get 3 or 4 then the remaining two I will not put if I by that I will reduce my power consumption etc. So, today when somebody wants to buy your architecture, first thing they will say how fast is your architecture second thing they will also come and see how much power it is going to spend, it is going to consume, how much area it is going to consume, how much heat it is going to generate all these things also now start coming up when people try to build your buy your architecture specifically in a high performance computing type of environment.

We are looking all these super scalar more from an performance point of view. So, what should be see in this case as we see here we have actually assumed that all these units are symmetric, all these units can execute every instruction, but that will be ideally the case right and that is where we land up with what we call as the structural hazard. I will just read out the red statements here, non availability of a functional unit will cost the structural hazard say I would like to schedule the seventh instruction in our example now I have only 6 units that is also one of the symptoms. The new instruction has to wait and then ideally we will have separate integer floating point unit and load store units and so, in a in a load store architecture of course, you know what a load store architecture is. Now when there are lot of load store instructions then it will not there can be a potential functional hazard.

Now the next type right what it means to do compiler fine tuning for an architecture right? Fine tune the program for your architecture it will see that for this architecture there are only 4 integer units right. So, it will if there are two floating point instructions far off right. So, I have say let us say I have 4 integer instructions then suppose I have lot of integer instructions and some two floating point instructions here. So, I say I 1, I 2, I 3, I 4, I 5, I 6, I 7, I 8, I 9 and these two are floating point instructions there I 8 and I 9

are dependent on each other of course, these are all I 1 to I 4 are independent, I 5 to I 7 are independent let us say or I 1 to I.

So, suppose there are 4 integer units and one floating point unit in your architecture, what I can do is I can push this I 8 here and send this together because then this 4 will be fetched along with I 8 and. So, right rather this I 5 going and it has no units and it will start waiting. When it starts waiting everything else has to wait in the program header because the does not the computer the system does not know that you know because all these things all the instructions are waiting, now all the subsequent instruction has to wait. You cannot now go and schedule I 5, I 6, I 7 etc because there is no functional unit there is no reservation station right. So, if at all somebody has to wait has to wait on the reservation station.

So, I 8 has no I 5 there is no reservation station because there are 4 integer units and I 5 is also an integer instructions, but there is a floating point in it. So, what the compiler can do is it can push this I 8 along with this. So, that when the hardware fetches it will fetch this 4 integer instruction and one floating point instruction and that can start executing on this in the next step it can bring three integer instruction along with this floating point instruction correct otherwise what will happen I 1 to I 4 one has to finish then I 5, then I 6 then I 7 then I 8 alone will go then I 9 will go because there only floating point in them are you able to understand this.

So, by doing this type of you know execution by doing this type of what we call as a code motion, where I move the code from one like move an instruction from part to the another by doing this rearrangement the compiler can ensure that the finally, this is the dynamically scheduled process, but it can effectively use this understanding. So, even for a dynamic dynamically scheduled super scalar processes, there are some (Refer Time: 09:27) the compiler can do in order to improve the performance ok.

So, this is what we understand by structural hazard, we will talk about one more structural hazard as we move down with memory management right at that point of time, but as far as execution goes this is what we mean by structural hazard, and this is what the compiler can do to handle structural hazards. Now with this this is all about super



scalar pipelining stuff and all that we have discussed I now over. Now what we will do is now we will start talking about more on deep more deeper into the architecture we will I will cover this Amdahl's law which is very very important and then we will start talking about multi core architectures that is what we will do till we touch. Quiz two post quiz two will look at cache memory and organisation cache and I o. So, any doubts in pipelining super scalar shall I proceed yes [FL] can we go, can we go ahead.

Now comes to this Amdahl's law, see Amdahl's law is something that an architecture it is a tool for the computer architect say I want to do some enhancement, I am not satisfied with the performance, I want to do some enhancement to improve my performance right. Now I need some guidance for doing this enhancement and that is what precisely Amdahl's law basically talk about see. So, Amdahl's law basically quantify something called speed up, we are only talking about performance here no area no power. We are talking about performance here. Amdahl's law talks about what is speed up speed up is defined as the execution time without enhancement divided by the execution time with enhancement the time should decrease.

(Refer Slide Time: 11:31)

SPEC  
**Architectural Enhancements**

**Amdahl's Law**

$$\text{Speedup(Overall)} = \frac{\text{Exec time without Enhancement}}{\text{Exec time with Enhancement}}$$

$A$  = Fraction of computation time in the original architecture that can be converted to take advantage of enhancement.

$$\text{Exec\_time(New)} = (1 - A) \text{Exec\_time(old)} + \text{Exec\_time of Enhanced portion} \cdot (1)$$

$\frac{0.6 \times 10^5}{60} + 2 = 55$

$\frac{10}{2} = 5$   
 $\frac{10}{2} = 5$   
0.4

So, the speed up should always be greater than one correct. So, the speed up this

parameter is defined as execution time without enhancement, divided by execution time

with enhancement. The execution time without enhancement whatever enhancement that we are talking of will be less than the will be greater than the execution time with enhancement hopefully greater than the execution time with enhancement, and hence we will get a better speed up.

So, Amdahl's law helps you quantify the speed up right. I go and say I want to do this enhancement then you will ask what would be the speed up right that is a even (Refer Time: 12:04) you will ask, oh go and do this what will be the speed up how fast will the computer run in contrast or in comparison to what it is currently what would be the enhancement in speed and that is what we are talking of. So, this is the. So, we will be trying to get some very good expression for speed up. Now what will happen in an enhancement, when I enhance speed up how do you compute this speed up? We run benchmark programs and we compute the speed up. We ran benchmark programs on a model without enhancement we get some total number of cycles, we now run the same thing with enhancement we will get a new total number of cycles, we divide one by another and that is what we call as speed up.

Now when I want to do this enhancement it will not be at the entire program gets every instruction of that program starts working faster. For example, my enhancement would be on the floating point so; that means, the floating point instructions alone will start working faster right. So, when I do an enhancement in the architecture, it is not that every part of the program gets enhanced a fraction of a program will get enhanced. For example, in my executable if I have say 10 floating point instructions out 100 say 10 percent, and I improve the floating point performance; that means, those 10 percent or 0.1 fraction will improve right are you able to are you able to comprehend this right. So, see every book just gives this law and proves it, but you should see what is the justification of having such a law. See there will there will be several ways by which I can estimate speed up why should I estimate it in this fashion why should I have these type of a that all these wise are not answered. So, please note it down very carefully.

A actually stands this a actually stands for the fraction of computation time in the original architecture that can be converted to take advantage of this enhancement. Fraction of computation time, there are 10 percent of the time why this definition it is not

just not the fraction of the instructions; fraction of the instruction will not make any sense right I would have only three instructions, but the three instructions will run one million times. So, in a. So, there is that 10, 20, 10, 8, 10, 90 rule right 90 percent of the code will execute only for 10 percent of the time, 10 percent of the code will execute for ninety percent of the time. So, if suppose I have 3 floating point instructions and 10 integer easy 9 integer instructions. So, totally 12 instructions. So, what is 2 out of 12 25 percent if I go and improve the floating point performance I say that 25 percent there will be improvement is a wrong statement. Suppose these three floating point instruction this 12 instruction execute for say 100 units of time and this floating point instruction itself consumes 90 percent of the time right. If I improve the floating point performance the performance of your code will increase much more than the 25 percent ok.

Suppose these three floating point is to instruction only consume you know 3 units of time out of 100 and I go and improve the floating point and nothing will happen because the majority of the time you spend on still your untouched instructions. So, when I am trying to model this a, I should it is it is more appropriate for me to model a as a fraction of computation time rather than a fraction of the number of instructions are you able to get this any doubts are you able to follow? Why should a be fraction of computation time and not fraction of instructions as I just told a bit earlier are you able to get the feel yes or no.

So, what is the new execution time? Execution time of the new code is nothing, but one minus k times the execution time of old because a percent of that code is only enhanced. So, what, so the remaining 1 minus a portion will remain the same. So, 1 minus k times execution time old right plus the execution time of the enhancement right because a percent of the time is now enhanced and that is accounted for and that is accounted for a percent of the code is enhanced a percent of that execution time is enhanced and that is accounted for here while the remaining one minus a is still the old code. So, that is one minus a times execution.

So, if say 0.4. So, totally I am taking say some 10 seconds right and 0.4 percent of your 0.4 percent of your program is now enhanced, remaining 0.6 into 10 seconds the remaining 0.4 means that 4 seconds of my running time is now improved, the remaining

6 seconds will remain 6 seconds and that is what this 0.6 into 10, that 4 seconds 0.4 of this 10 second is improved to say 2 seconds. So, the total time now becomes 8 seconds it is are you able to follow. So, the new execution time if I say 0.4 the original old execution time was 10 seconds this is 10, 0.4 percent 0.4 is the fraction of computation time which is taking the advantage; that means, 4 seconds there will be some instructions executing for 4 seconds, that those instructions are now enhanced the remaining 6 seconds will remain 6 because the remaining 0.6 has to remain.

So, that 0.6 into 10 this 6 seconds is accounted for here, now the remaining 4 seconds for which I have done the enhancement that 4 seconds has now become 2 seconds. So, the new execution time will be 8 seconds. So, the speed up now will be 10 by 8, what is 10 by 8 1.25 correct is it.

Student: Sir does this not depend on the program execution time without enhancement upon with enhancement, different programs is instruction can be different right.

Yeah yeah that is always.

Student: So.

So, we are only talking about some benchmark programs. So, see we cannot actually measure for all the programs on here. So, that is why that benchmarks have come. So, we will run all these benchmark for example, and for that we will get the speed up. So, speed up it will be respect to some benchmark set up benchmark. On a whole these benchmark performance is improved. So, I can go and sell it in the high performance computing server market when I want to go and compute in the server market people will only look at this type of execution time. So, I can go and sell it in the server market if my thing (Refer Time: 19:16).

So, we will do this for one class of benchmarks spec benchmarks I have already asked you to go and look at benchmarks right. So, this is any doubt. So, let us keep this mind execution time new is 2 minus 8 times execution old plus execution time of the enhanced portion, and overall speed up is execution time without enhancement divided by



execution time enhancement and A is the fraction of computation time in the original architecture that can be converted to take advantage of that enhancement ok.

(Refer Slide Time: 19:50)

$$\begin{aligned} \text{Speedup(enhanced)} &= \frac{\text{Exec\_time of enhanced portion(old)}}{\text{Exec\_time of enhanced portion(new)}} \\ &= \frac{A * \text{Exec time (old)}}{\text{Exec\_time of enhanced portion(new)}} \\ \text{Exec\_time of enhanced portion(new)} &= \frac{A * \text{Exec time (old)}}{\text{Speedup(enhanced)}} \end{aligned}$$

Substituting in (1) above we get

Now, what is speed up enhanced, speed up of the enhanced portion of your speed up of the enhanced portion of your code. The speed up of the enhanced portion of your code is nothing, but the execution time of the enhanced portion old divided by execution time of enhanced portion new correct the speed up of the enhanced portion alone is nothing, but execution time of the enhanced portion old divided by execution time of the enhanced portion.

The execution time of enhanced portion old is nothing, but A into execution time old because A percent of the time is what I am getting the enhancement for. So, execution time of enhanced portion old is nothing, but A into execution time of old. So, the execution time of enhanced portion new is nothing, but A into execution time old divided by speed up enhanced. Substituting back in one what I am going to substitute this execution time of enhanced portion. So, the execution time of enhanced portion is A into execution time old by divided by speed up. So, this is. So, the execution time new will be 1 minus A execution time old plus A times execution time old by speed up enhanced.





(Refer Slide Time: 21:09)

Final form of Amdahl's Law

$$\text{Exec\_time(new)} = \text{Exec\_time(old)} * \left( (1-A) + \frac{A}{\text{Speedup(Enhanced)}} \right)$$
$$\text{Speedup Overall} = \frac{1}{\left( (1-A) + \frac{A}{\text{Speedup(Enhanced)}} \right)}$$

So, execution time new is equal to 1 minus A times execution time old, 1 minus A times execution time old plus A times execution time old by speed up enhanced.

So, what is overall speed up? Execution time new divided by execution time old is nothing, but sorry execution time old divided by execution time new is nothing 1 by 1 minus A plus A by speed up enhanced. Just take this please take this derivation this is a very very simple derivation of Amdahl's law, please copy this and then we will go and find out why this take why am I interested in this one have you all copied.

Now can you tell me palgart and I madras can you tell me why do I need a formula like this why I want to calculate over all speed up, but that over all speed up has something called a and something called speed up enhanced and nothing more I could have had other things right I could have had execution time execution time of enhanced portion, there are so many variables that we have seen why am I interested just in a c speed up enhanced. Why I am why should I get a formula like this and not something else. I could have got different verities of formulas right why am I interested in this form speed up overall is equal to 1 by 1 minus A plus A divided by speed up enhanced (Refer Time: 22:35) .



Student: We change something all the overall performance changes.

No that is what we are trying to calculate and in the process of calculation I am just landing up with this form, this is the formula that I want to I am using now for this right this is the formula I am using why in this for what why not in something else I could have derived other ways of this speed up overall we have n different forms.

Student: It will compare which portion of the.

Why this form.

Student: If the data is already available for (Refer Time: 23:20).

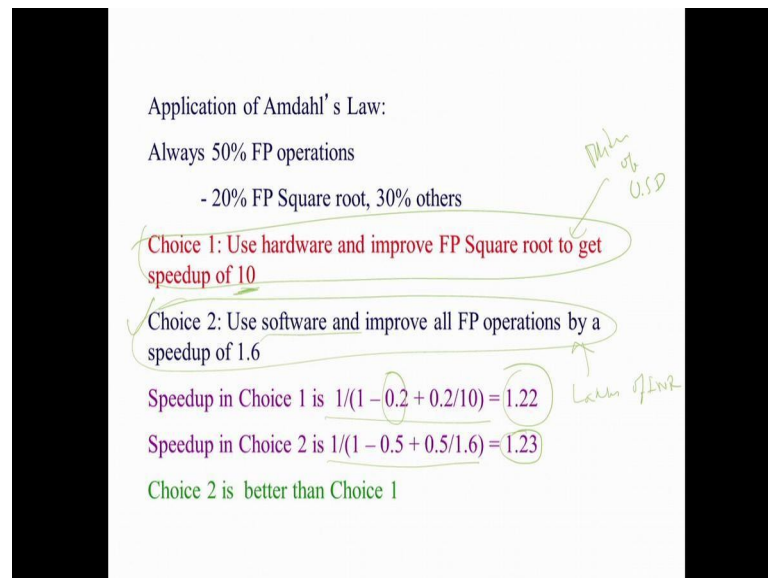
Availability of data excellent see a is available to me right, because I can run the benchmark when I run the benchmark there is something called the execution trace right when I run the benchmark I will know every instruction how many times it is executed and when I do my enhancement I know which are all the instructions I am touching from that I can go and find out and I can I know for every instruction how many cycles it will take I know how much total cycle the program took from that I can actually compute a accurately correct. Then next thing, I go and improve some performance speed of some part I can really find out what is the speed up there for that part alone. So, my speed up enhanced and a can be accurately computed that is why I would love to express my speed up overall on in terms of A and speed up enhanced because these are the things that I could compute with lot more accuracy.

I have a model I know I can run the entire spec benchmark on instruction set stimulator and actually I can go and find out every instruction how many times it is executed, and I know what is the total length of this program total number of cycles consumed not the length, this is a number of cycles consumed by this program and when I go and touch the enhancement I know which are all instructions that are going to get affected from that I can basically get this a is the fraction of the computation time, that is affected by this. So, I can get this a very (Refer Time: 24:52). Similarly I could also get the speed up enhanced because I work on that model. So, I know the previous model now I have go



and improve improved on that model, now for me to get speed up enhanced are now very easy. So, by this I can basically find out whether there will be speed up or not.

(Refer Slide Time: 25:13)



Application of Amdahl's Law:

Always 50% FP operations

- 20% FP Square root, 30% others

Choice 1: Use hardware and improve FP Square root to get speedup of 10

Choice 2: Use software and improve all FP operations by a speedup of 1.6

Speedup in Choice 1 is  $1/(1 - 0.2 + 0.2/10) = 1.22$

Speedup in Choice 2 is  $1/(1 - 0.5 + 0.5/1.6) = 1.23$

Choice 2 is better than Choice 1

*Handwritten notes:* "Ratio of 0.5" with an arrow pointing to "Always 50% FP operations"; "Less of 0.5" with an arrow pointing to the denominator of the Choice 2 formula.

So, let us take some examples before we end up today's class. So, let us take a program which always takes 50 percent FP operations in which 20 percent are FP square root and 30 percent are others. So, I am taking a benchmark scenario where always any program I will have 50 percent of floating point operations. Let us assume is a highly scientific computer and in that 50 percent of FP operations 20 percent of it will be square root, and 30 percent will be some other instructions. Now this is the scenario this is the program that is given to you the assembly program that is given to you and you have to now start executing an architecture arrive at an architecture which will handle these classes of assembly program very nicely. So, you are given two choices, use a hardware and improve floating point square root to get speed up of 10 right this floating point square root currently is implemented in the instruction set. Now can I have can I go and improve the floating point square root to get a speed up of 10 is the question.

Suppose there is a way I put it as choice one, choice two is use software and improve all floating point operation by a speed up of 1.6 do not bother about all these floating

point hardware we will give software which will go and improve all the floating point

operations by a speed up of 1.6. So, what is this speed up in choice.  $1 \div (1 - 0.2 + 0.2 \div 10)$  right because 20 percent this floating point square root. So, in a over all program the a is 0. right. So, so  $1 \div (1 - 0.2 + 0.2 \div 10)$ . So, this is because I need a speed up of enhanced of 10. So, this is. So, the speed up in choice one is 1.22, interestingly my speed up in choice two is 1.23. So, if I really want to enhance then I will prefer choice two not only that it is not a hardware it can be done in your lab you know it is just software and compilation, but interestingly you will find that the software solution gives you much better much much better almost equivalent, but slightly more better speed up than the pure hardware connection. So, just to wind up.

So, we had given two choices one choice was that I will go and improve 20 percent of this square root 20 percent of my run time I will improve, but then the speed up would be 10 times. The other was that 50 percent of the time I will improve, but the speed up for by a speed up of 1.6 for that portion alone. Now when we see that the choice two becomes much more favourable than choice one, and choice two is also important because I can go and do everything in software I need not touch the hardware. So, this is much more simpler we will talk in lacks of rupees while here I will while lacks of Indian rupees while here I am talking of millions of u s d. So, that is the difference right. So, please understand.

So, Amdahl's law if you just look at it to start it initially if I had ask this question before teaching Amdahl's law I say floating point square root takes lot of time I will get a speed up of 10 it is taking 120 cycles and I will make it 12 cycles, something like that if I say then you would have actually gone and fall in fall in pray for this choice one, but then if I use choice two I save lot of money and I also get better performance at least point not one more than this. If the program runs for 10 hours right 0.01 essentially is a very significant time if the program runs for 10 days some simulations run 10 you know 0.01 is you know some 2.5 hours. So, it will. So, it actually means quite a large amount of time in terms of.

(Refer Slide Time: 29:22)

### Shared Memory Architectures

- Sharing one memory space among several processors.
- Maintaining coherence among several copies of a data item.

We will now go in the next class we will now go in to the multi core. So, whatever I have talked so far is about a single core. So, now, we need to enhance it for multiple cores and the systems that you are going to use are multi core system. So, whatever you have learnt here we will become extremely useless if you do not understand the multi core architecture all the super scalar will be one core like that many things will be put and what it means to arrive at that bigger architecture and that we need to understand because we do not find uni core machines anymore everything is multi core your phone has 8 cores or will have 8 cores currently it will have 4 core and it and it will have 8 core.

So, even when you want to understand how your mobile phone works somebody can ask you this question how your phone works you should be in a position to answer those questions. So, we will talk about. So, the next joke comes in what we call as the cache coherency protocol. So, now, we will teach some rudimentary simple, but working protocol then people have started working on it some 100 protocols have come just handling cache. So, but we will see some interesting things in that some real salient things that we will cover ok.



Thanks.