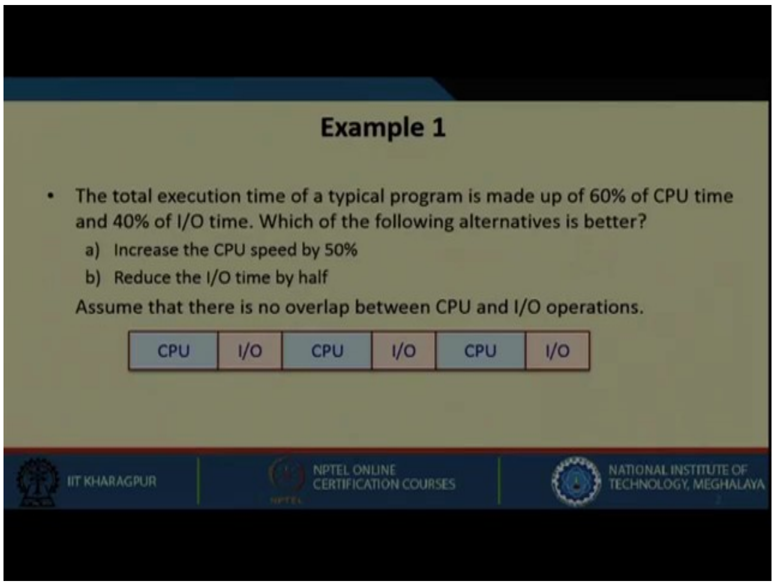
Computer Architecture and Organization

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

Lecture – 16
Amdahl's Law (Part II)

(Refer Slide Time: 00:25)



Example 1

- The total execution time of a typical program is made up of 60% of CPU time and 40% of I/O time. Which of the following alternatives is better?
  a) Increase the CPU speed by 50%
  b) Reduce the I/O time by half

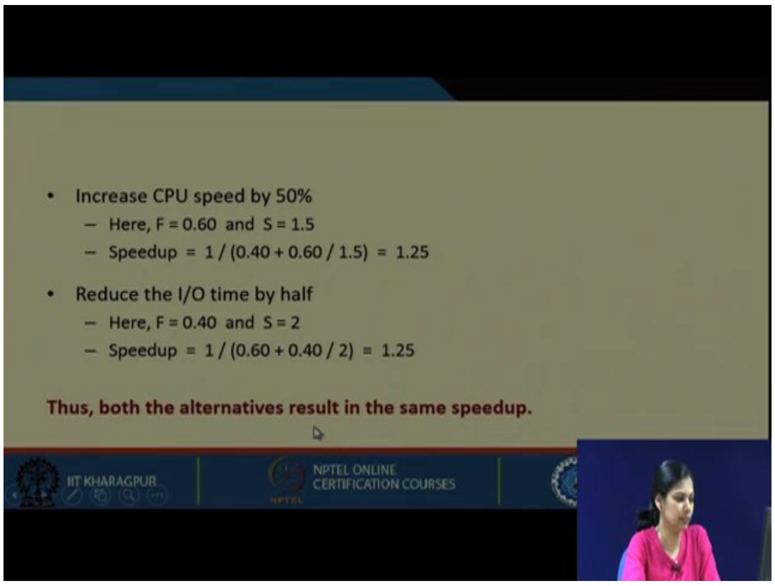
  Assume that there is no overlap between CPU and I/O operations.

CPU
I/O
CPU
I/O
CPU
I/O
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Welcome to lecture 16 where we will continue with Amdahl's law little more. Let us start with an example where we see that, the total execution time of a typical program is made up of 60% of the CPU time, and 40% of the IO time. So, out of total number of tasks we are dividing into 2 types. One is, CPU bound job another is IO bound job. So, CPU bound job is taking 60% of CPU time and another is taking IO bound is taking 40%.

So, let us see this overall thing as CPU I/O CPU I/O. And we assume that there is no overlap between CPU and IO operation; that means, when CPU is used, only CPU bound job will be taken care of. When IO bound job is getting performed, only IO bound jobs will be taking care of and so on. So, we want to see that, in this particular scenario, there are 2 alternatives that we are trying to improve the performance. So, we want to see which one is better. So, first one is increase the CPU speed by 50%. And in another case we are reducing the IO time by half; that means; when you are increasing the CPU speed by 50% we are not doing anything on this 40%, and when we are reducing the IO time by half, we are not doing anything on this 60%. So, let us see this.

(Refer Slide Time: 02:22)



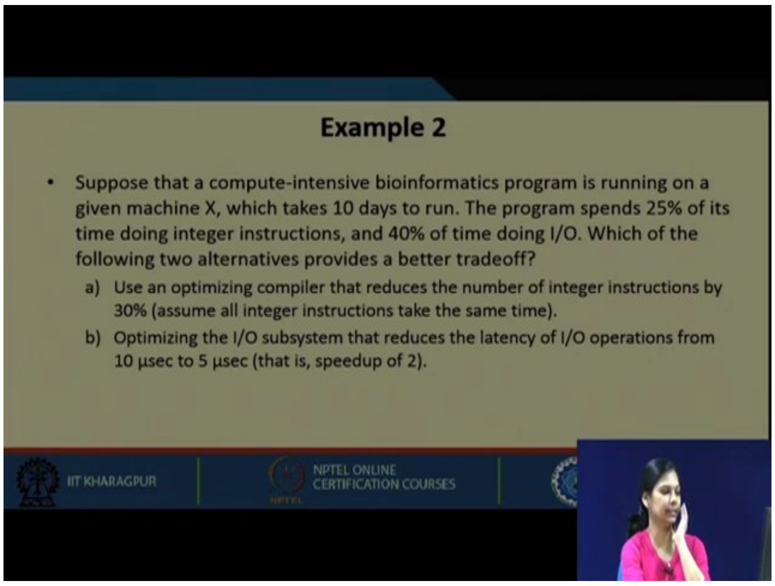
- Increase CPU speed by 50%
  - Here, F = 0.60 and S = 1.5
  - Speedup = 1 / (0.40 + 0.60 / 1.5) = 1.25
- Reduce the I/O time by half
  - Here, F = 0.40 and S = 2
  - Speedup = 1 / (0.60 + 0.40 / 2) = 1.25

Thus, both the alternatives result in the same speedup.

Firstly, increase CPU speed by 50%. So, here F clearly is 0.60. And S will be 1.5. And so overall speedup will be 1.25.

Similarly, we say that we reduce the IO time by half. So, as you are reducing IO time by half; that means the speedup you are increasing by a factor of 2. So, in this case F is 0.40 and S is 2. You put it in the formula of speedup and we are getting the same result. So, both alternatives result in the same speed up, as what I said just.

(Refer Slide Time: 04:04)



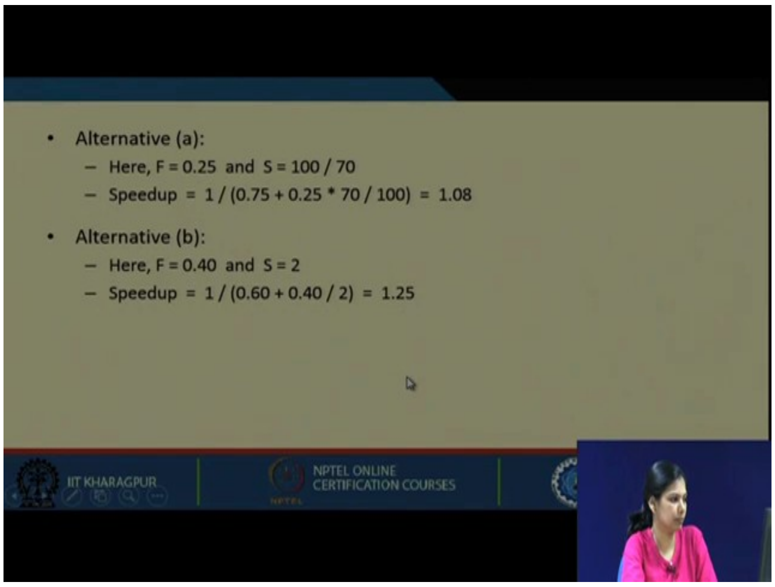
Example 2

- Suppose that a compute-intensive bioinformatics program is running on a given machine X, which takes 10 days to run. The program spends 25% of its time doing integer instructions, and 40% of time doing I/O. Which of the following two alternatives provides a better tradeoff?
  a) Use an optimizing compiler that reduces the number of integer instructions by 30% (assume all integer instructions take the same time).
  b) Optimizing the I/O subsystem that reduces the latency of I/O operations from 10 μsec to 5 μsec (that is, speedup of 2).

Now, let us take another example. Suppose that compute intensive bioinformatics program is running on a given machine X that takes 10 days to run. So, it is compute intensive. Lot of computation is going on for this program. The program spends 25% of it is time doing integer instructions, and 40% of the time doing IO.

So, 25% of the time is spent on doing integer, and 40% of the time doing IO. Which of the following 2 alternatives provides a better tradeoff? The first one is, use an optimization compiler that reduces the number of integer instructions by 30% assume all instruction take the same time. And the next one is we are optimizing the IO subsystem, that reduces the IO operations from 10 micro second to 5 micro second. That is again we are saying a speedup of 2. Let us see the 2 alternatives.

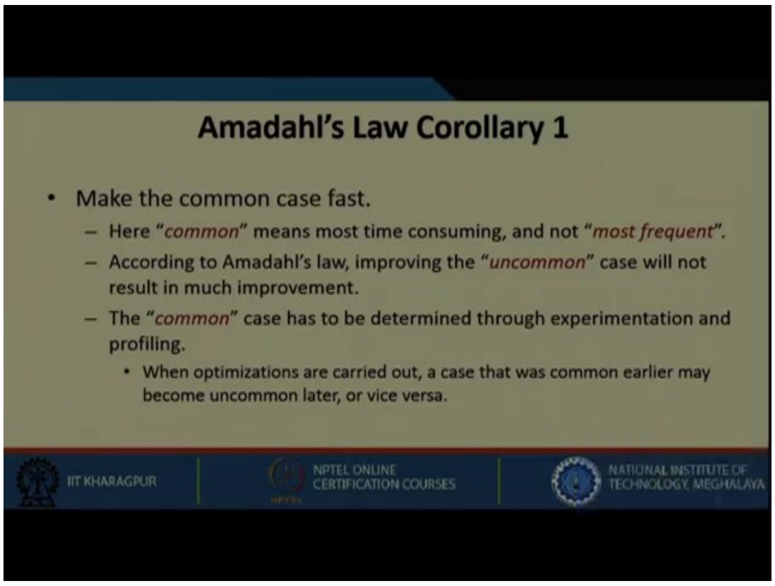
(Refer Slide Time: 05:44)


- Alternative (a):
  - Here, F = 0.25 and S = 100 / 70
  - Speedup = 1 / (0.75 + 0.25 * 70 / 100) = 1.08
- Alternative (b):
  - Here, F = 0.40 and S = 2
  - Speedup = 1 / (0.60 + 0.40 / 2) = 1.25

So, in the first case F is 25%, and S is we are reducing that by 70 because 30% is reduced. And the program spends 25% of the time in integer operation. So, S becomes 100 / 70. So, 0.75 if there is no change on that. And 0.25 we are making a change, and we are getting a speedup of 1.08.

Let us take the next alternative where IO bound job is 40%. And the rest 60% we are not doing anything. But on that 40% we are having a speedup of 2. So, in this case if we solve for the speedup equation, we are getting 1.25. So, in this case we can say that, alternative B is better than alternative A, where here the speedup on 25% is made and here the speedup on 40% is made.

(Refer Slide Time: 07:10)



Amadahl's Law Corollary 1

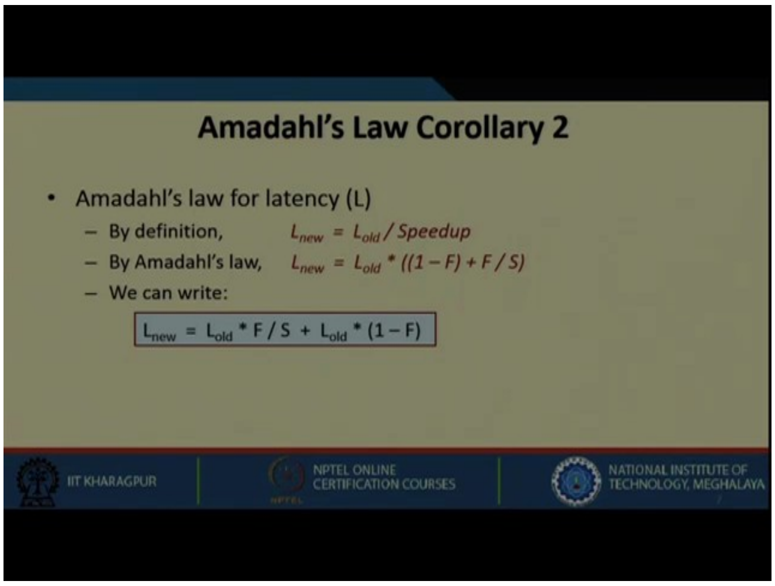
- Make the common case fast.
  - Here "*common*" means most time consuming, and not "*most frequent*".
  - According to Amadahl's law, improving the "*uncommon*" case will not result in much improvement.
  - The "*common*" case has to be determined through experimentation and profiling.
    - When optimizations are carried out, a case that was common earlier may become uncommon later, or vice versa.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us move on with the Amdahl's law Corollary 1. What it says is, make the common case fast. What do you mean by common case? By common case we mean that most time consuming and not most frequent. So, let us understand this. There are 2 things. One is when we say that we are more frequently doing something. But, more frequently with which we are doing it is taking less time. But the most common means it is consuming the most amount of time. So, it is most time consuming. According to Amdahl's law, improving the uncommon case will not result in much improvement. Rather if we make improvement on the common case, the improvement will be much more; that means, the portion of the task that is taking more time we will try to improve on that part. Rather than which is more frequently used because most frequently that use that is not taking much time.

So, the common case has been determined through experimentation and profiling. So, how we can say that this part is common, through experimentation and through profiling. When optimizations are carried out, a case that was common earlier may become uncommon later or vice versa. So, you need to analyze the program. You need to analyze it time and again to understand this. So, when we are saying common part, common part means most time consuming part. So, you have made some improvement and you have reduced that time. And then again if you can finally say that now once this common part I have reduced, next time when you do that some it might change. So, this particular phenomenon says that when optimizations are carried out, a case that was common earlier may became uncommon later because you have already taken care of that particular case.

(Refer Slide Time: 09:47)



Amadahl's Law Corollary 2

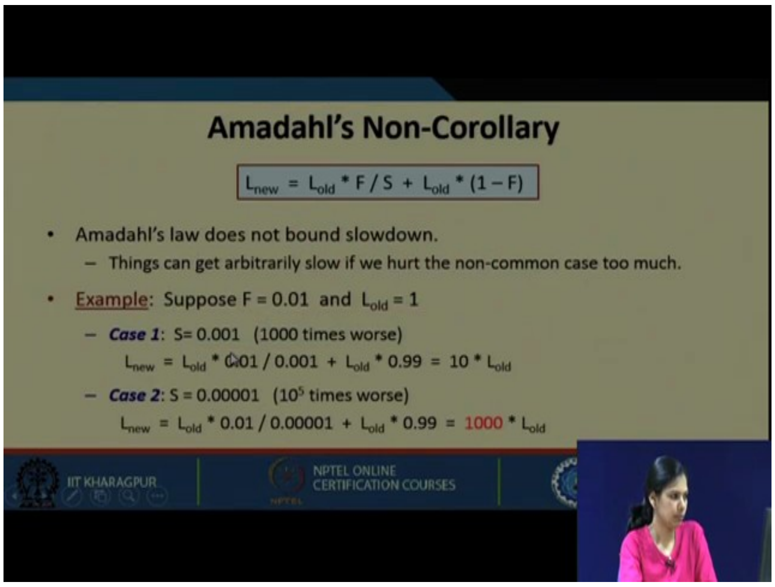
- Amadahl's law for latency (L)
  - By definition, $L_{new} = L_{old} / Speedup$
  - By Amadahl's law, $L_{new} = L_{old} * ((1 - F) + F / S)$
  - We can write:

$$\boxed{L_{new} = L_{old} * F / S + L_{old} * (1 - F)}$$

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now Amdahl's law Corollary 2, for latency what it says let us see. $L_{new} = L_{old}$ / speedup.

(Refer Slide Time: 10:33)



Amadahl's Non-Corollary

$$\boxed{L_{new} = L_{old} * F / S + L_{old} * (1 - F)}$$

- Amadahl's law does not bound slowdown.
  - Things can get arbitrarily slow if we hurt the non-common case too much.
- Example: Suppose F = 0.01 and $L_{old} = 1$
  - ***Case 1***: S = 0.001 (1000 times worse)
    $L_{new} = L_{old} * 0.01 / 0.001 + L_{old} * 0.99 = 10 * L_{old}$
  - ***Case 2***: S = 0.00001 (10^5 times worse)
    $L_{new} = L_{old} * 0.01 / 0.00001 + L_{old} * 0.99 = 1000 * L_{old}$

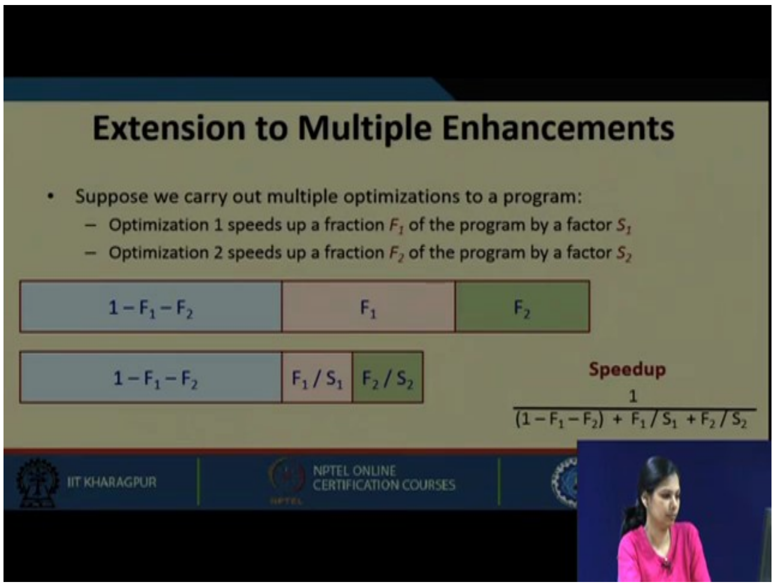
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Let us see this Amdahl's non-corollary. So, here what it says that, Amdahl's law does not bound the slowdown; that means, we are always saying that a part of the program where improvement can be made, and you can finally, get an overall speedup of this much. But what if those portions where we are not making, if you slowdown that particular portion then how it will affect?

So, the parts that are not used so much and you are trying to further make some techniques such that that particular part will reduce. I mean, the performance of that part will further reduce. So, you are not making any speedup on that and you are for making something such that, that performance of that part will further reduce. Then what will happen? So, things can get arbitrarily so slow if we hurt the non-common case too much. So, we should not do that as well. So, those portions which are not so common we cannot hurt that too much. So, we will see this example suppose F is 0.01 and L_{old} is 1.

Now, we are making a part 100 times worse. So, this becomes the speedup becomes 0.001 on the uncommon part that that that is now your F. So, L_{old} will become multiplied by 0.01. That is, 0.99. So, how much it is coming? It is coming 10 times the old value. So, L_{new} is 10 times the old value. So, it has worsened. And in case 2, if it is 10^5 times worse then what is happening? Finally, we are getting 1000 * L_{old}, which is even worse. So, we cannot just make the non-common part too much bad.

(Refer Slide Time: 13:18)



Extension to Multiple Enhancements

- Suppose we carry out multiple optimizations to a program:
  - Optimization 1 speeds up a fraction F_1 of the program by a factor S_1
  - Optimization 2 speeds up a fraction F_2 of the program by a factor S_2

Speedup

$$\frac{1}{(1 - F_1 - F_2) + F_1/S_1 + F_2/S_2}$$

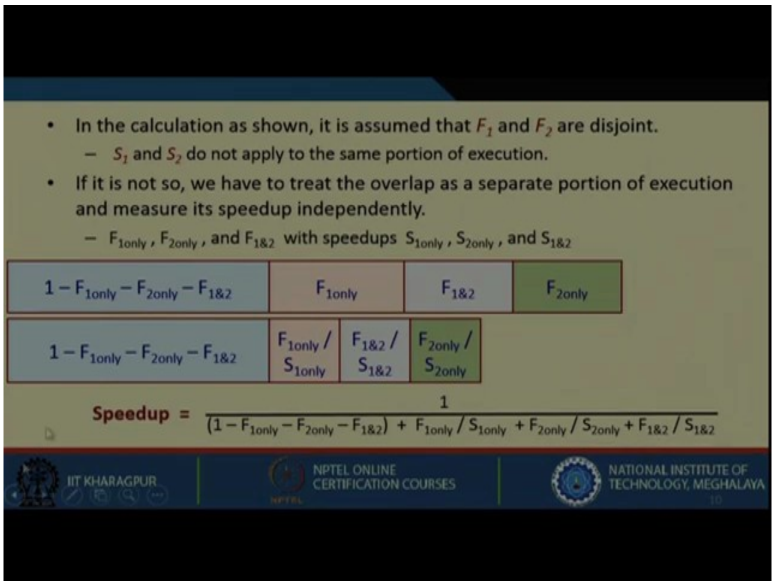
Now, let us see this extension of Amdahl's law to multiple enhancements. By multiple enhancement what we are meaning is that, earlier we were we were saying that we have a total portion, out of which one part we are making some improvement. And the other part remains the same. Now let us say, in a computation, we have IO bound job, we have

CPU bound job we have some other kinds of job as well. And now we are making improvement on these parts. So, earlier we were just making improvement on one, now we are saying that we will make improvement on these as well, others as well. So, in that case what will happen? How this multiple enhancement can be taken care in Amdahl's law? Let us see we have already seen for one part one fractional part if you improve what final speed up we will gain. Now we will see that we have multiple parts and we are improving multiple parts, and now how much you will gain.

Suppose we carry out multiple optimizations to a program. So, optimization 1 speeds up fraction F1 of a program by factor S1, optimization 2 speeds of fraction F2 of a program by a factor S2. So, F1 and F2 are two fractions, on which we are making the improvement now. On which we are making improvement S1 for F1, S2 for F2. So, this is the part where no improvement can be made. And this is a part on which we are making improvement S1. And this is a part where we are making improvement S2. So, earlier it was taking F1 and F2 and it has got reduced, and now we are getting F1 / S1 and F2 / S2.

So, similarly what will be the speedup? The speedup formula will be same. So, this will be the total improvement when we are saying that we have multiple enhancements. We are not making enhancement on a single portion, rather we are making enhancements on both the portions.

(Refer Slide Time: 16:11)



- In the calculation as shown, it is assumed that F_1 and F_2 are disjoint.
  - S_1 and S_2 do not apply to the same portion of execution.
- If it is not so, we have to treat the overlap as a separate portion of execution and measure its speedup independently.
  - F_{1only}, F_{2only}, and $F_{1\&2}$ with speedups S_{1only}, S_{2only}, and $S_{1\&2}$

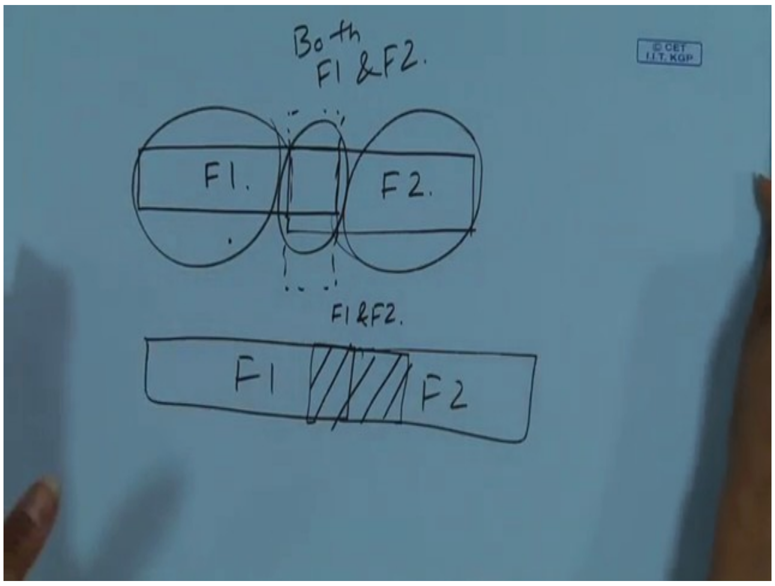
$1 - F_{1only} - F_{2only} - F_{1\&2}$	F_{1only}	$F_{1\&2}$	F_{2only}
$1 - F_{1only} - F_{2only} - F_{1\&2}$	F_{1only} / S_{1only}	$F_{1\&2} / S_{1\&2}$	F_{2only} / S_{2only}

$$\textbf{Speedup} = \frac{1}{(1 - F_{1only} - F_{2only} - F_{1\&2}) + F_{1only} / S_{1only} + F_{2only} / S_{2only} + F_{1\&2} / S_{1\&2}}$$

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, let us say in the calculation as shown it is assumed that F1 and F2 are disjoint. So, in this particular case F1 and F2 are disjoint. And what we are doing that we are making improvement on this part only and this part only, but now let us say there can be a situation like this.

(Refer Slide Time: 16:38)

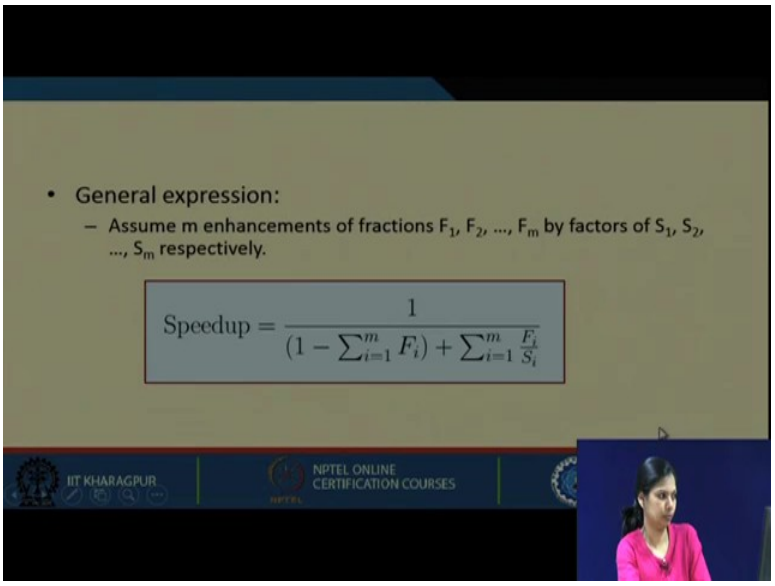


So, in this situation what happens? This is your part, with says this is your F1 part and this is your F2 part. And this part is having both F1 and F2. So, this enhancement can be taken care of separately, this part enhancement can be taken care separately, this part this part, and this part separately. Earlier it was having disjoint. So, earlier the case was this was one part this was another part. So, this is considered F1 and is considered F2. No common part was there, but now we are extending F1 till here and we are extending F2 till here. So, this has become now the common part, which contains both F1 and F2. Now under such scenario, let us see how we will calculate the speedup. So, S1 and S2 do not apply to the same portions of execution.

If it is not so, we have to treat the overlap as a separate portion of execution, and measure its speedup independently. Now this is very true that, this is one part, this is one part. And this should be also considered another part. And we have to calculate speedup separately. Although, we have a common part of both F1 also is here and F2 is also here, but we have to take care of this. If it is not so we have to treat the overlap as a separate portion as it is not disjoint. So, we have to take care this part, this part, and this part separately. So, in this case what will happen? This is the portion where no change will happen, and these are the 3 portions where you are making some changes. Where this part is F1 only, this part is F2 only, and this part is both F1 and F2.

So, what we are doing? F1 only with S1 only, F1 and F2 with S1 and S2, and F2 only with S2 only. It was initially this much, after this reduction it has become this much. Initially this much after reduction it has become, finally the speedup equation can be said as this. So, we are taking into consideration all the parts, plus F1 only divided by S1 only, plus F2 only divided by S2 only, and then take into consideration both F1 and F2 divided by S1 and S2.

(Refer Slide Time: 19:55)



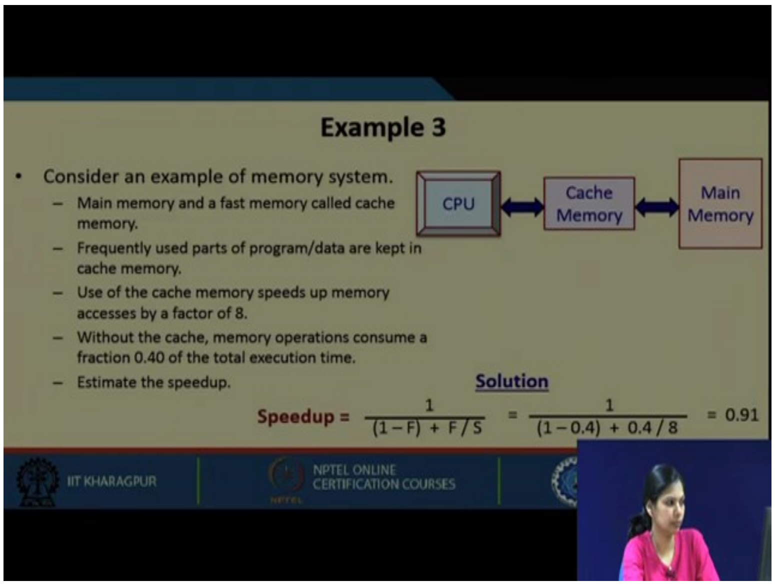
- General expression:
  - Assume m enhancements of fractions $F_1, F_2, \ldots, F_m$ by factors of $S_1, S_2, \ldots, S_m$ respectively.

$$\boxed{\text{Speedup} = \frac{1}{(1 - \sum_{i=1}^{m} F_i) + \sum_{i=1}^{m} \frac{F_i}{S_i}}}$$

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, this is the general expression just now what we have said. So, we assume m enhancements of fractions. So, in that case what will happen? The overall speedup will be 1 divided by (1 - summation of all these F_i's). How many enhancements will be there? Those many F_i's will come here. Plus, summation of F_i divided by S_i. So, this is a general expression of speedup when multiple enhancements are taken into consideration.

(Refer Slide Time: 20:45)



Example 3

- Consider an example of memory system.
  - Main memory and a fast memory called cache memory.
  - Frequently used parts of program/data are kept in cache memory.
  - Use of the cache memory speeds up memory accesses by a factor of 8.
  - Without the cache, memory operations consume a fraction 0.40 of the total execution time.
  - Estimate the speedup.

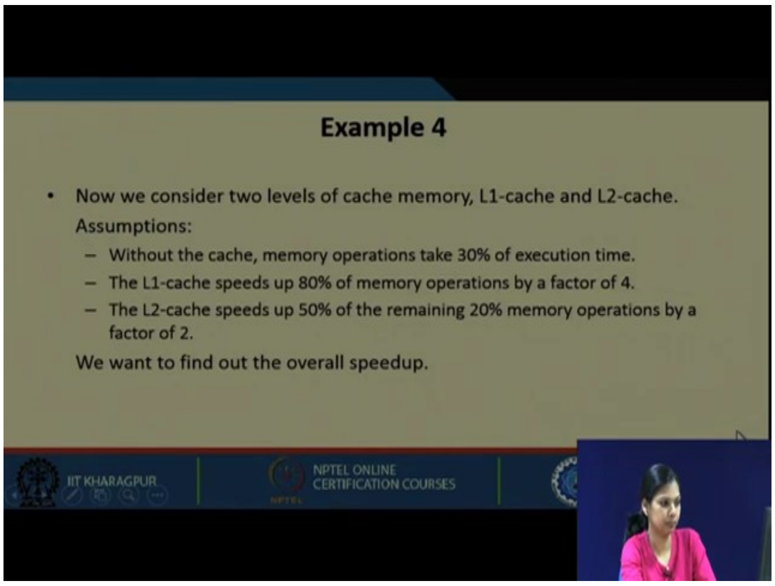
Solution

$$\text{Speedup} = \frac{1}{(1-F) + F/S} = \frac{1}{(1-0.4) + 0.4/8} = 0.91$$

Now, let us take an example. Although we have not discussed about cache memory yet, I can just tell you that cache memory is a fast memory that sits is between CPU and main memory. And it is used to enhance the overall performance of the memory, i.e. overall speedup of the memory. So, consider an example of memory system. Where main memory and a fast memory called cache memory is used. So, this is your cache memory, this is your main memory. Frequently used parts of the program or data are kept in cache memory.

Suppose, use of cache memory speeds up memory access by a factor of 8. So, whenever cache memory is used then the speedup of memory is by a factor of 8. Without cache the memory operation consumes a fraction 0.40 of the total execution time. So, what will be the speedup? So, in this case as without cache memory it was taking this much. Now using cache memory, you can actually make us speedup on this 0.4 by a factor 8; and the remaining will not change. So, if you put this in the formula of speedup you will be getting 0.91.

(Refer Slide Time: 22:41)



Example 4

- Now we consider two levels of cache memory, L1-cache and L2-cache.
  Assumptions:
  - Without the cache, memory operations take 30% of execution time.
  - The L1-cache speeds up 80% of memory operations by a factor of 4.
  - The L2-cache speeds up 50% of the remaining 20% memory operations by a factor of 2.

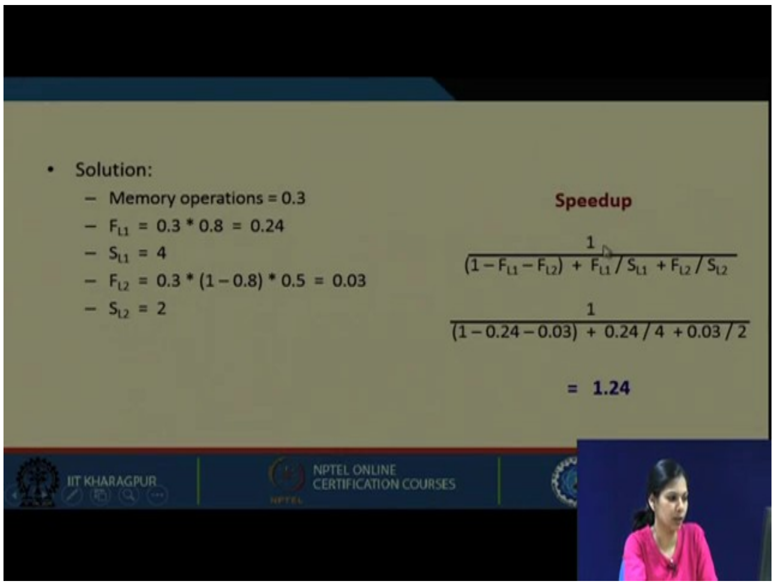
  We want to find out the overall speedup.

Let us take another example here where we consider 2 levels of cache memory where we see that the first level is called L1 cache. And the next level is called L2 cache. The assumption here is, without the cache memory, the memory operations take 30% of execution time. When L1 cache is used it speeds up 80% of the memory operation by a factor of 4.

Now, out of total execution time only memory operation is 30%. Now out of that 30%, if L1 cache is used 80% of the memory operation improves by a factor of 4. And the L2 cache speeds of 50% of the remaining 20% of the memory operation by a factor of 2. So, there are 2 things out of total execution time, the memory operation is taking 30%. Out of this 30%, now 80% of the memory operations are found in L1 cache. And 20% of the memory operation is found in L2 cache.

So, when L1 cache is used 80% the memory operation is improved by a factor of 4. And when L2 cache is used, 50% of the memory operation speeds up by 20% memory operation by a factor of 2. Because this is 80% to remaining 20% of the total 30% is 50%. So, L2 cache speedup is 50%. We want to find out the overall speedup. Let us see how we can find out the overall speedup.

(Refer Slide Time: 25:02)



- Solution:
  - Memory operations = 0.3
  - F_{L1} = 0.3 * 0.8 = 0.24
  - S_{L1} = 4
  - F_{L2} = 0.3 * (1 – 0.8) * 0.5 = 0.03
  - S_{L2} = 2

Speedup

$$\frac{1}{(1 - F_{L1} - F_{L2}) + F_{L1}/S_{L1} + F_{L2}/S_{L2}}$$

$$\frac{1}{(1 - 0.24 - 0.03) + 0.24/4 + 0.03/2}$$

$$= 1.24$$

Memory operation 0.3, that is 30%. So, F_{L1} will be when L1 is used; as L1 is used 80% of the time it will be 0 0.3 * 0.8 = 0.24. And S_{L1} is 4 that is already given. Similarly, for F_{L2} that is 0.3, multiplied by it was 80%. So, this will be 20%, out of 20% it is 50% used. So, 0.5 that is coming on to 0.03 and S_{L2} will be 2. Now we will put this value here and we are getting a speedup of 1.24.

So, we came to the end of lecture 16 and which will end the week 3 lectures. So, in this week we have actually seen how we can calculate performance. How Amdahl's law is actually happening. And how it is helping us to determine that, how much speedup can be actually achievable when only a part you are actually making the improvement.

Thank you.