High Performance Computing for Scientists and Engineers
Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Module - 01
Fundamentals of Parallel Computing
Lecture - 11
Performance Metrics of Parallel Systems (continued)

Welcome. We are continuing our discussion on Fundamentals of Parallel Computing, the 1st module of the course High Performance Computing for Scientists and Engineers. We are in the last lecture of this module which is our lecture 11, we are continuing the previous discussion which we have started on Performance Metrics of Parallel Systems. As I was discussing that parallel system is deployed to do high performance computing simulations.

Why do we need high performance computing? Because simply if we solve using a simple computer which we have been mentioning a sequential computing, it might require enormous amount of time if you are trying to solve a very large problem. In order to reduce the time in order to get faster solution we try to do high performance computing which is typically using multiple computing systems using multiple CPUs in different architectures to solve a single problem.

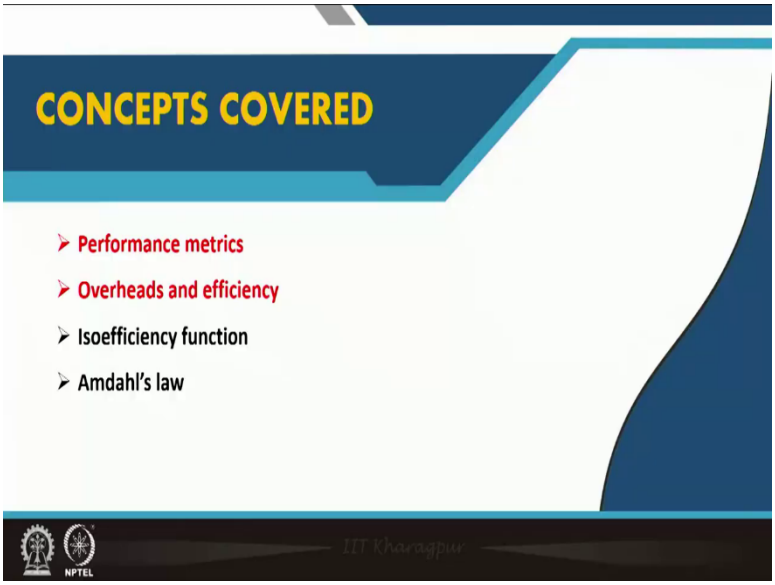
These CPUs work in parallel, they are basically trying to solve different parts of the same problem, but they working parallel. And we have seen that when all the CPUs are working in parallel, they will have some overheads due to interaction between them, due to setup of the parallel environment, and some other issues like improper load balancing latency of certain systems etcetera.

Therefore, it is extremely important that we find out a quantitative metric to see how good is the parallel environment or the parallel system architecture as well as how efficient is our algorithm by which we are trying to solve a serial a problem not used using a sequential computer, but using multiple computers .It also depends on how we are designing the algorithm.

We have discussed about design of parallel algorithms and how are we are utilizing different systems in the parallel in infrastructure. So, we have introduced about some

concepts in on performance of parallel systems in our last discussion and I am continuing with this.

(Refer Slide Time: 02:43)



So, we have introduced performance metrics which include the parallel computation time, the speed up, the efficiency the cost of computation etc. And we have also discussed about overheads and efficiency and we have seen that if we consider a parallel system to be very efficient, that means, that if we are increasing the number of processors the speed up is increasing almost linearly. The benefit is almost linear if we are adding two more processors or if we really doubling the number of processors, the computational time is getting halved that is a parallel system with efficiency one.

But we really cannot achieve that efficiency because there are overheads as we are increasing number of processors, they are interacting in between them, there are some other issues like load balancing, strong sequentially in some parts of the algorithm which does not make it fully efficient algorithm and we lose some efficiency which sees a reduction in speed up.

So, we will discuss about this efficiency and overheads little more. We have already started the discussion in our last class on performance metrics of parallel algorithm systems. We will see something called an ISO efficiency function, which is a very important parameter in order to estimate that, what should be the optimum number of processors, in which I should run my parallel program to get best efficiency and get the right speed up.

One thing we should also not forget when we are discussing about parallel systems that if we are having a large problem to solve and this is running over a large number of computers and processors, these computers do not come for free we have to spend substantial amount of money in establishing the set up as well as for running the setup, there is huge operational cost including the energy investment.

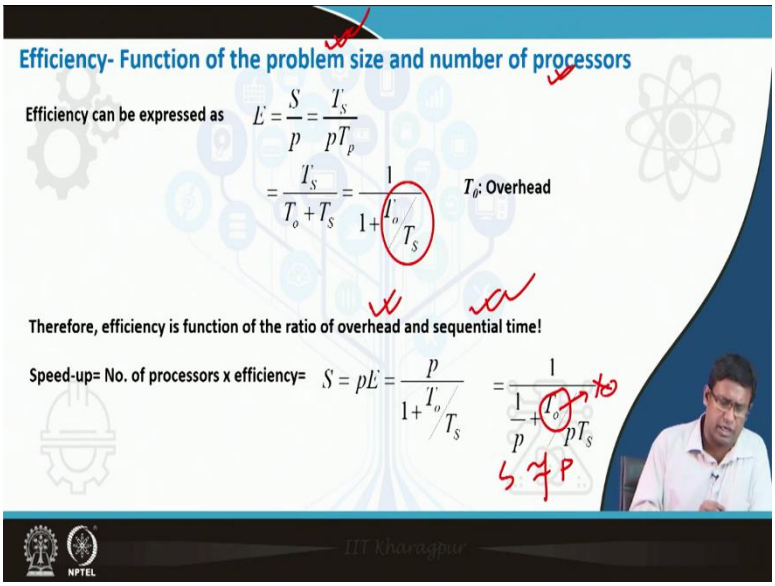
Also, these computers are they as they require energy, they do contribute to our energy scenario as well as to global warming. I mean data centers or the hubs in which we host the parallel computers, they are infamous for sources of global warming they are very high carbon footprint. So, we should be very judicious while deciding that what is the optimum number of processor and what is the optimum size of the computing environment in a computing infrastructure that we should use to solve this problem.

We should not take it for granted that we can increase the number of processors as much as possible and we will get a better efficiency or we will get the job done in a faster way. We also should be judicious in deciding how many processors we should work with and ,if somebody is in some place where there are a lot of computational resources he must be happy, but in general everybody working in high performance computing are challenged with the problem that the computational resources are limited you do not get large number of processors.

If you were submitting you have to a very high end supercomputer, you see that your job will go in queue and you have to wait few days say for 5 days run time and get your get some part of your problem solving you may not solve the entire problem in one set of run.

So, availability of the computational resource is also one very serious issue and therefore, understanding the performance metrics and understanding how to decide what should be the optimum number of processors to get right performance. Also understanding the fact that whether the program or the algorithm you have developed is the right one or whether we need to do something else to get better performance out of it are very important.

(Refer Slide Time: 07:01)



So, we look about efficiency in little more detail, we understand that efficiency is an important parameter while estimating performance of the parallel systems. In case a parallel system is completely scalable, that means, if we increase the number of processors by twice time required will be halved it will require half than the previous time requirement, we call that the system has the efficiency equal to 1, but we understand that efficiency cannot be 1 because there are overheads.

As we are increasing number of processors there are more interaction between the processors in case of a distributed memory system, in case of shared memory system there are more risk conditions, there are more false sharing and cache coherency protocols are much required and therefore, there will be certain overhead .There will be also be some overhead due to improper load balancing and efficiency will fall down.

So, if we are increasing number of processors, we will not see that the speed is increasing by the same factor. We can see efficiency is a function of both problem size as well as number of processors and we will see how.

That means, if we have a large problem whatever is the efficiency if we increase the problem to a larger extent efficiency will probably increase. If we keep the number of processor fixed, and if we increase the number of processors for one particular problem, we will see the efficiency will fall down and let us see it in detail.

So, efficiency we have seen can be expressed as speed up divided by number of processors. What is speed up? Speed up is the ratio of the computational time required by sequential processor or sequential program divided by the computational time required by the parallel program.

And this is also inverse of the ratio of speed of these two programs ratio; that means, speed up is ratio of the speed of parallel program divided by the and the ratio of the ratio of the speed of the parallel program and speed of the sequential program.

$$E = \frac{S}{p} = \frac{Ts}{pTp}$$

Now, if you understand what is pTp? pTp is the cost of parallel computation and therefore, pTp is the sum of the sequential computing time and the computational over head? So, we write

$E = \frac{S}{p} = \frac{Ts}{pTp} = \frac{Ts}{To+Ts} = \frac{1}{1+\frac{To}{Ts}}$,where $T_{o:\ overhead}$

This $\frac{To}{Ts}$ is the ratio of over head divided by sequential computing time therefore, we can see efficiency is a ratio is a function of the ratio of overhead and sequential computing time.

What is overhead over head? Why does overhead occur? Overhead occurs mainly due to interaction between the processors or data transfer and load balancing. So, as we increase number of processors overhead will increase because more processors are there, more data transfer time will be required and more processors are there. So, load balancing effort and latency due to improper load balancing will increase.

So, this becomes a function of the number of processors and T_s is the computing time for a sequential program and T_s is simply determined by how large is our problem. So, you can see efficiency is a function of the ratio of overhead and sequential computing time or is a function of the problem size and number of processors.

And we can see that efficiency is the function of ratio of over head and sequential computing time and therefore, it is a ratio of it is a function of problem size and number of processors we can understand from here that overhead is related with the number of

processors, as we are increasing number of processors overhead will increase sequential time is related with the size of the problem. As the large as the problem is larger the time to compute the solution in a single processor is also more.

So, speed up =number of processors X efficiency simply we can write.

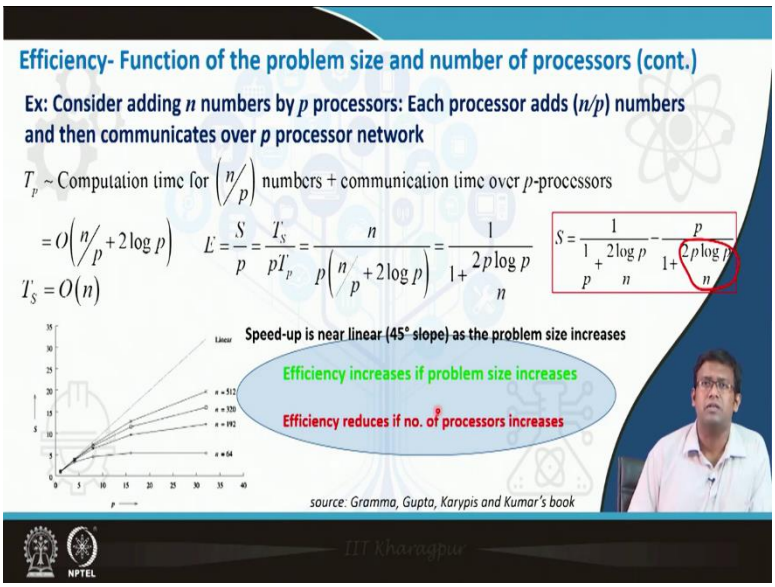
$$S = pE = \frac{p}{1 + \frac{To}{Ts}} = \frac{1}{\frac{1}{p} + \frac{To}{pTs}}$$

And now we can see that in case the overhead is very small say in case To is a small number in case it goes to 0. Speed up becomes 1 by 1 divided by 1 by p or speed up is equal to p in that case speed up is equal to p.

That means, we are increasing the number of processors and we are getting the ratio of the computing time for that parallel problem and the sequential computing time which is equal to the number of processors. So, if you are using five processors the computing time is one fifth of the sequential time and which is an ideal system which is an efficiency one system.

But it never happens because S never equal to p because To is never is equal to 0. So, there is a significant amount of over head and it leads to reduction in speed up.

(Refer Slide Time: 13:19)



So, we consider an example to look to look these things into detail, we consider addition of n numbers by p processors and n is divisible by p. So, each processor adds n/ p numbers and then communicate its sum over this p processor network and after this communication one computer picks on processor picks up these values and it adds which are local additions which are coming from all different processors.

So, computing time in a parallel computing setup will be computing time for n / p numbers. So, what is adding n/ p numbers? The computing time will be of the order of n/ p, n/ p floating point operations will be there and communication time over p processors because each processor has its local sum which is communicated and p processors are doing these things.

This communication time sending data by a processor and receiving by another processor over a p processor network it can be shown that this is of the order of 2log p. Therefore, the total time requirement becomes of order of (n/p+ 2 log p) in case of parallel computing of summation of n numbers in a p processor network. Now if we try to do it in a single processor sequential system there are n numbers, they will be added there will be n floating point operation therefore, computing time will be of the order of n.

So, what will happen to the efficiency? That is $E = \frac{S}{p} = \frac{Ts}{pTp}$ and we can see efficiency is $\frac{1}{1+\frac{2plogp}{n}}$. And speed up will be efficiency multiplied by p which is $\frac{p}{1+\frac{2plogp}{n}}$. This is we have not considered load balancing overhead we are only considering the communication overhead because this is a completely load balanced problem.

So, this $2plogp$ term is the overhead term here. In case this overhead becomes 0 speed up is equal to p a perfectly scaled up system efficiency is equal to 1. When this overhead can be 0? This overhead can be 0 if p is equal to 0 it can tend to 0, but there is no meaning of that problem.

But the overhead also can be small in case n is large. So, if p is equal to 1 over head will be 0, but that is a sequential problem it does not have any over head. But if n is large if you are solving for large number of numbers to be added, the problem size is a itself large, then this number will be a small number and s will tend to p.

So, if we are solving in single processor overhead is 0? But there is no point in this of discussing single processor parallel system. If you are solving for a large problem in small number of processors then this will be 0 the over head will tend to 0. So, we have taken here from Grammas book how overhead varies with number of processors.

And we can see that as for one particular problem as we increase p, p log p increases rapidly and over head becomes constant and if over head is large therefore, speed up becomes almost constant even if we are increasing more number of processors you can see here even if here increasing more number of processors for a small problem we are not seeing any change in speed up.

For high for a larger problem there is some change in speed up; however, speed up does not increase much if we are increasing the number of processors. Initially it increases, but then it becomes almost flat.

But if we keep on increasing the problem size keeping the same number of processors, say for processor number 16 if we keep on increasing the problem size, we can see this speed up is increasing because as n is increasing over head is reducing or the effect or the effect of overhead in term efficiency is increasing. The effect of overhead in scalable scalability or speed up of the system or calculation of efficiency of the system is reducing as the problem size is increasing. Overhead is plog p which is fixed which is only increasing with number of processors.

But as the problem size is increasing its effect in calculating efficiency and speed up is reducing because n is increasing. So, with increase in n we get better speed up. The same thing happens for efficiency therefore, we can say that speed up is near linear as the problem size increases.

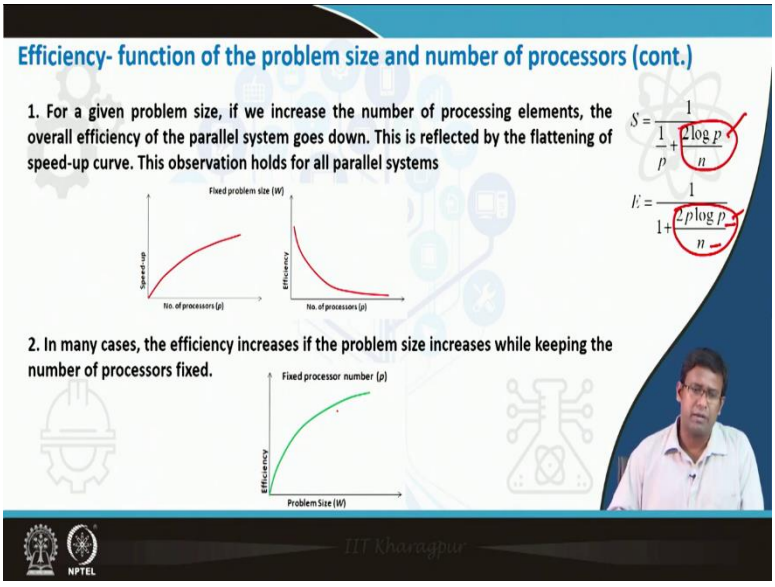
If we have if we take a much larger problem, if we take n is equal to say 10,000 or even more than that n is close to a million, then we will see that this curve has become much flatter. So, speed up in is becomes more linear or follows 45-degree slope as the problem size increases. Efficiency increases if problem size increases. We can find also here as speed up increases' efficiency approach the highest efficiency point which is one.

But efficiency reduces, if number of processor increases. Speed up becomes flat the slope of speed up reduces, efficiency speed up divided by p therefore, as number of processors

increases speed up becomes flatter and s/ p will be a smaller number. So, efficiency reduces as number of processors increases.

So, we can summarize these that efficiency is a function of problem size and number of processors. As the problem size increases efficiency is more as the number of processors increases efficiency is small.

(Refer Slide Time: 19:59)



We look into a little more detail that for a given problem size if we increase the number of processing elements the overall efficiency of the parallel system goes down. Given a problem as we keep on increasing the number of processors, we can get better speed up, but the speed up curve will be flattened the speed up will not increase as much as the number of processors are being increased.

This observation holds for all parallel system. Even if speed up is increasing the rate of increase of speed up or speed up divided by number of processors s by p that is not increasing, that is falling down. If therefore, efficiency is reducing as you are increasing number of processors for a given problem size and single processor problem is the best efficient problem, it always has efficiency is equal to 1.

But as we again if we go to two processors, the computational time is less speed up is if speed up is more than 1. However, speed up by efficient number of processors when efficiency is reducing. Because speed up is not exactly following 45-degree line had speed

up followed 45-degree line efficiency would have been constant or 1, but it is not following a 45-degree line.

Again, due to the fact that there are overheads and the overheads are increasing as we are increasing number of processors. So, we can see that speed up, though it increases it becomes flattened down and there can be cases where speed up is reducing actually you have seen in the last problem for a very small problem size, if you are using large number of processors, speed up is reducing. Efficiency is also reducing almost asymptotically.

So, we can see that as number of processors are increasing these terms ,2 log p here and 2p log p here, these terms increase, keeping the problem size fixed these and therefore, efficiency falls downs, not necessarily it will fall down, but it slows down.

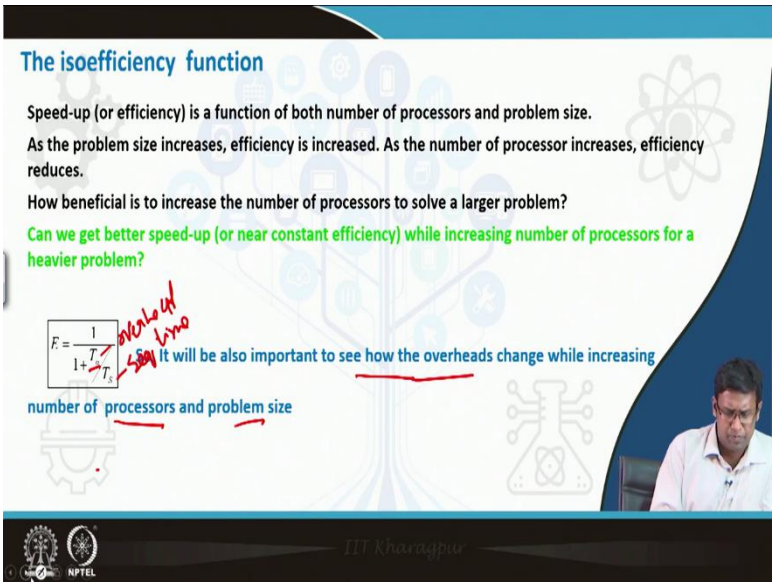
In many cases efficiency increases if the problem size increases while keeping the number of processors fixed. If the number of processors is fixed, but we are increasing the problem size, efficiency is probably increasing and why is it increasing? Because the ratio of the overhead divided by the number of processors which comes in the efficiency calculation that is reducing.

As we are increasing the ratio of the overhead divided by the number of problem size that is reducing. As we are increasing the problem size this ratio is coming down and therefore, efficiency is increasing. So, there can be cases where converse is true, but for first part that if you increase number of processors efficiency will reduce this is universally true.

But if you increase number of process if you increase the problem size in general or for most of the cases, the efficiency will increase. There can be some contrary examples, but in most of the cases it holds, that if you increase the problem size keeping the number of processor fixed efficiency increases.

So, we can understand that for a larger problem with the same number of processors efficiency will be more. For the same problem with increased number of processors efficiency will be less. Now, if we have to solve a larger problem should we increase more processors or not in order to get same efficiency? And that becomes a very important question here. To address that question, we also need to know about iso efficiency function.

(Refer Slide Time: 23:50)



Speed up or efficiency is a function of both number of processors and problem size. So, if we have a larger problem should we use a greater number of processors? That is the question. As the problem size increases efficiency is increases, as the number of processors increases efficiency is reduced.

So, how beneficial is to increase the number of processors to solve a larger problem? If we have a larger problem if we are using same number of processors, we are getting better efficiency. In case you want to have same efficiency, better efficiency means the speed up increases. If we have to use same s/ 2 ratio will increase a greater number of processors and we will get better speed up in the similar ratio.

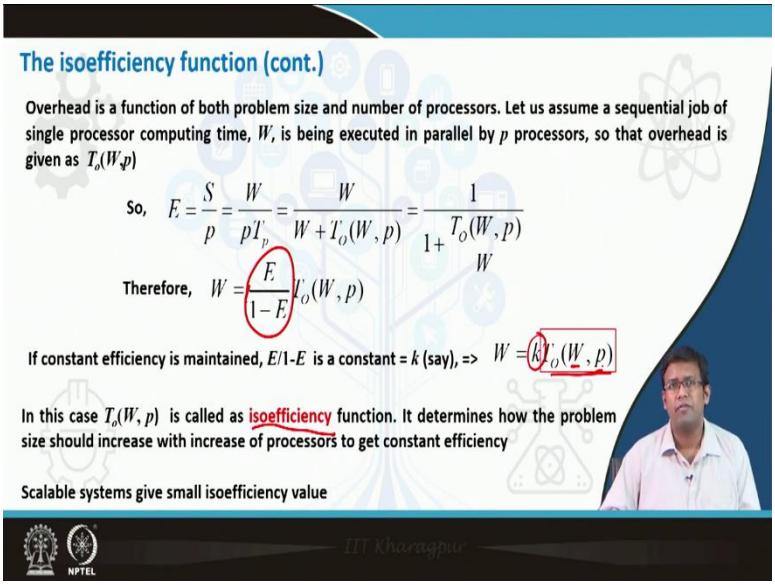
Should we use a greater number of processors if we have a larger problem? For the same problem if we use a greater number of processors efficiency will fall down speed up will be flattened. We are always using high performance computing architecture to solve large problems

From one problem we go to a larger problem will it be judicious decision to increase a greater number of processors that becomes the question and what is our optimal function here? That we want to keep the efficiency fixed we do not want to reduce in terms of efficiency, because we discussed about costs, overheads etcetera. We see that it is important to maintain the efficiency of the computation.

Can we get better speed up while increasing the number of processors for a heavier problem or near constant efficiency, can you get that? And what is the parameter by which we can decide this? So, you know efficiency is $\frac{1}{1+\frac{To}{Ts}}$ where To is the overhead time and Ts is the sequential time that if we solve it in single processor Ts will be the time.

So, it will be important to see how overhead To changes while increasing the number of processors and the problem size.

(Refer Slide Time: 26:17)



Overhead is a function of both problem size and number of processors, let us assume a sequential job which is running in a single processor, and the single processor computing time is W, that is being executed in parallel this job will be executed in parallel by p processors. And the overhead is a function of the main problem size W and the number of processors. So, over head is To(W, p). So, efficiency is $E = \frac{S}{p} = \frac{W}{pTp} = \frac{W}{W+To(W,p)}$, pTp is the cost which is sequential computing time plus over head.

So, efficiency is $\frac{1}{1+\frac{To(W,p)}{W}}$. Therefore, the sequential computing time W can be written as $\frac{E}{1-E}To(W,p)$. So, over head can actually be expressed as a combination of an implicit function with efficiency and the main problem size.

If we have to maintain a constant efficiency this $\frac{E}{1-E}$ term, if E is constant this becomes constant (k say), therefore, W is equal k $T_o(W, p)$. T_o is the overhead which is the function of W or the main computing time of the sequential job and p or number of processors. W is a sequential computing time which is constant and k is also constant.

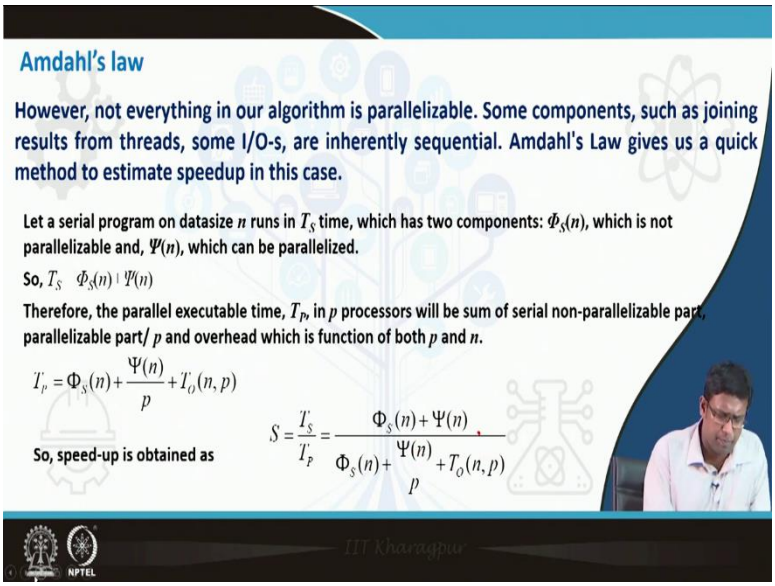
Therefore, for a system where $\frac{E}{1-E}$ is constant ,$T_o(W, p)$ must be a constant and what is $T_o(W, p)$? What is the function which relates overhead with number of processors and the problem size? This function is known as isoefficiency function. In this case if we can keep k is equal to constant ,efficiency is equal to constant by increasing number of processors, then $T_o(W,p)$is called as an isoefficiency function. It determines how the problem size should increase with increase in processors so, that efficiency or k remains constant .In case I am changing p how W will change, so, that this function k will remain constant.

And it is also seen that for constant efficiency systems or scalable systems it has near constant efficiency systems, this value is small or over head is small. So, if we have small over head it is following near 45-degree slope in terms of speed up and efficiency is nearly constant. This function which relates over head with the problem size and the number of processors is known as isoefficiency value.

Well so, we can do a numerical experiment or we can do some analysis on the algorithm and find out how overhead is related with number of processors and the size of the problem and find out the isoefficiency function in certain cases and that can help us to determine what should be the optimum number of processors for a problem.

If you are increasing the problem size, if you are increasing W how judicious is it to increase speed?

(Refer Slide Time: 30:11)



Now the next question which comes to us is that even after doing all these things what is the best efficiency that we can get and for that we use Amdahl's law? That everything we considered that there are certain overheads which will reduce the efficiency ,and efficiency will increase if you have larger problem, and efficiency will reduce if you increase number of processors fine.

Now, what is the best possible efficiency so, that we try to achieve that efficiency? So, now, if we look into a parallel algorithm, we understand that the entire algorithm is not parallelizable, some components such as joining results from threads like, when we are calculating maximum out of n numbers in p processors, each local n/ p max values will be joined.

Or writing or reading from a file some I O-s they are inherently sequential that joining the thread results or reading from a file , doing the load balancing, deciding how many data size data points will go to which processors, all these things they are inherently sequential, they cannot be done in parallel. Either one computer will do it and let others know or all the computers have to repeat the same step. So, there are always some sequential part in an algorithm which we cannot parallelize.

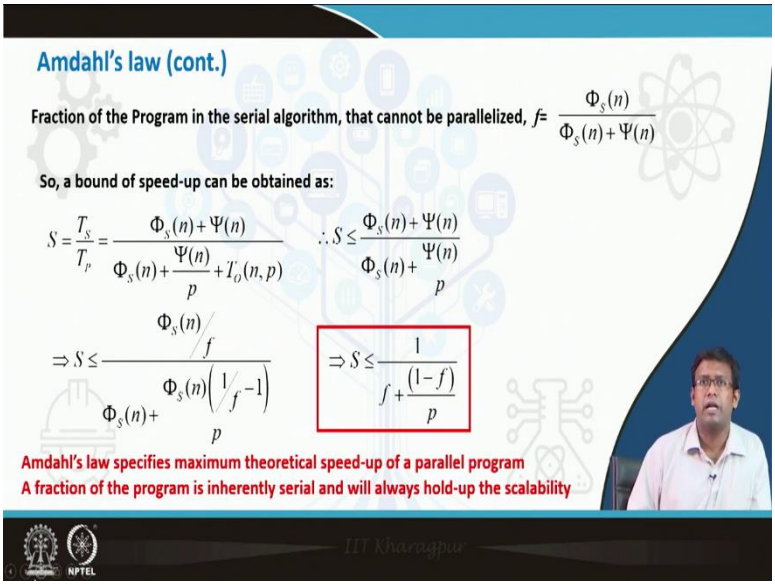
So, Amdahl's law gives us a quick method to estimate the speed up in that case. Let a serial program of data size n runs in T_s time, the total time is T_s for a sequential program and it has two components, one part is not parallelizable essentially sequential which is $\phi_s(n)$ and another part is parallelizable which we write $\psi(n)$.

So, when we paralyze it, we cannot do anything with ϕ_s, this is sequential time in any parallel algorithm this time will be present, but this ψ it can be distributed in many processors and we can reduce this time. So, the total time for this sequential program is $\phi_{s+}\, \psi(n)$. The parallel execution time T_p , will be sum of the serial non parallelizable part ϕ_s ,it will be always there and it cannot be parallelized and then the parallelized part can be divided by p if we are using p processors, and also there is an overhead. So, the parallel time will be

$$\phi_s(n) + \frac{\psi(n)}{p} + To(n,p)$$

because this is the parallel part of the algorithm which can be reduced by using multiple computers and $\frac{\psi(n)}{p}$ plus and overhead. So, speed up is $\frac{Ts}{Tp} = \frac{\phi_s(n)+\psi(n)}{\phi_s(n)+\frac{\psi(n)}{p}+To(n,p)}$

(Refer Slide Time: 33:20)



Now, we assume that the time requirement for sequential part of the program is f for the entire program time. The total time of the sequential run is

$$f = \frac{\phi_s(n)}{\phi_s(n) + \psi(n)}$$

f is the ratio of the or the fraction of the program of the serial algorithm which cannot be parallelized; is the ratio of the sequential time divided by total time which will help us to parameterize the previous expression.

So, the speed up bound we can obtain as that speed up $\frac{Ts}{Tp} = \frac{\phi_s(n)+\psi(n)}{\phi_s(n)+\frac{\psi(n)}{p}+To(n,p)}$, $To(n,p)$ is always greater than 0 over heads is always greater than 0,so speed up is bounded as

$$S \leq \frac{\phi_s(n) + \psi(n)}{\phi_s(n) + \frac{\psi(n)}{p}}$$

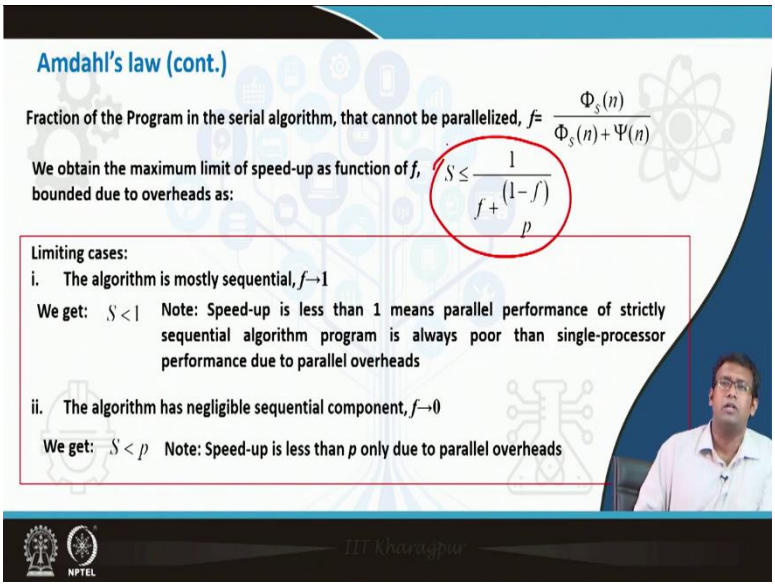
. Now we know that $\phi_s(n) + \psi(n)$ is equal to $\frac{\phi_s}{f}$. So, we substitute all the terms and we get

$$S \leq \frac{\frac{\phi_s(n)}{f}}{\phi_s(n) + \frac{\phi_s(n)(\frac{1}{f} - 1)}{p}} \Rightarrow S \leq \frac{1}{f + \frac{(1-f)}{p}}$$

So, if f is the sequential fraction of the program with p number of processors, this is the maximum speed up that we can get. Due to overheads ,speed up is less than equal to that we cannot get we cannot think of getting higher speed up, and our attempts can be only made to get speed up as close to this particular value and this is known as Amdahl's law which gives us maximum theoretical speed up for a parallel problem.

A fraction of the program is inherently sequential and it always hold up the scalability.

(Refer Slide Time: 35:30)



So, we got that $S \leq \frac{1}{f+\frac{(1-f)}{p}}$. It is usually less than that because overhead is always positive and so, this is bounded due to overhead. The limited cases are one is that the algorithm is mostly sequential.

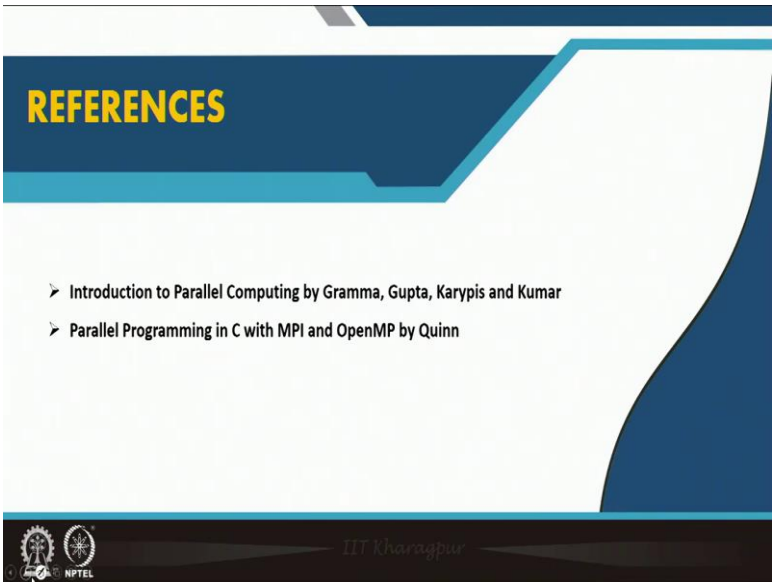
Say we are computing Fibonacci series we cannot parallelize it, even if you are using multiple computers all have to do same things or every computer has to sit idle except one computer which will process it. So, if the sequential part is very close to 1 in if is equal to 1 this term will be 0. So, speed up will be less than 1 because there is an over head.

So, speed up is less than 1. So, it is a sequential program, but if we run it in parallel environment speed up will be less than 1. As many processors as we are using will not see any benefit and speed up is always smaller than 1 .Why it is not 1? Why it is less than 1? Because there are over head. So, if we are using parallel environment multiple computer processors are there, they are interacting in between them and that is adding to over heads.

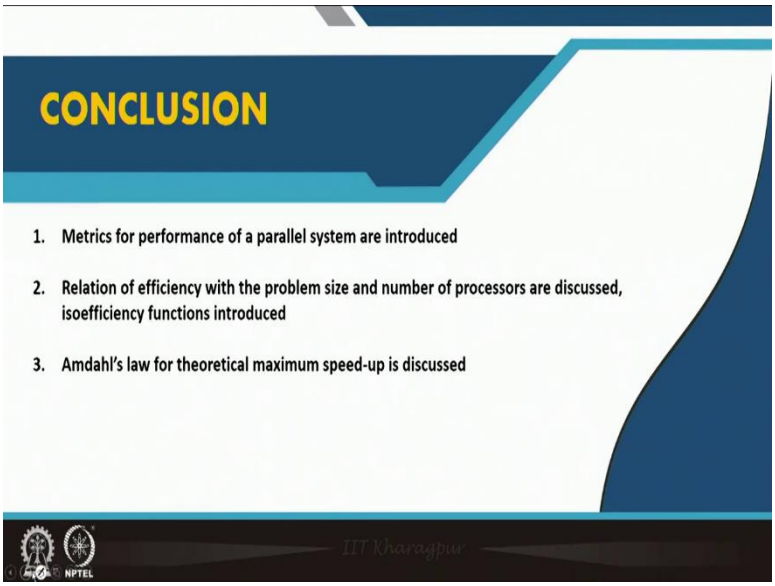
So, that is the most and beneficial par. So, the speed up is less than 1 means parallel platform of strictly sequential algorithm program is always poor than single processor performance due to parallel overheads. In case the algorithm has negligible sequential component, almost the entire program can be parallelized f goes to 0, we get speed up is less than p, S is less than p.

So, speed up is less than p is not exactly p ,due to parallel overheads and in between we will get a value of speed up which is$\frac{1}{f+\frac{(1-f)}{p}}$. So, if we know the algorithm if we can do an analysis and find out how much sequential steps are there and how many parallelizable steps are there, we can find out what is the maximum what is a maximum bound on its parallel speed up and that is what we can try to achieve and this is what is specified as Amdahl's law. It is a very important law in parallel programming.

(Refer Slide Time: 37:51)



(Refer Slide Time: 37:55)



So, these are the references we got here and through this discussion we noted down the metrics for performance of parallel system, saw the relation of efficiency with the problem size and number of processors and looked into isoefficiency functions.

And Amdahl's law is discussed which gives us the theoretical maximum speed up for any algorithm. With this we can start discussing on parallel programs and we will start discussing on open mp next then MPI followed by cuda discussion. Whenever we will try to parallelize a program and try to see its performance, we have the fundamentals in our

mind and we can estimate how good is our parallel algorithm compared to the best theoretical parallelization possible of that problem.

Thank you.