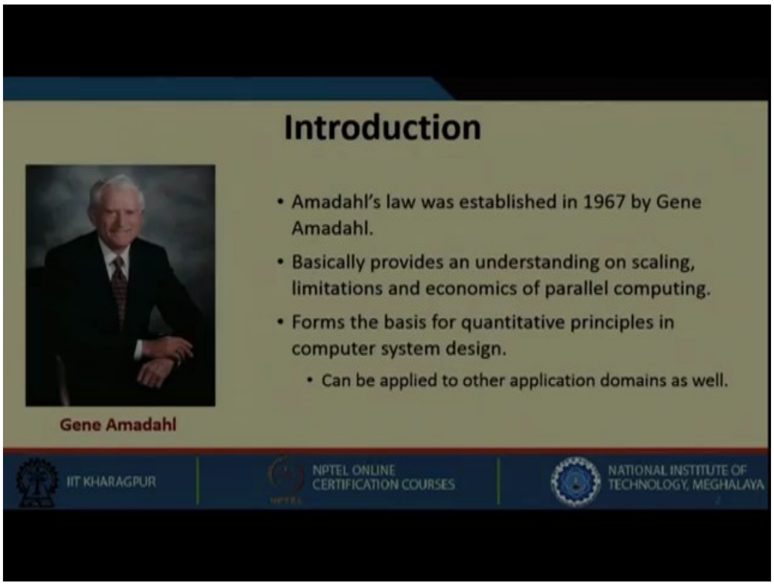
Computer Architecture and Organization
Prof. Kamalika Datta
Department of Computer Science and Engineering
National Institute of Technology, Meghalaya

Lecture – 15
Amadahl's Law (Part I)

Welcome to the 15th Lecture on Amadahl's Law. So, till previous lecture what we have seen that how we can summarize all the results, and then come up with a particular solution. Now we will see another law that will tell us little more about what we have learnt.

(Refer Slide Time: 00:46)

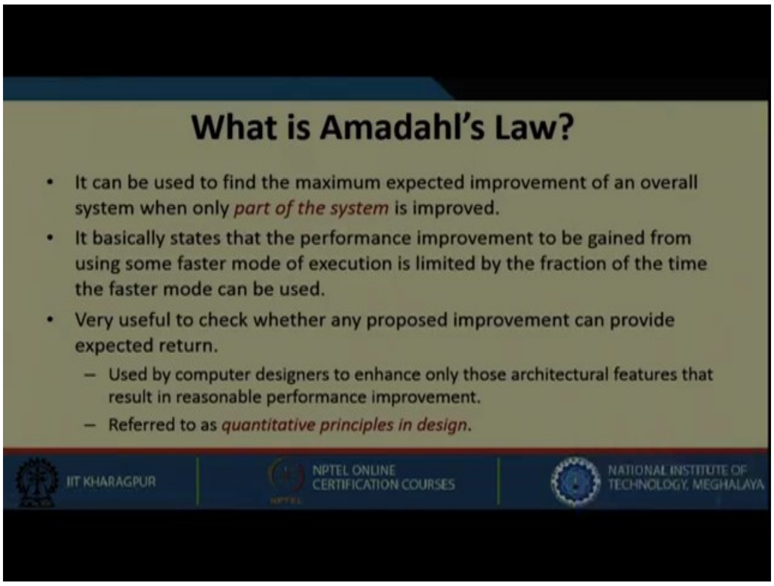


Amadahl's law was established in 1967 by Gene Amadahl. And what it gives is that it provides an understanding on scaling limitation and economics of parallel computing. So, by scaling limitation we can understand that the transistor size is becoming smaller and smaller. So, it gives an understanding on scaling limitation and also on the economics of parallel computing.

By economics of parallel computing what we mean is that --- suppose today we have quad core, we have multi core technologies. So, when you are investing or when you are having so many cores, are you getting the return; when you are having just one core how much was your performance, by having more number of cores are you really getting that advantage that you are supposed to get? So, it also gives us the economics of parallel computing.

And of course, it forms the basis for quantitative principles in computer system design. We can actually in a quantitative fashion tell about the design principles of a computer system. And Amadahl's law is not only for computer system design, it can be applied to other applications as well and for other domains.

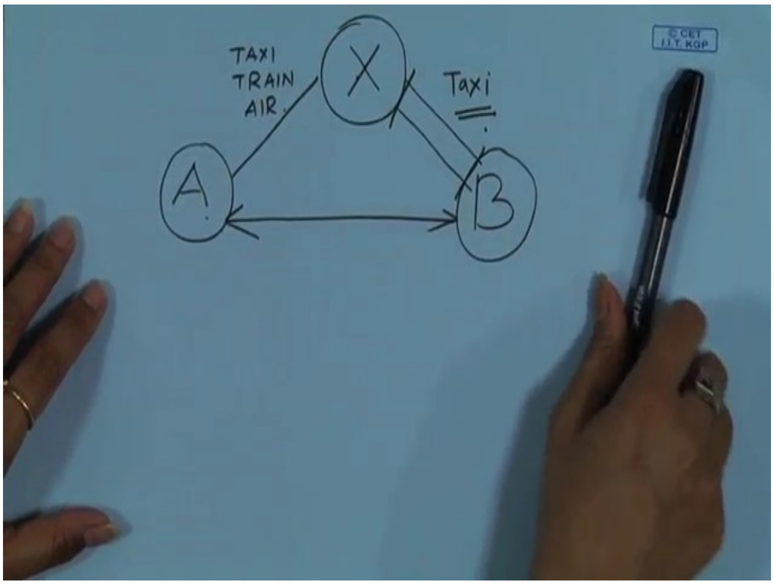
(Refer Slide Time: 02:30)



Let us see what exactly is Amadahl's law. It is basically used to find the maximum expected improvement of an overall system when only a part of the system is improved. What do you mean by this? Suppose we have full overall system, but maybe part of it let us say only 40% of that part we can make an improvement. So, we want to see that by making 40% of the part improved how we can get an overall improvement.

So, I will just give you a small example. Let us see you want to travel from place A to place B.

(Refer Slide Time: 03:28)



So, I want to travel from A to B, and you have a middle point because you have to travel through this point, you have to reach this point and from this point again you have to take. Now from this part to this part let us say for travelling you can travel through taxi and there is no other way. But for travelling from A to X you can use many modes: one is taxi, one is train, another is by air. So, what you can basically do is that you cannot do anything for this part, this part is limited by the speed of this taxi only, but all we can do is that we can improve this part that is from A to X by either using taxi, using train, or by air.

So, this is where we will have to see that how can we optimize, we need to reach from A to B through X, and then only I can improve this part not this part this part. But by improving this particular part how much improvement we are getting out of it that we want to see. So, that is where Amadahls' law comes into picture.

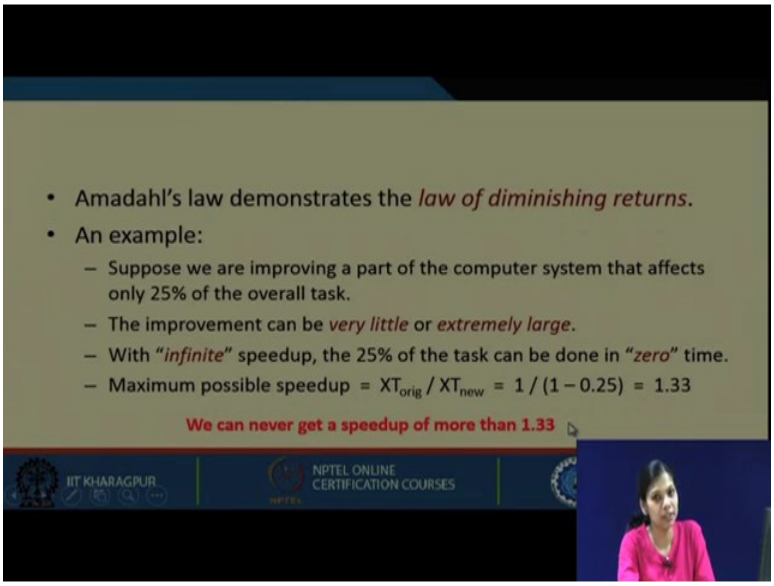
It can be used to find the maximum expected improvement of an overall system. We will see the improvement of an overall system when only a part of which is improved, because another part we cannot improve. It basically states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of time the faster mode can be used. So, these again come into in terms of execution of some instructions. So, by this what we mean is that as we are saying that this we are actually improving a fraction of the part, and fraction of that particular part can be used more, than the overall improvement can be further increased.

So, very useful to check whether any proposed improvement can provide expected return or not; that is what I am saying. That let us say I put some improvement on that particular part where I can improve, but by putting improvement on that particular part will you actually get that return that you are expecting, or you will not be able to get that return, or how much return you will be able to get. So, all these things can be determined through Amadahl's law.

And it is used by computer designers to enhance those architectural features that result in reasonable performance improvement. And this is referred to as quantitative principles in design. So, again by this what we mean? Let us say we have an adder, a multiplier and in general purpose computing we may say we are using adder more than multiplier. So, in such cases what can happen? If you try to improve the multiplier that you are not using much, then it will not give you that particular kind of improvement.

So, that is also what you have to look into. That we are improving a part that is also used more number of times. The number of times it is used is more and we are improving that particular part, so where we can get overall improvement much more.

(Refer Slide Time: 07:32)

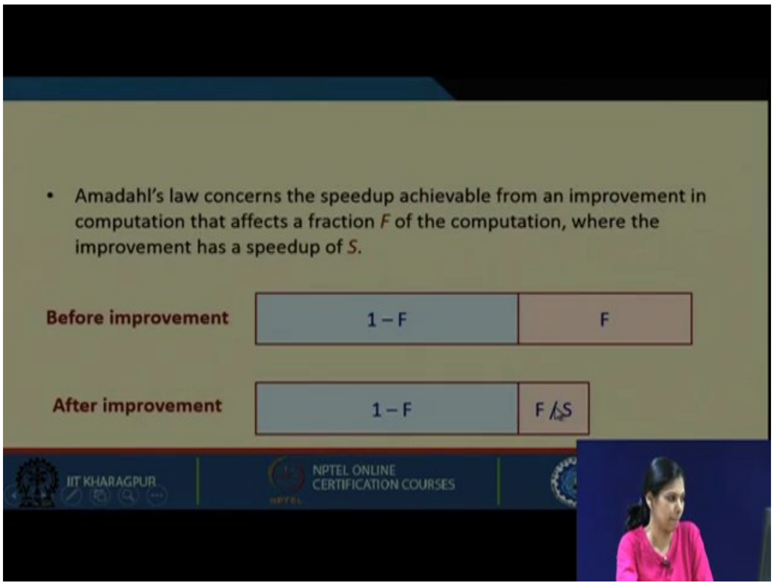


Amadahl's law demonstrates the law of diminishing return. So, what is that? Let us take an example; suppose we are improving a part of the computer system that affects only 25% of the overall task. So, out of 100% only 25% of the work can be improved. The improvement can be very little or extremely large, let us say how. With infinite speed up, only 25% of the part can be improved; and we are saying that if we make that 25% part task can be done in 0 time maximum, that much improvement can be done that it will take no time to finish that 25% of the task, but we cannot do anything with the rest 75% of the task.

75% of the task remains, but only that 25% of the task we are making some improvement. And what kind of improvement, we are saying that we are keeping that time 0. So, in no time that particular part of task can be performed. So, maximum possible speed up can be XT_{orig} / XT_{new}. So, if the original execution time is 1, what will be the new execution time; out of total 100% of the task 25% can be done in new time. So, you can just take out that 0.25 which is coming to 1.33.

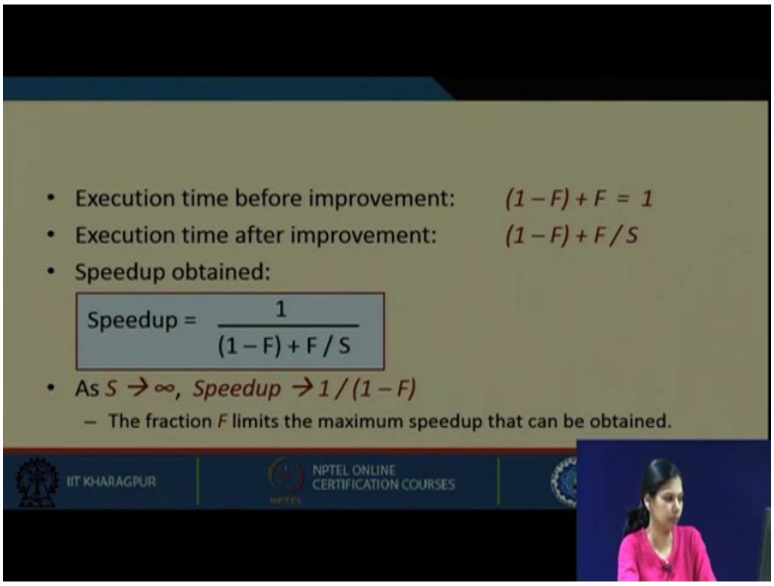
So, we can have a maximum of this much improvement. If you can improve that 25% and we can say that the 25% of the task can be done in new time. So, we can never get a speed up more than 1.33, even if you do whatever you want we cannot get an overall improvement because we can only make that 25% faster, because 75% still is working at the previous speed.

(Refer Slide Time: 09:46)



So, Amadahl's law actually concerns the speedup achievable from an improvement in computation that affects a fraction F of the computation, where the improvement has a speed up of S. So, on this fraction F we are having an improvement of S. So, this is the F portion, and this is 1-F. So, 1-F there will be no change, and we can only have a speed up of S to this F portion. So, if you can reduce this so it will become F / S. So, earlier it was taking this much time and now it is taking this much time. So, after improvement we can execute the same task by this percentage.

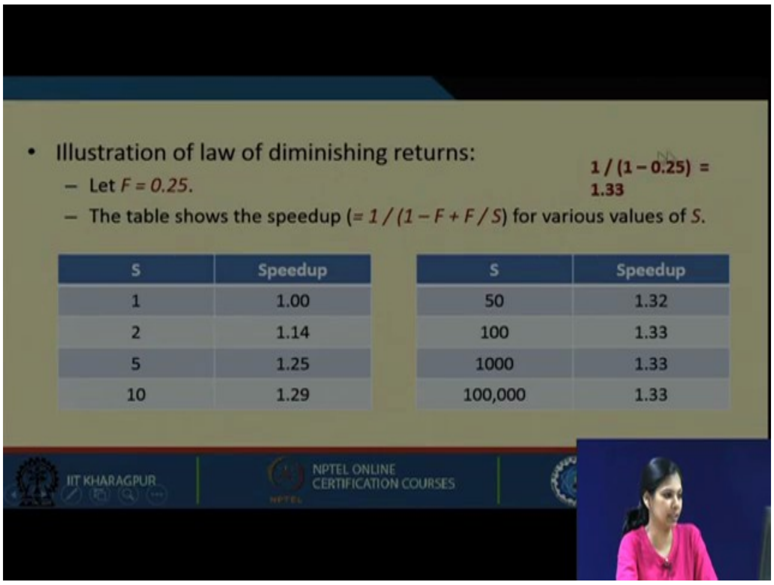
(Refer Slide Time: 10:45)



So, let us find out how we can find out the overall speed up. So, execution time before the improvement is 1. This was fractional part and this is 1-F, basically this is 1. And what is the execution time after improvement we have seen; 1-F will remain as it is, and we have to make improvement on F. So, it will become F / S. So, speed up will be XT_{old} / XT_{new}. XT_{new} is (1 – F) + F / S; this is what we have got.

So, as S tends to infinity, speed up will be this. If you make this part 0, so what will happen? The speed up will be 1 / (1 – F). So, the fraction F actually limits the maximum speedup that can be obtained.

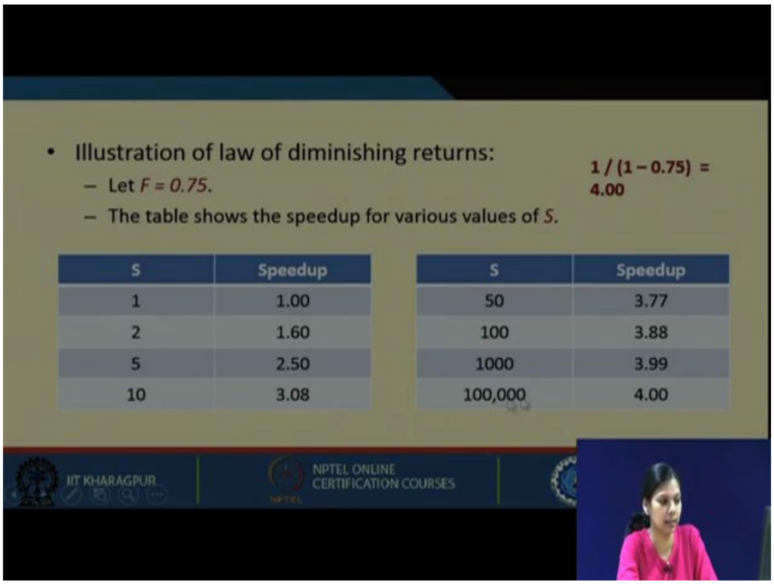
(Refer Slide Time: 12:12)



So, let us illustrate this law of diminishing return for F = 0.25 by changing the value of S. So, speed up is 1 / (1 - F + F / S) for various values of S we have computed. So, let us say when S is 1 we are getting as total speed up of 1.00, when it is 2 we are getting 1.14, when it is 5 it is 1.25, when it is 10 it is 1.29. So, the difference if you see it is basically reducing. This difference was more, this is less, this is less, this and this is even less.

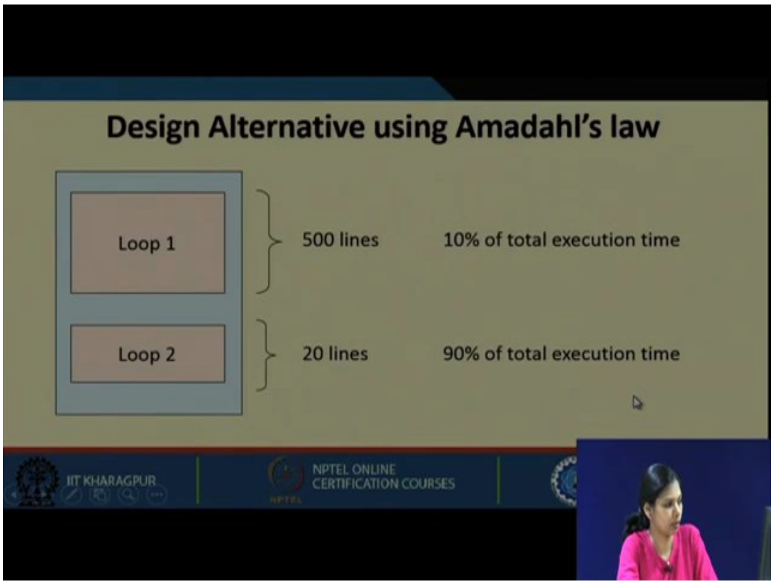
And what we can see is that after certain amount of time that is after certain speedup we are not getting any further speedup. The speed up is limited by a factor of 1.33, because we have already said we can get a maximum speed up of 1.33; that is what we are getting, even if you are increasing this fractional speed up by any amount. So, unnecessarily it is no point increasing this speedup as there is a limit to it, because the maximum speedup that can be achievable is 1.33.

(Refer Slide Time: 13:51)



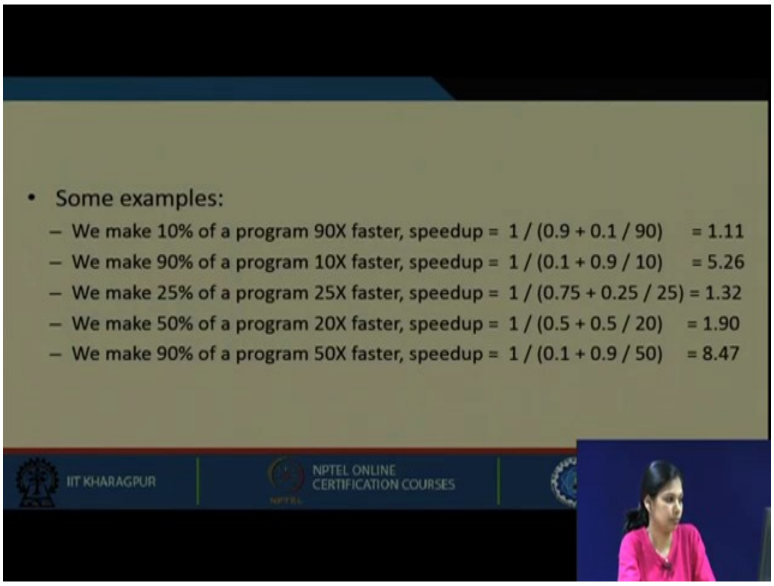
So, let us take another example where F = 0.75. This table also shows the speed up for various values of S; as we increase S what it depends on how it affects overall speed up. So, again we see that when S is 100000, then the speed up is 4. And the maximum achievable speed up when the fraction is 0.75 is also 4. So, even if you increase this S, this will remain the same.

(Refer Slide Time: 14:38)



So, these are some design alternatives using Amadahl's law; what we are saying that let us say we have a portion of a program this is loop 1 and this is loop 2. This loop is of 500 lines and this loop is of 20 lines. But this 500 lines of code takes 10% of the total execution time. What do we mean by that, is this takes certain amount of time to execute, but this particular loop is only taking 10% of that total time. And this Loop 2 which is only 20 lines of code, but it is taking 90% of the total execution time. Although this is small portion, but that small portion is taking more amount of time; so we need to also take into consideration this design alternative.

(Refer Slide Time: 15:46)



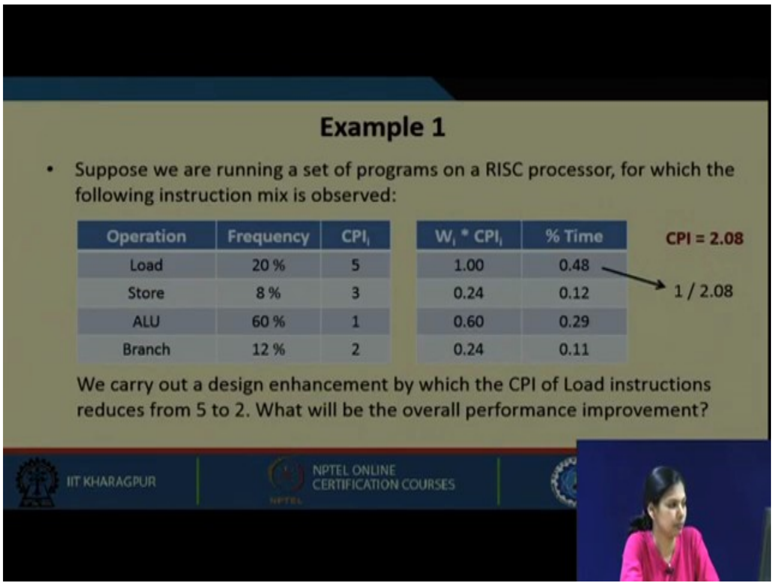
So, let us see if you make some improvement on this particular case, where only 10% of the total execution time is used for Loop 1, and 90% of the total execution time is used for Loop 2. So, let us say we make 10% of the program 90 times faster, so how much speed up we will get? So, this is 10% of the program you are making 90 times speed up. So, 1 divided by this is (1 – F) and this is F / S, where F is 0.1 and S is 90, because we are making 90% speed up of on this. And what we are getting? We are getting a value 1.11, the overall speed up.

Now we make 90% of the program 10 times faster only, but that portion is 90%. So, 1 divided by this is (1 – F) and this is F / 10 and we are getting 5.26 which is much more, but we are only making a speedup of 10 times. So, making 10 times speed up of a portion that is used most amount of time, which is taking more amount of time that is 90% it is taking that will giving a better result.

Similarly make 25% of the program 25 times faster. So, 25% we are making 25 time faster and this remains (1 – F) is 0.7 we are getting this much. Similarly, 50% of the program making 20% faster so this is giving 1.90. And if we make 90% of the program 50 times faster we are getting a value where we are getting maximum, because this is 90% of your program we are making a speedup of 50%. So, here we are getting a value of 8.47.

So, this is how you can see that if you are making improvement on a part of your program that is used more number of time that will give a better speedup.

(Refer Slide Time: 18:27)

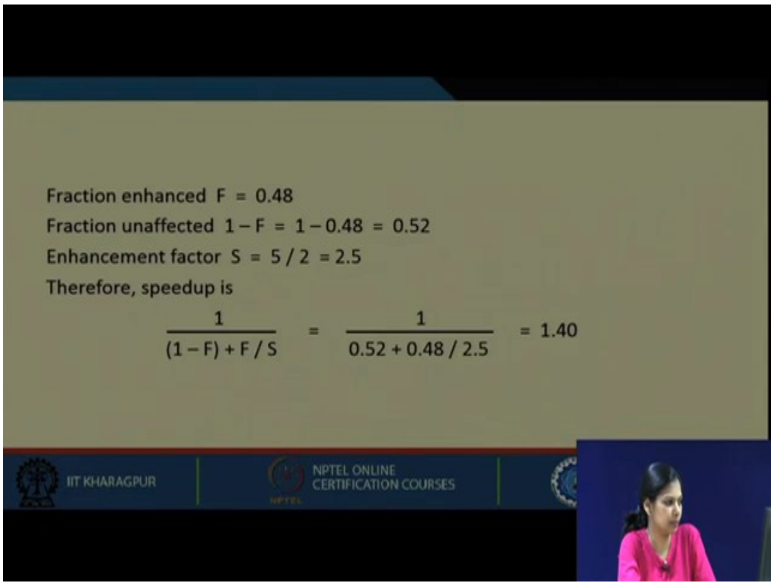


Let us take another example. Suppose we are running a set of programs on a RISC processor for which the following instruction mix is observed. So, we are having load, store, ALU and branch operations. And this is the frequency at which load instructions are taking place; this is the frequency at which store instruction is taking place, and so on. And this is the CPI for load, store, ALU and branch. Now, let us find out W_i * CPI. So, we multiply this into 0.2 we get this, this into 0.08 we get this, and so on.

Now percentage time it is used can be found out by total CPI; total CPI will be 2.08. So, if total CPI is 2.08, we will divide 1 / 2.08 to get this. So, what we do this is the given thing and we have found out W_i * CPI and also the percentage of time it is used. Now we carry out some kind of enhancement and now we will see what is the improvement that we can get.

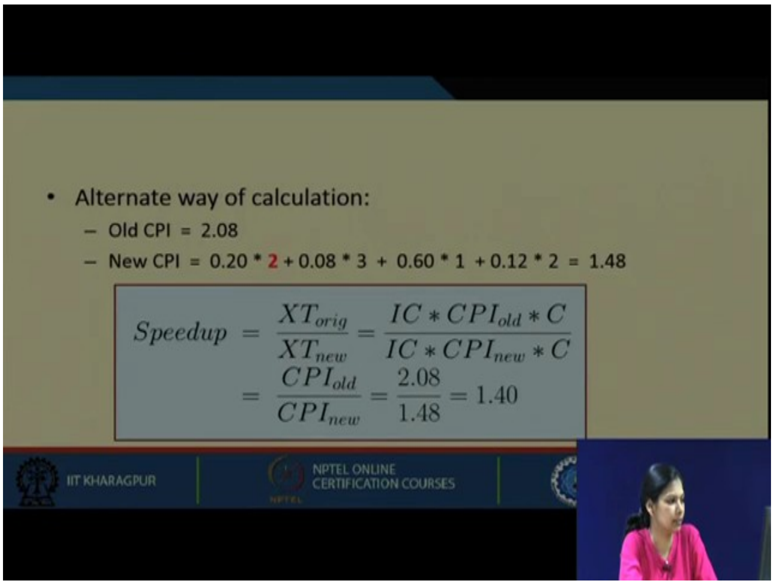
So, we carry out a design enhancement by which the CPI of load instruction reduces from 5 to 3. So, earlier this load instruction was taking 4 CPI, now it will take 2 CPI. So, what will be the overall performance?

(Refer Slide Time: 20:55)



Now, we want to see the overall performance that we will get. So, how do we get it? So, fraction enhanced F is 0.48. So, we can see that we are reducing from 5 to 3. So, fraction which we can actually enhance is 0.48 and unaffected portion will be 0.52, that is (1 – F). So, if F is this and (1 – F) is 0.52, what will be S? Earlier it was 5 now it has become 2. So, this speedup will be old divided by new, which is coming to 2.5. So, if we just put it in this value we get 1.40.

(Refer Slide Time: 21:59)

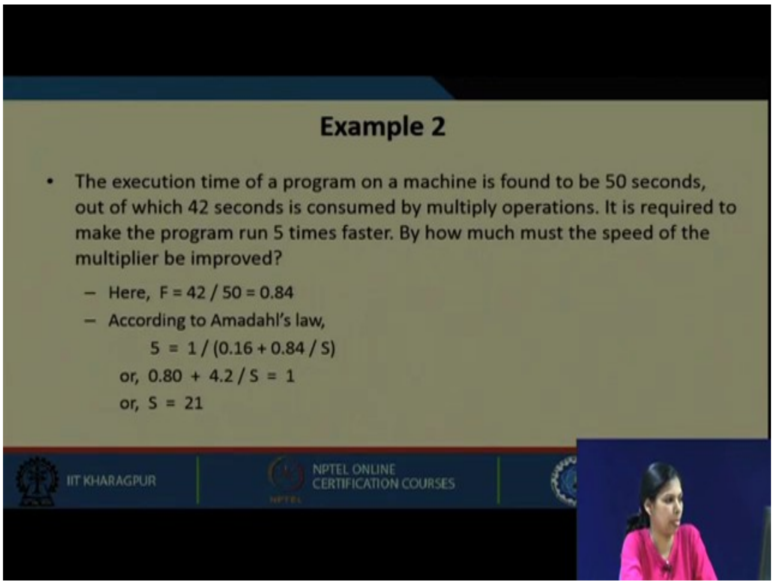


Therefore, the speed up will be 1.40 with that enhancement. Similarly we can solve this using the other way; what is the other alternative? Let us just go back to two slides and figure out we have found out the CPI that is 2.08. Now we will make an enhancement and make this as 2, and then we will calculate the CPI once more. So, let us see how to do this.

So, old CPI was 2.08 and the new CPI we can find as 1.48. Earlier it was 2.08 when for the load instruction the CPI was 5. Now we have made some improvement and made the CPI = 2 and we have figured out the new CPI is 1.48. So, with this, what will be the speedup?

So, the formula of execution time if we recall from our previous lecture is instruction count x cycles per instruction. So, we are getting 1.40. So, either way if we do it we are getting 1.40. So, these are the two alternative ways through which you can perform this.

(Refer Slide Time: 23:59)

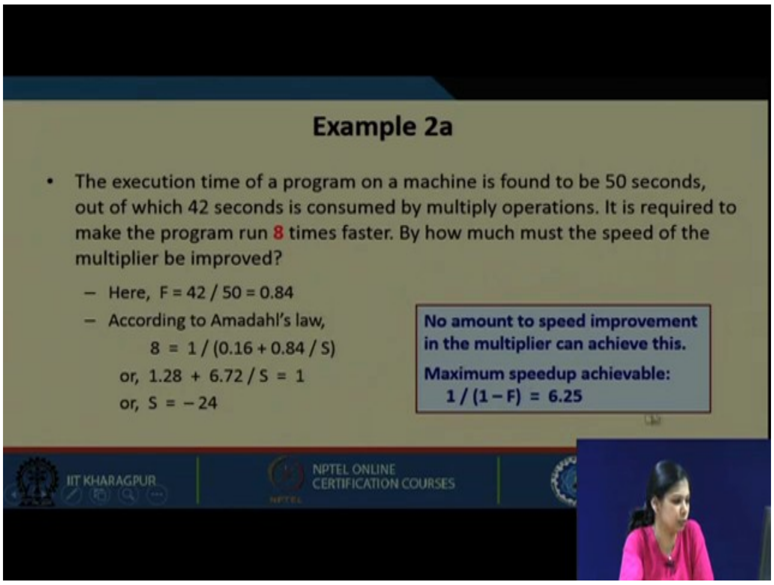


Let us take another example. In this example what we are saying the execution time of a program on a machine is found to be 50 seconds, out of which 42 seconds is consumed by multiply operation. So, most of the time multiply operation is taking more time. It is required to make the program run 5 times faster.

So, the overall speedup of the entire system you want to make it 5 times by how much speedup of the multiplier, by how much must the speed of the multiplier be improved. So, let us say it has been given that you want this 5 times speedup. So, how much this multiplier part can be improved? So, we will just put that in the formula F will be 42 / 50 that is 0.84, and according to Amadahl's law speedup is 5 will be equal to 1 / (1 – F).

So, I am doing dividing that by S because we need to find out this how much speedup on the multiplier you have to do such that the overall speed up is 5. So, if you solve this equation you will get S as 21. So, you have to make a speedup of 21 times to the multiplier; so as to get an overall speed up of 5. So, this is how this problem can be solved.

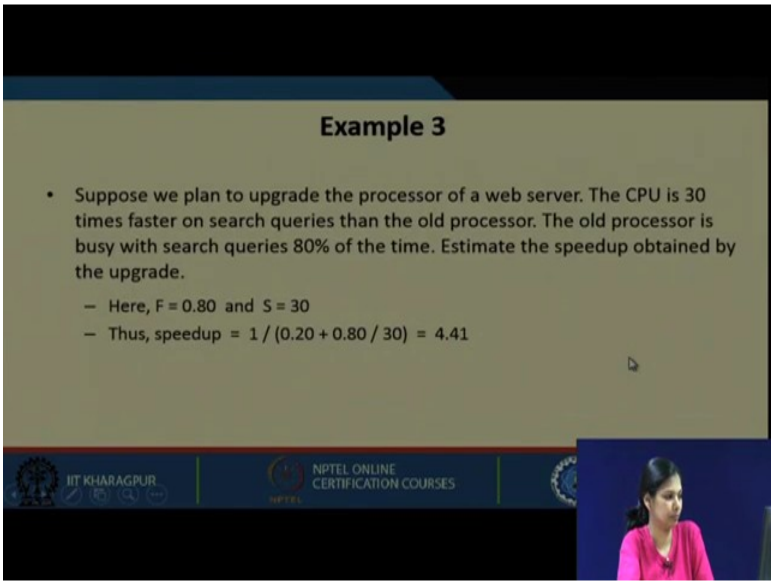
(Refer Slide Time: 26:05)



Let us take another example. It is the same example, but I want to make it more fast. The execution time of the program and the machine is found to be 50 seconds out of which 42 seconds is consumed by multiply operation it is required to make the program 8 times faster. So, similar way earlier it was 5 now it is 8, I put it 8 and I try to solve it. And what I get? I am getting a negative value. Why I am getting a negative value? We need to see that can we at all make 8 times faster, because there is a limitation of something that how much part the enhancement can be performed.

So, in this case we can see that 1 / (1 – F) is 6.25. So, the maximum achievable speed up is 6.25 and you are saying you want to get 8; that is not possible. So, in this case you cannot have a speed up of more than this achievable speedup; as we have already seen that earlier. So, no amount of speed improvement in the multiplier can achieve this.

(Refer Slide Time: 27:40)



Let us take another example. Suppose we plan to upgrade the processor of a web server. The CPU is 30 times faster on search queries than the old processor. So, the CPU is 30 times faster on the search queries than the old processor and the old processor is busy with search queries 80% of the time. So, the old processor was always busy 80% of the time with the search queries only. Estimate the speed up obtained by this upgrade.

So, we are making an upgrade of 30% on the search queries and the old processor was using 80% of the time that search queries. So, F will be 0.80 and (1 – F) will be 0.20. So, you substitute in this particular equation in the speedup and you get 4.14. So, with this upgrade we can get the speed up of 4.14.

So, in this lecture we have seen what is Amadahl's law and we have also seen that how the maximum speedup achievable can be found through this Amadahl law. And we have also seen that while designing certain system, you have to take in to consideration that if the speedup is not achieved on the entire system, it can be only made on the part of the system; we need to analyze that based on the part of the system how much overall speedup you can get. So, Amadahl's law actually states that on the part where you are making improvement how much maximum speedup you can make, that is the overall speedup you can achieve.

Thank you.