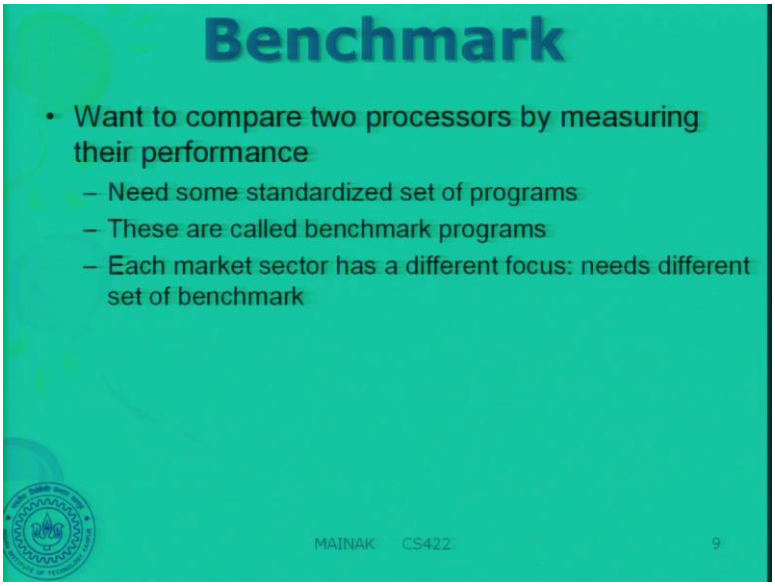
Computer Architecture
Prof. Mainak Chaudhuri
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

Lecture - 1
Introduction, Amdahl's laws, CPI equation

(Refer Slide Time: 00:15)

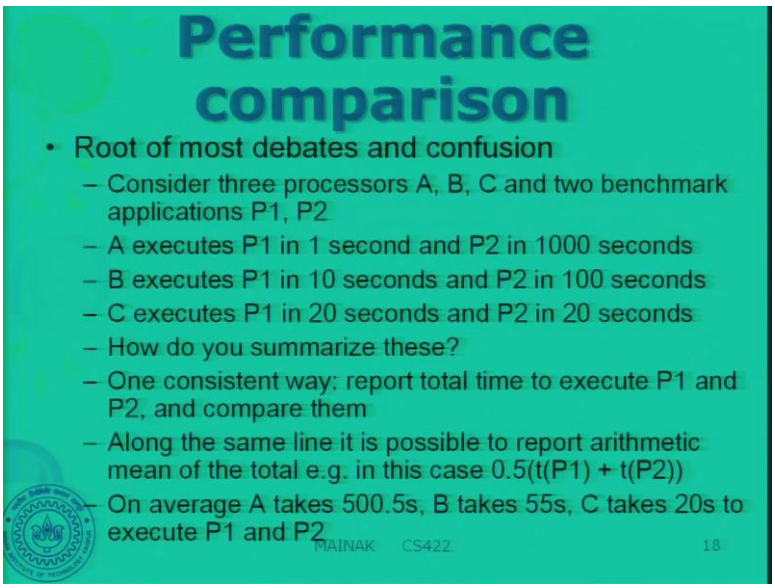
Talking about benchmark applications.

(Refer Slide Time: 00:20)



He also talks about some of the metrics that we use to measure performance, we discuss the benchmark applications for several markets segments including desktops servers and embedded process.

(Refer Slide Time: 00:37)



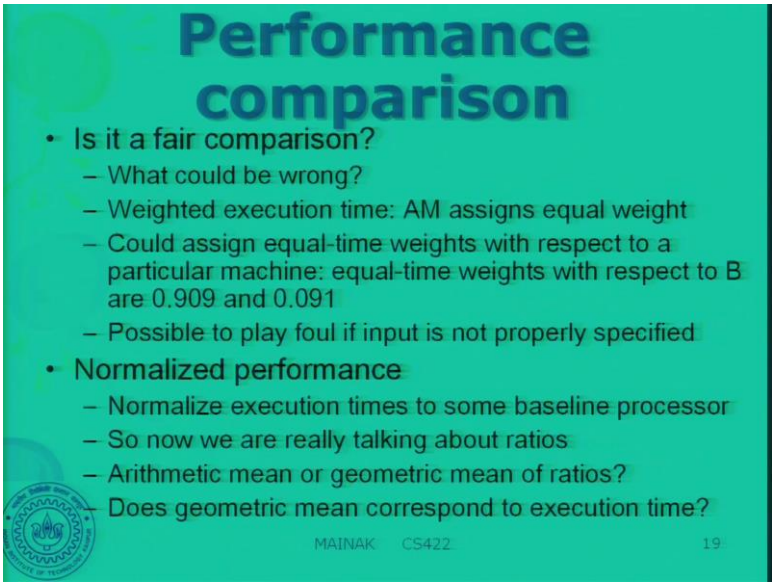
So, today we will start with perform (( )) example how do you (( )) b and c with the two

benchmark applications P 1 and P 2.

A (( )) in one second P 2 in thousand (( )) b executes P 1 in ten seconds and P 2 in hundred seconds and c executes P 1 in twenty seconds and P 2 in (even this data (( )) which of a b c is the best one right. Because you can see one then (( )) P 2 where as in both cases b is in the middle of the benchmark (( )) processes you would be dealing with hundreds of application meaningful summarization of this data that meaning by meaning (( )) which misleading that that should not lead you which is actually not the not a good point ok

So, let us take a take a look at the possible ways that we have we put the total time to execute P 1 and P 2. So, similarly along the same line it is possible to report arithmetic mean of the total right. So, is the same thing in these two (( )). So, if what you get is on (( )) 500 and half (( )) b takes fifty five seconds and c takes twenty seconds to executes P 1 and P two. So, according arithmetic mean you would conclude that well c is the best. So, that well I would go on by c right do you say any problem with that P 1 may be a more popular.

(Refer Slide Time: 03:29)



So, two alligate this problems is there can execution mean assigns to everybody. So, as

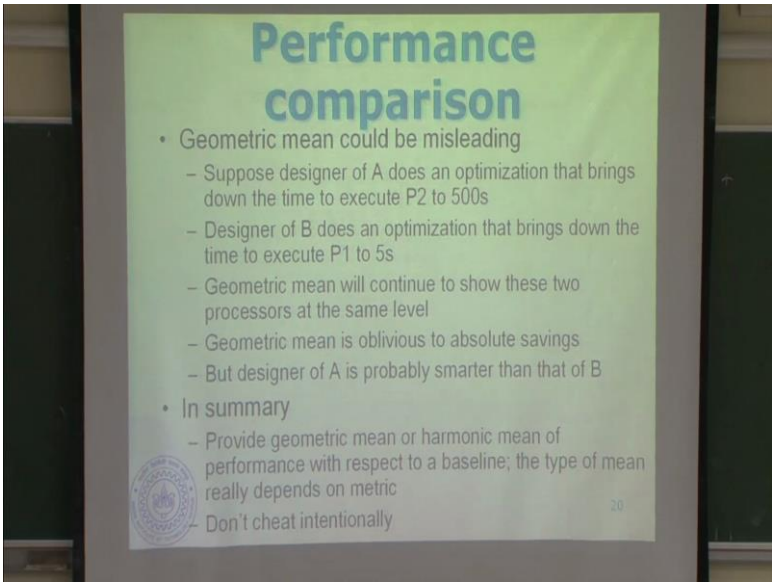
somebody always suggested P 1, is it which is often very difficult to. So, what one of although it probably does not have much meaning has such you could assign equal time with respect to a particular machine. So, what do a mean by that if I take machine b what is that I assign weight to P 1 (( )) amount of time right

So, equal time weight with respect to be would be P 1 getting a weight of point b nine and b and P 2 get weight of point 0 nine one point nine one nine if you give weightage of point nine one nine to P 1 you get ninety point nine seconds of execution here this will (( )) enter one nine one weightage to to program. So, you might wonder wine why should anybody do that mean why wherever should be a equalize P 1 and P 2 to b, but any ways. So, the point taking that particular machine now you can report, but it is easy to machine.

The the other option is to report normalize and that is normally normally we have normalize execution times to some ways. So, the point is that will you designing the new processer you already have one and your improve upon that. So, it make sense to specify your performance improvements related to the current process that is after call the base . So, now, we really talking about ratio's. So, so for example, example if I base line I would report the performance b and c respect to and you say that P 1 then b is .

Whereas could be state as faster P 2 talk about ratios. So, how do you summaries ratios same questions again arrives and again you have options you can do arithmetic mean you can geometric mean which one make sense the problem is that arithmetic mean if it a arithmetic mean of time, if it geometric mean of time it is a normal time it something is number; however, geometric mean of ratio has many good properties.

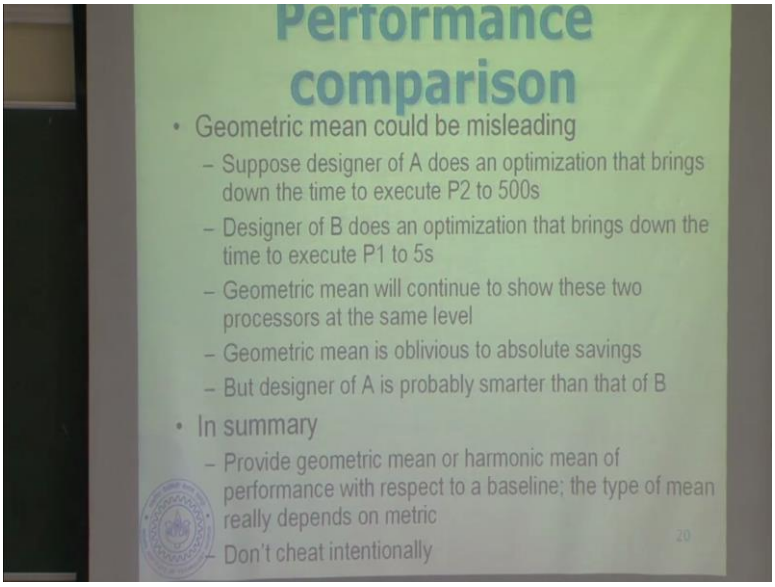
(Refer Slide Time: 06:43)



So, we come back to that before that here is the example which shows how geometric mean can be using suppose desire of a does not optimization; that means, down the time previously first .

Is that b does not optimization that brings down their time to execute P 1 to five seconds previously it was ten seconds geometric mean will continue to show these to process's the same level because same what designer of b does is that half's the execution time of P 1 and design of a does is half the execution of P 2. So, if I if I if I take the ratio of these. So, if I couple of a and b here. So, I have 2 ratio's for P 1 the ratio is one over ten, but P 2 is ratio is ten I take the geometric them I take the new optimization machine a and b and the same thing same geometry.

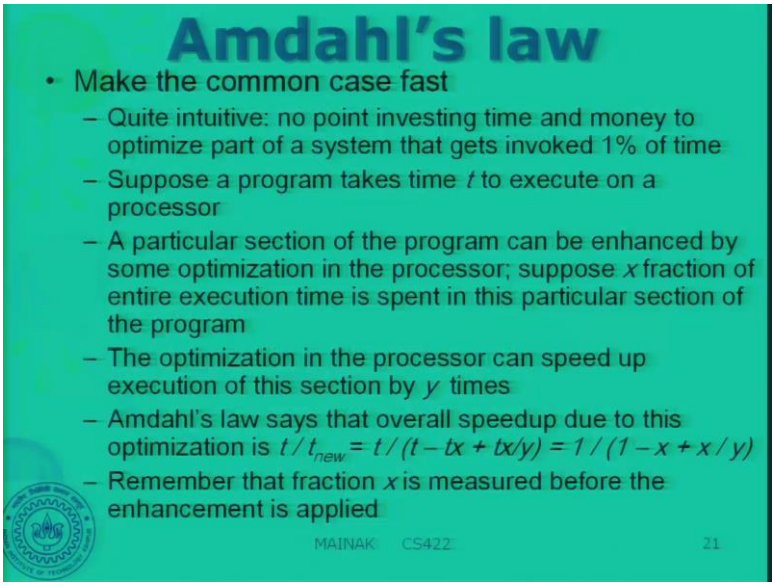
(Refer Slide Time: 07:51)



So, geometric meaning of previous we absolute say yes designer of the a. Smarter than that will b because designer of a was able to say 500 seconds the designer of b something to say. So, summery provide geometric mean of harmonic mean a performance which aspects the base line the type of been really depends on metric and the that do not cheat intentionally. So, that is the most important point the geometric mean in mention ratios and the reason is that the geometric mean of the ratio is same as the ratio of the geometric mean of the company become easier and people harmonic. So, how would be arithmetic and geometric is the largest arithmetic. So, harmonic mean has the good property that it usually works your performance improvements because which is the lowest.

So, if you really want to be in terms of reporting your achievements. So, that you are the same side that you are not probably to arrogant mistakes reporting your your results where as if you if you really want magnify you achievement that is the rural. So, as we go on along the course I will also mention what mean to use what particular type of measurement. So, there are certain things that also need to taking care of questions on this measurements. So, summarization.

(Refer Slide Time: 09:51)



So, people usually use means harmonic mean is the most popular one sometime geometric mean. Right and the an usually these are the statistic that you normally see and do not try to for higher order statistics reporting of summarizations. So, any question all right. So, now, will talk about particular law which is the actually. So, was the P h d student in physics to responding when was having 50. So, the transistor were just coming. So, develops certain concepts or theories in physics and certain equations that he needed to solve which he could not solve. So, numerical solutions where required machine computer.

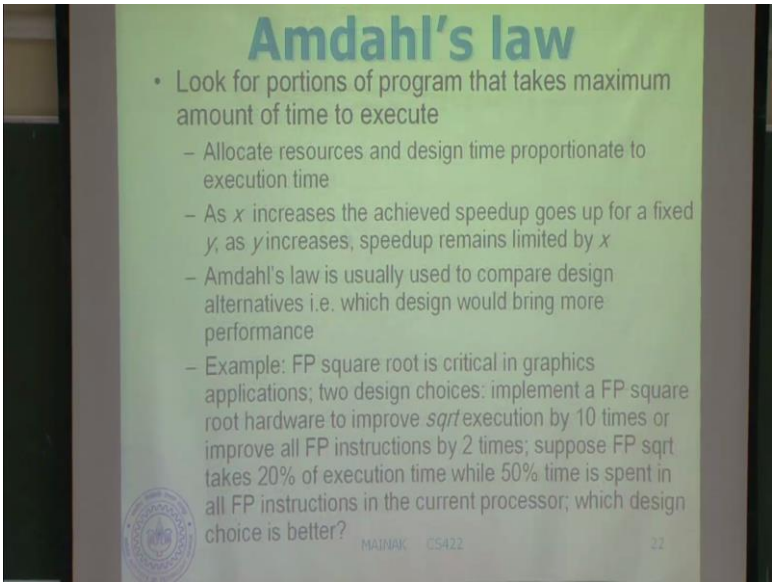
computer museum actually available there the tenth and they know ambrall saff a I m you'll never come to in nineteen sixty seven we came up in this law which essentially common sense. So, which says that make the common starts. So, exploiting to the which essentially say is that there is a point in time and money to optimize part of the that we should really the optimizing the parts of program for example, which requires maximum time to execute. So, like for example, if we have the function which gets ninety percent of time you better money to optimize that function forget about the various portion relax that is a o mathematically ig you want to put it a forty plus.

Session of the programs can be enhance by some optimizing the processer. So, let us

suppose extraction of the entire execution of time in this particular section of the program. So, we saying that your program and certain part of the program can optimize software or hardware it does not be exactly matter of how do and extraction of the entire execution time the original execution it spent in the optimizing in the processer can speed up the execution of the section by wise. So, what does it mean; that means, the speed up that you get t over t new t was previous execution which t over t minus t x because t x t x y it is better these portion. So, that is what you get one over one minus six. So, that is the speed of you get and that is the that is what .

And if look at this for are equation and you may ask the questions right that suppose I say that you can get the unbounded amount of speed up. So, y is what will happening in that case amount of by one over one minus x there's after whatever you. So, let me that is a very important point that says that the portion of program that is cannot visible ultimately which is the actually again common source the... So, fracture x is measure before the applies. So, that is the very important thing not the it is not after next this is actually go to change this in the apply it scenarios. So, which as says that you know wherever you can. So, here of course, that the assumption is that x is large that is what time. Because we can see that if x not very large ultimately you can 1.

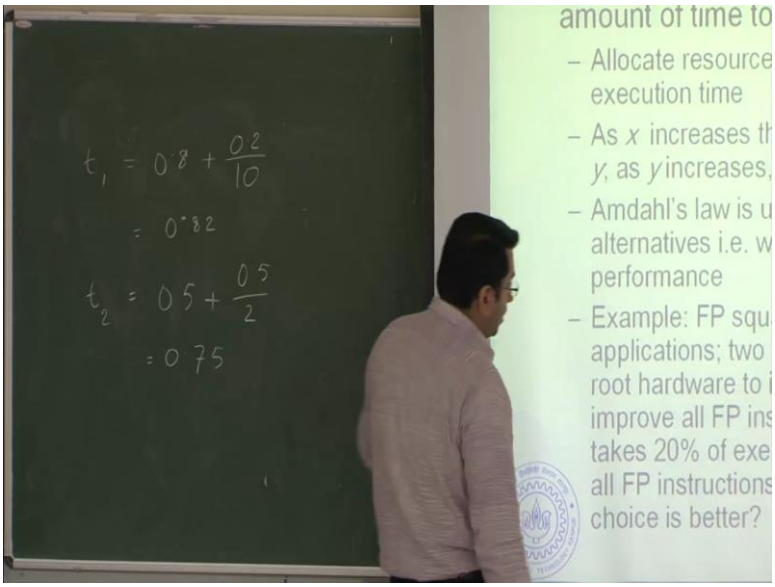
(Refer Slide Time: 14:21)



So, for portion of program that takes maximum time to execute the allocate resources and design time proportionate to execute time as x increases as x increase the achieved speedup goes up for a y as y increase speedup remains limited by x amdhal's law is usually used to compare design alternatives that is which design would bring more performance. So, let us take a simple example fording point square root is critical in graphics applications two design choices exist.

If it fording point square root hardware to improve square root execution by times that is one design choice the second choice is improve all fording point instruction s by two times. So, suppose you have these two choices which way you would go. So, suppose fording point square root takes twenty percent of execution time. So, this is the portion that we can action that 50 percent time spent in all fording point instructions in the current processer. So, which design choice better. So, let us walk this out.

(Refer Slide Time: 15:45)

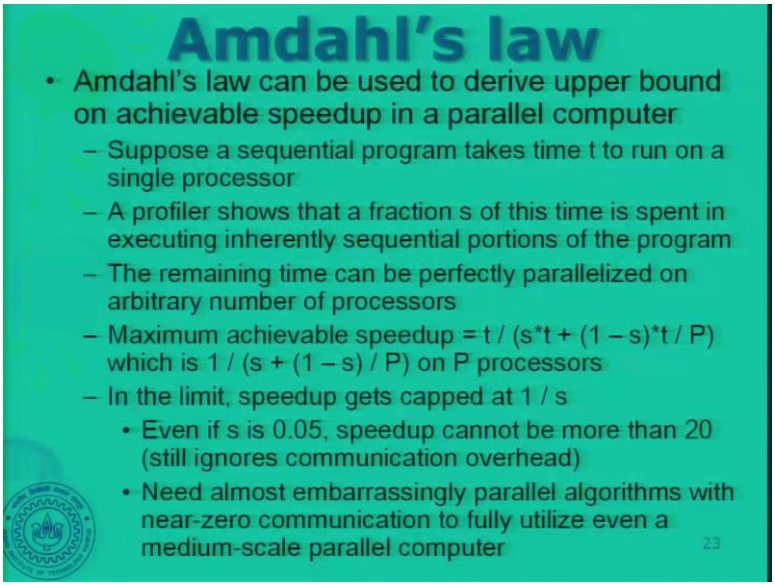


So, let us suppose in the choice one what we have. So, t one is the time taking to execute in the first choice. So, here we are saying that we are focusing on the 20 percent of our execution time that is 40 point square feet. And up by. So, what we get is 0.8 and the remaining point two will get skilled of by ten. So, what does it give it 0.82. So, am actually at the originally started with one is the time of one choice one what is a choice

two we are looking at 50 percent of time that. So, remaining 50 50 percent would be may not change. So, 0.5 and the remaining 50 percent by there are fifty points the second choice is. So, I should actually go head and I think to all 40 point instructions.

So, if you look at... So, so clearly in this upper in these case happened was that the time spent the 50 percent time spent to fording point operations I focused on that I got better performance, but it also depend on this factors one was ten times two times. So, you can easily figure of what we. So, it is not that you can always speak up the time improved by the some factor and you are always going to be for example, if I this is on point five instead of two. So, below one 0.51.2 in that will be option one. So, you can actually plugins the numbers formula we will get directly the answer without going through the calculations questions.

(Refer Slide Time: 18:22)



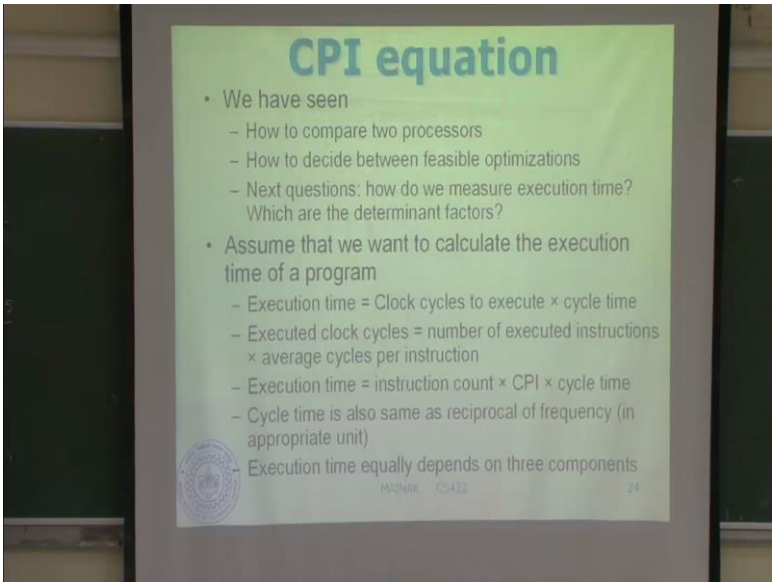
So, Amdahl's law can be used to derive upper bound on achievable speedup in a parallel computer. So, these this is use very often you find out find out achievable speedup for particular. So, let us see how you do that suppose a sequential programming a profiler a shows that. So, profiler those who do not profile is software which takes a program and tells you where is time spent how much time spent in.

It shows that fraction s of this time is spent in executing inherently sequential portion of the program. So, that s fraction has no chance of getting parallelized it is inherently sequential the remaining time can perfectly parallelized on arbitrary number of processor suppose you figure. So, what does. So, now, I can actually did use speedup machine with pre processors how do I do that the speedup is sequential time that is t divided by parallelized what is my parallelized time s times t will remain sequential and one minus s time will be divided by P it has perfectly parallelized. So, that gives be one over s plus 1 minus s P on the sequences I could actually get this.

Get this by plugging in my it is exactly Amdahl saying that I could speak up this for execution by P times alright. So, what are the changing something interesting can see. So, in the limit the speedup gets capped at one over s these are most important part of the this analyses essentially tells you in finite number of processers cannot get the impossible, now if you look at some the things if you plugins values there suppose f s is 0.5 the program is ninety percent parallelized the speedup cannot surprise am running of this show 32 machine is 95 percent parallelized only 20 times you cannot go.

An off course ignores communication overhead many other processors they actually got utilize. So, essentially this particular tells you that you need almost embarrassingly parallel algorithms with near 0 communication to fully utilize even a medium scale parallel computer by medium scale I mean you know number of notes below fifty five if we have a 99 percent parallelized program your speedup will go to hundreds not. So, you need almost completely parallel you two exploit large scale of. So, that is the very, very important flow I wanted to mention this although probably not use this particular thing in this course these are normally use the courses that we need parallel programs or parallel computer lectures, but keep this in mindany question.

(Refer Slide Time: 22:11)



So, the next thing that is important for performance measurement is something called CPI equation again this is common sense. So, we are seeing that you know how to compare the processors you know how to decide physical optimization by applying amdahl's laws the next question is how do measure it, it is time because here we knew that you know some percentage of execution time cannot be analyze some percent can be enhance do it to measure that. So, which of the terminal factors assume that we want to calculate the execution time of program. So, execution time we guess clock cycles to execute multiplied by cycle time.

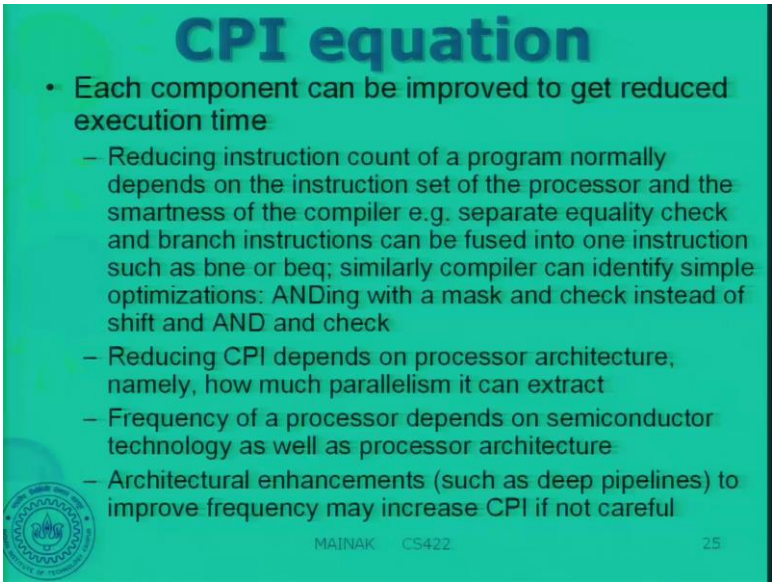
We take hundred cycles your cycle sign is one nano second you required hundred nano second to execute the now if I expand this particular term to execute the clock cycle this is equal to number of executed instructions multiplied by average cycle per instruction. So, execution time now becomes now to instruction count one multiplied by cycles or instructions and CPI multiplied by the cycles. So, this is the CPI equation interesting part of this is that this particular first time which determine by the processor an almost nothing to do it combiner generates a binary and that is what executes in processors that determines how many instructions.

Of course we will also depends on a that is a means which program cycle time is mostly

determine by underline semi corner the technology; however, which by the architectures these one is only cycles for exception and that is what an architect holds power we can do something to view your CPI, and that is what we are going to interest. So, I progressive the course to bases of depends to see how CPI and we will also mention solve the things to include cycle time after you will find that these two are inter related often you will try to improve CPI.

You get the... So, we have to keep both of these time. So, cycle time is also same as reciprocal of frequency in appropriate unit for example, one gigahertz my cycle time is one. So, execution time equality depends on three components they equally weight and see each components reduce the execution time reducing instruction count of a program normally depends on the instruction set of the processor.

(Refer Slide Time: 25:36)

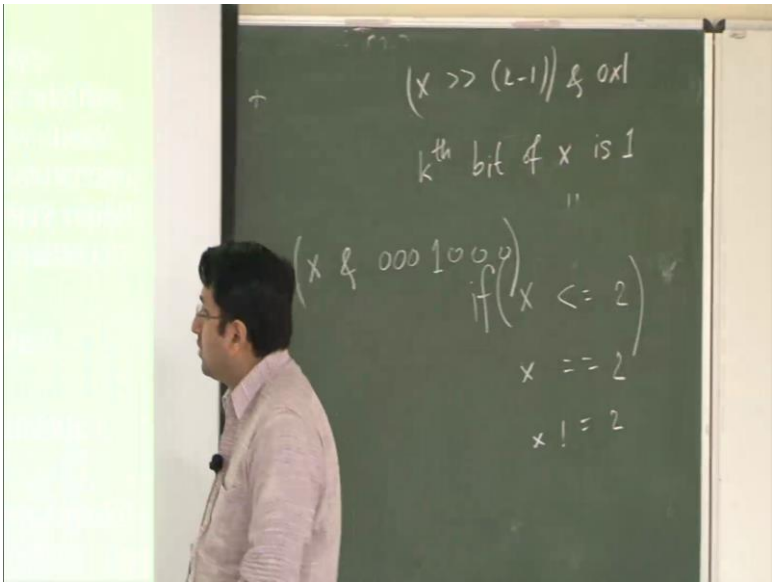


And the smartness of the complier that is separate. So, so let me first first part of the. So, it says that we depends the instruction set of the processor. So, that; that means, some we talk about instructions. So, ruff is seeking what it means that up the processor support certain set of instruction they need be very complicated .

Now, to do a very complicated offer that processor may have for example, we can think

about the operation live stream stream means stream of right from one part of the memory to another part there we there we just one instructions even though it is doing a lot of operations particular on the other hand another processor may be able to maybe maybe expose this to the compile I could allow you to copy you more than four bites at a time only have four bite copy you can copy four bite of data from you to . So, now, copy which essentially you get transparent which copy four bites of time. So, there is the of here in the first case the CPI is particularly very high instructions (( )). The second case in instructions count will be large, but CPI like to do one or may be. So, here is an example certain processor have separate instructions for doing the comparisons followed by checking the comparison outcome to taking a branch.

(Refer Slide Time: 27:31)



So, for example, I have suppose a piece of Sea code which. So, there are many ways of compiling one way could be that in first two x less that equal to 2 of the comparison is one instruction. So, that generates some out come true false they say some flags somewhere all right the next instructions goes and takes that flag and decides where to jump whether to jump.

If it is exact actually x less than equal two you would not jump control your security otherwise you jump to the x or there is no else part just keep the your portion and then

start. So, these are instructors all by a transistors and other processors who could actually fuse this to get these to operations to say that branch. So, you do not the this operation actually I say that branch is greater than. So, that exactly what is saying here separate equality check and branch instructions can be fused one instructions such as b n e or b e q. So, these are branch not equal to a branch depending on the nature of the here are I am showing less than equal to here is talking about x equal to 2 x not equal. So, in these two cases essentially what to happen is that the instruction come to the program.

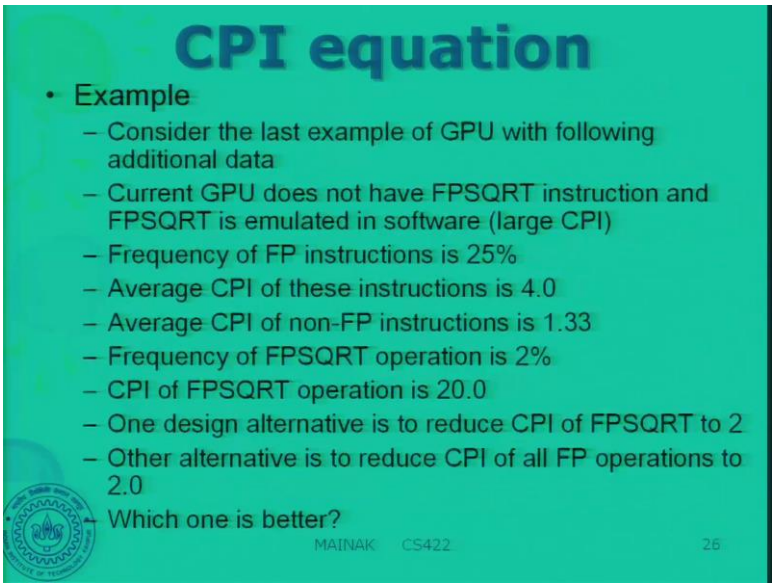
We checks and this was possible in the second case to fuse this only because processor actually processor actually implement advance. So, that is where the support from instruction set the instruction set has the instructions of the otherwise cannot generate this instructions it will break it down two piece similarly compiler can identify simple optimizations. So, these example examples purely about profiler of ending with a mask and check instead of shift and check so we are talking about something like this I want to check k bit of the x is 1 alight there are two ways of one is that, two an x with the masks. So, so what is that mask can somebody guess what sorry of a is that does anybody see that what what I want in mask actually I want a one in the right everything else should be 0. So, I ending this mask and now is to enough to check this is 0 or non 0 then the exactly I have done this the other we have doing it is that I would shift x by one bits and then your and operations. So, essentially x shifted by k minus one and 0 x 1 I would.

This is much for expensive this is the and forward by a comparison shift and comparison. So, it beyond a depends on profilers smartness will profilers that I should be actually able to this not a this. So, that is about your instruction count. So, we talk about some of these things of the first categories that is designing instructions at without increasing your CPI too much. So, because see there is the there is the a key balance balance between instructions common I can make very complicated instructions. So, that you know this entire execution of the these keeps keep block will be the that possible I can do it the CPI going to very, so to balance.

The second component is a CPI and the goal is to minimize CPI reducing CPI depends on past architecture how much parallelism and that is exactly worth it is going to be

major portion of the frequency of a processor depends on semiconductor technology as well as processor architecture architectural enhancements such as deep pipelines to improve frequency may increase CPI if not careful. So, this what essentially it mentions how is to large the actual saying that I can reduce my cycle time we talk about this more in detail by designing a very deep pipeline alright. So, that each pipe doing a very small amount of water. So, the I can very fast alright. But displaying the CPI we will talk about I cannot changing mention right now how can see what.

(Refer Slide Time: 33:14)



Yes that is what I mentioned it depends on the instructions processors.Yes yes so, but once you instructions that it fix given a program how many instructions you going to execute it depend on a yes, here are compiler is able to generate the optimum number of the instructions that depends on. So, here is the example. So, let us check the same example of last one sum additional.

So, the current g P u does not support forty point square and fourteen point square root is software which means essentially what we do is is you implement some square root algorithm does not even you know of any algorithm, yes.

Newton’s formula.

Newton can can who said newton and you yeah

If you basically the approximation algorithm start with initially cycle

Can you just tell me how it form the newton or how to a.

Roots of polynomial

Yeah. So, basically you defined it as the polynomial like

What is the polynomial square root.

F x minus square root of x is equal to 0.

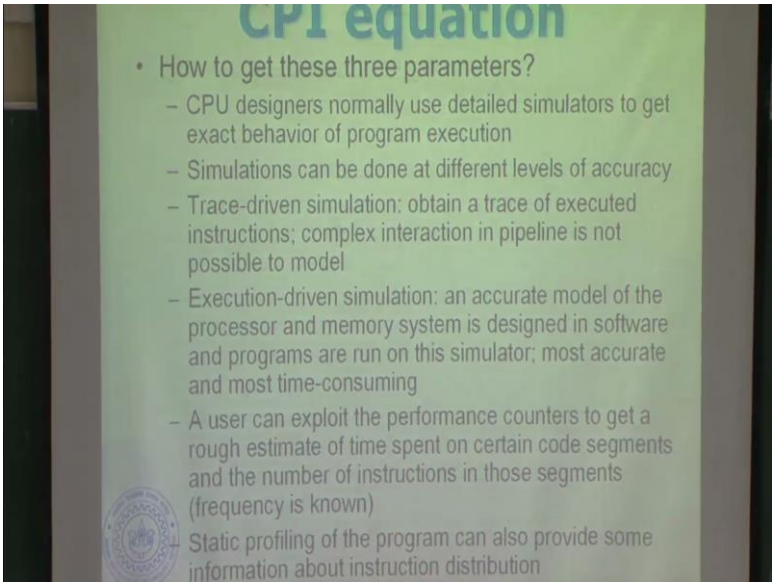
No what is it?

Y square minus c, right.

So, I am trying to evaluate say it square root of c. So, I would basically take x square minus c equal to 0. So, sorry (( )) is one of the simplest way of the giving this and there are other many smart algorithms and square. So, talk about one of those full stop. So, clearly it have a large number of instructions which means if I look at look at this large fording point square root it will look like single operations with a large CPI frequency of fording point instructions is 25 percent average CPI of these instructions is four point 0 average CPI of non fording point square root instructions is one point three three and frequency fording point operation is 2 percent alright CPI of fording point square root operation is 20.

So, five time more than average of all fording point. So, one design of alternative was to reduce CPI of fording point square root by same times coming 2 alright the other alternative was reduce CPI of all fording operations to by fourth 2. So, which one is better. So, the question that still remains any questions

(Refer Slide Time: 36:29)



How do you still get these three parameters alright we require number of instructions the CPI and cycle time will measure it time. So, CPU designers normally use detailed simulations to get exact behavior of program execution simulations can be done at different level of accuracy.

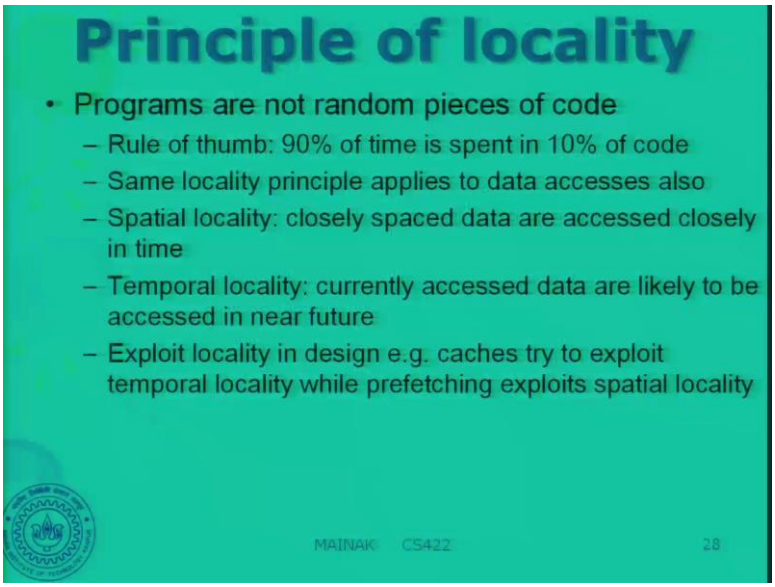
So, there are three options there is one called trace driven simulation obtain the a trace of executed instructions and trace simulator which essentially simulate the processor the trace goes to simulator and what comes out is of course number of instructions that we which of course already he can already get from the trace because he already have; however, it you also get cycle for instructions and you know the cycle time the problem which simulations is that complex interaction in pipeline is not possible to model, because the trace by running the program on some machine now you simulator you are the process that you are trying to simulate may be different from that machine, but there is no way to model those in interactions the trace of each instructions is already fixed by that executions on that machine.

So, exactly what you cannot find you cannot model you come back to that most important things that you, but in most cases this is the best possible option is do a execution driven simulation what is that it is an accurate model of processor and memory

system designed in software and programs are run on this simulator the essentially what to do is simulator can actually they finally, of the program and can interpret the final meaning g code the bio an actually execute the bio alright it is just like a machine executing in you program. So, this is the most accurate and also most time punctuality a user can exploit the performance counters to get estimate of time spent on certain code segments and the number of instructions in those segments frequency is known. So, today's machine already locked on large performance propose you can gets estimate of time spent is an codes.

Static profiling of the program can also provide an information about instruction distribution, it is also profiler program you know what type of instruction we have for example, we can find out information like 20 percent 40 point instructions is I have 50 percent instructions.

(Refer Slide Time: 39:31)



So, few sorry few principles that you should keep in mind one is principle of locality we will also on that I am already mentioned in last class that program are not random pieces of the that ninety percent of time is spent in ten percent of code it is a rule of thumb also. So, in average estimate of course there in that is what you will find in the simple reason is that.

The most interesting programs would have some kind of structures either if you loop or otherwise if you do not have any of these it is essentially straight line it is a program doing pretty much alright. So, any interesting piece of code would have loops or takes a lot of time execute the loop of the structure and that is where this small piece of code essentially lead to a large amount same locality. So, this is essentially say to the locality code locality I spent a lot of time same locality principle applies to data accessed also. So, there were two types of locality let talk about talking about one is called spatial locality which means that closely spaced data are accessed closely in time.

Temporal locality which says that currently accessed data are likely to be accessed in near future exploit locality in design for example, caches try to exploit temporal locality because what we are caching now will be cached hoping that in future near future will be touching the while prefetching exploits spatial locality that is what prefetching does not that touching data over x we will also make a x plus 1 x plus 2 x plus 3 x plus 4 be spatial locality saying that x. So, may be touching you nearby data also.